UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Topic Maps for Specifying Algorithm Taxonomies:

A Case Study using Transitive Closure Algorithms

Vreda Pieterse

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Topic Maps for Specifying Algorithm Taxonomies:

A Case Study using Transitive Closure Algorithms

Vreda Pieterse

Supervisors:
Prof Dr Derrick G. Kourie
Prof Dr Dr Bruce W. Watson
Dr Loek G.W.A. Cleophas

Submitted in fulfillment of the requirements for the degree
Philosophiae Doctor (Computer Science)

Faculty of Engineering, Built Environment and Information Technology
Department of Computer Science
University of Pretoria
South Africa
December 2016

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

Printed by STN Printers,
Pretoria

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Topic Maps for Specifying Algorithm Taxonomies:

A Case Study using Transitive Closure Algorithms

Vreda Pieterse

Abstract

The need for storing and retrieving knowledge about algorithms is addressed by creating a specialised information management scheme. This scheme is operationalised in terms of a topic map of algorithms.

Metadata are specified for the adequate and precise description of algorithms. The specification describes both the data elements (called attributes) that are relevant to algorithms as well as the relationship of attributes to one another. In addition, a process is formalised for gathering data about algorithms and capturing it in the proposed topic map.

The proposed process model and representation scheme are then illustrated by applying them to gather and represent information about transitive closure algorithms. To ensure that this thesis is self-contained, several themes about transitive closures are covered comprehensively. These include the mathematical domain-specific knowledge about transitive closures, methods for calculating the transitive closure of binary relations and techniques that can be applied in transitive closure algorithms.

The work presented in this thesis has a multidisciplinary character. It contributes to the domains of formal aspects, algorithms, mathematical sciences, information sciences and software engineering. It has a strong formal foundation. The confirmation of the correctness of algorithms as well as reasoning regarding the complexity of algorithms are key aspects of this thesis. The content of this thesis revolves around algorithms: their attributes; how they relate to one another; and how new versions of the algorithms may be discovered. The introduction of new mathematical concepts and notational elements as well as new rigorous proofs contained in the thesis, extend the mathematical science domain. The main problem addressed in this thesis is an information management need. The technology, namely topic maps, used here to address the problem originated in the information science domain. It is applied in a new context that ultimately has the potential to lead to the automation of aspects of software implementation. This influences the traditional software engineering life cycle and quality of software products.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Acknowledgements

This thesis owes its existence to the help, support and inspiration of numerous people.

I would like to express my appreciation and gratitude to Prof. Derrick G. Kourie for the manner in which he guided and supervised me during my research. I am also indebted to Prof Bruce W. Watson who has opened many doors for me. In addition Dr Loek G.W.A. Cloephas deserves special thanks. His support and inspiring suggestions have been extremely valuable.

I would also like to thank the three external examiners; Dr Paul E. Black, Prof J. Andre van der Poll and Prof Stefan Gruner. Their comments and suggestions assisted in the improvement of the quality and clarity of this thesis.

My thanks also extend to the many colleagues, friends and family members who influenced me in many ways during my research.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

Contents

Contents i

List of Algorithms vii

List of Lemmata ix

List of Tables xi

List of Figures xiii

List of Listings xv

1 Introduction 1
- 1.1 Research paradigm . . . 1
- 1.2 Taxonomy-based software construction . . . 1
- 1.3 Research problem . . . 2
- 1.4 Solution space . . . 4
- 1.5 Research products . . . 5
- 1.6 Choosing an illustrative example . . . 6
- 1.7 Intuitive definition of transitive closure . . . 8
- 1.8 Overview . . . 10

I Prologue 13

2 Syntax and Semantics 17
- 2.1 Notation . . . 18
- 2.2 Topic Map Basics . . . 21
- 2.3 Guarded Command Language . . . 32
- 2.4 Lemmata . . . 37
- 2.5 Summary . . . 38

3 Domain Knowledge 41
- 3.1 Symbolic Logic . . . 41
- 3.2 Sets . . . 42
- 3.3 Relations . . . 43

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.4 Properties of relations . . . 47
3.5 Representing Relations . . . 48
3.6 Functions . . . 51
3.7 Summary . . . 52

4 Matrices 53
4.1 Definition . . . 53
4.2 Notation for matrices . . . 53
4.3 Two-dimentional matrices . . . 54
4.4 Submatrices . . . 56
4.5 Partitions . . . 58
4.6 Operations . . . 59
4.7 From relations to Boolean matrices . . . 61
4.8 Relative strength of Boolean matrices . . . 67
4.9 Summary . . . 68

5 Paths 69
5.1 Relational equivalent and length of a vector . . . 69
5.2 Definitions . . . 70
5.3 Concatenation . . . 71
5.4 Cyclic paths and loops . . . 73
5.5 Relation exponentiation and paths . . . 74
5.6 Summary . . . 76

6 Transitive Closure 79
6.1 Formal Specification . . . 79
6.2 Complexity . . . 80
6.3 Mathematical preliminaries . . . 80
6.4 Lemmata involving Kleene closure . . . 83
6.5 Constructing the transitive closure of a relation . . . 84
6.6 Summary . . . 87

II Representing Algorithmic Information 89

7 Representation methods and models 93
7.1 Introduction . . . 93
7.2 Techniques to order programs in a hierarchy . . . 94
7.3 Representation models . . . 97
7.4 Representation . . . 104
7.5 Summary . . . 106

8 Existing algorithm repositories 107
8.1 The algorithm design manual . . . 108
8.2 NIST dictionary of algorithms and data structures . . . 110
8.3 Handbook of exact string matching algorithms . . . 112

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

8.4 The Canterbury algorithm repository . . . 115
8.5 Summary . . . 115

9 Specification of a TM of algorithms 117
9.1 Concepts . . . 119
9.2 Topic definitions . . . 120
9.3 Attributes of core topics . . . 122
9.4 Topics specifying the context of an algorithm . . . 129
9.5 Supporting Information . . . 136
9.6 Summary . . . 144

10 Process model 145
10.1 Activities . . . 145
10.2 Identify an algorithm . . . 146
10.3 Position an algorithm . . . 147
10.4 Correctness assurance . . . 151
10.5 Create a topic in the TM . . . 152
10.6 Specify algorithm attributes . . . 152
10.7 Specify associations . . . 154
10.8 Summary . . . 155

III Transitive Closure Algorithms 157

11 Root algorithms 161
11.1 Problem and problem area . . . 161
11.2 Starting the derivation tree of TC algorithms . . . 162
11.3 The coat technique . . . 164
11.4 The grow technique . . . 167
11.5 Summary . . . 169

12 Using matrices 171
12.1 Adjacency matrices . . . 171
12.2 Transformation of the root coat algorithm . . . 172
12.3 Matrix multiplication . . . 174
12.4 Prosser's algorithm . . . 175
12.5 Transformation of the root grow algorithm . . . 178
12.6 Warshall's algorithm . . . 180
12.7 Summary . . . 184

13 Loop Interchange 187
13.1 Naïve implementation . . . 187
13.2 Martynyuk's algorithm . . . 191
13.3 Summary . . . 195

14 Monitoring change 197
14.1 Baker's algorithm . . . 197

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

14.2 A variant of Prosser's algorithm 201
14.3 Summary 204

15 Loop Fusion **205**
15.1 The fused coat algorithm 205
15.2 A neat derivation of the monitored coat algorithm 208
15.3 Summary 212

16 Loop Tiling **215**
16.1 Specifying a valid ordering 216
16.2 Processing order constraints 217
16.3 Lemmata related to processing order constraints 222
16.4 Tiling algorithm 227
16.5 Position in a derivation tree 230
16.6 Summary 231

17 Concrete Tiling Algorithms **233**
17.1 Warren's algorithm 233
17.2 Verification of Warren's algorithm 236
17.3 Blocked row algorithm 238
17.4 Verification of the blocked row algorithm 243
17.5 Blocked column algorithm 247
17.6 Verification of the blocked column algorithm 253
17.7 Summary 256

18 Short Circuiting **259**
18.1 The short circuit technique 259
18.2 Short circuiting opportunities 260
18.3 The short circuit version of Prosser's algorithm 262
18.4 Other short circuited coat algorithms 264
18.5 Derivation tree of coat algorithms 266
18.6 The short circuit Warshall algorithm 267
18.7 Other short circuit grow algorithms 270
18.8 Derivation tree of grow algorithms 273
18.9 Summary 273

IV Epilogue **275**

19 Conclusion **279**
19.1 Novel contributions 279
19.2 Self-reflection 283
19.3 Future agenda 284
19.4 Final remarks 286

Bibliography **287**

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendices **311**
Appendix A: Symbols 311
Appendix B: Acronyms and abbreviations 315
Appendix C: LTM listings of the TM of TCA 319

© University of Pretoria

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

List of Algorithms

2.3.1 Specifying variables . . . 32
2.3.2 Specifying constants . . . 33
2.3.9 Declaring and calling a function . . . 36
5.4.2 Constructing an acyclic path . . . 74
11.2.2 TC Root Algorithm . . . 163
11.3.1 The root coat algorithm . . . 165
11.4.1 The root grow algorithm . . . 168
12.2.2 Coat algorithm that applies matrix operations . . . 174
12.4.1 Prosser's Algorithm . . . 175
12.5.1 Grow algorithm that applies matrix operations . . . 179
12.6.1 Warshall's Algorithm . . . 181
13.1.3 Naive row-order grow . . . 190
13.2.1 Martynyuk's Algorithm . . . 193
14.1.1 Baker's Algorithm . . . 199
14.2.1 The monitored coat algorithm . . . 203
15.1.1 Fused Coat Algorithm . . . 207
15.2.1 Neat Coat Algorithm . . . 210
16.4.1 Tiling Algorithm . . . 229
17.1.1 Warren's tile function . . . 235
17.3.1 Tile function of the blocked row algorithm . . . 240
17.5.3 Tile function of the blocked column algorithm . . . 250
18.2.1 Algorithm to calculate $M_0 \times M_1$ . . . 260
18.3.1 Short circuit version of Prosser's Algorithm . . . 262
18.6.1 Short Circuit version of Warshall's Algorithm . . . 268

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

List of Lemmata

2.4.2 Illustration of the style used in proofs 38
3.3.4 If *Point* A then $A \circ R \circ A = \langle\, a \mid (a,a) \in A \wedge (a,a) \in R \mid (a,a) \,\rangle$ 45
3.3.5 If *Point* A and $A \not\subseteq R$, then $A \circ R \circ A = \varnothing$ 46
3.3.6 If *Point* A and $A \subseteq R$, then $A \circ R \circ A = A$ 46
3.3.7 $R^1 = R$ 47
3.4.2 Monotypes are Idempotent 48
3.4.3 The intersection of transitive relations is transitive 48
3.4.4 The composition of transitive relation with itself is transitive 49
4.7.2 $\varphi.(R \circ S) = \varphi.R \,\hat{\circ}\, \varphi.S$ 63
4.7.5 $\Phi(R \cup S) = \Phi(R) + \Phi(S)$ 65
4.7.6 $\Phi(E_A) = \mathbb{I}_B$ where $B = \langle a : a \in A : \varepsilon.a \rangle$ 66
5.3.2 Two adjacent paths in a relation can be combined to form a path . 72
5.5.1 $(a,b) \in R^t \equiv \langle\, \exists P \mid P \in U[t+1], p_0 = a, p_t = b \mid \mathcal{R}(P) \subseteq R \,\rangle$ 75
5.5.2 $P \in U[n]$ is an acyclic path in $R \subseteq U \times U \Rightarrow \ell(P) < |U|$ 76
6.4.1 $R \circ (E_\varnothing \circ R)^* = R$ 83
6.4.2 $R \circ (E_U \circ R)^* = R^+$ 84
6.4.3 $h.(A \cup \{u\})$ expressed in terms of $h.A$ and $E_{\{u\}}$ 85
6.5.1 Derivation of $R^+ = R \cup R \circ R^+$ 86
11.3.2 The zero instance of $Y_j = R \circ \langle \bigcup k : k \in \mathbb{N}_j : R^k \rangle$ 166
11.3.3 The unit instance of $Y_j = R \circ \langle \bigcup k : k \in \mathbb{N}_j : R^k \rangle$ 167
11.3.4 $Y_{j+1} = Y_j \cup R \circ R^j$ 167
12.6.4 $(M \times \mathbb{I}_{\{j\}} \times \mathbb{I}_{\{j\}} \times M)[i,k] = M[i,j] \wedge M[j,k]$ 184
13.2.2 Induction proof for Algorithm 13.2.1 (Martynyuk) 194
16.2.4 Compliance with the secondary constraint with respect to m_{xy} implies that all entries in the lower triangle of M and above $\langle\!\langle\, i,j \mid i = y \mid m_{ij} \,\rangle\!\rangle$ is processed before m_{xy} 221
16.3.2 Trivial case of the induction proof of Warren's Lemma 223
16.3.3 Warren's Lemma 224
16.3.4 Agrawal's Lemma 226
17.2.2 Warren complies with the primary processing order constraint . . 237
17.2.3 Warren complies with the secondary processing order constraint in the lower triangle 237
17.2.4 Warren complies with the secondary processing order constraint in the upper triangle 238

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

17.4.2 The Blocked Row Algorithm complies with the primary processing order constraint if both entries on the row are below the diagonal . 245
17.4.3 The Blocked Row Algorithm complies with the primary processing order constraint if the entries on the row are on different sides of the diagonal 245
17.4.4 The Blocked Row Algorithm complies with the primary processing order constraint if both entries on the row are above the diagonal . 245
17.4.5 The blocked row algorithm complies with the secondary processing order constraint in the lower triangle 246
17.4.6 The blocked row algorithm complies with the secondary processing order constraint in the upper triangle 247
17.6.3 The first interim argument to show that Agrawal's Blocked Column algorithm complies with the secondary processing order constraint 254
17.6.4 The second interim argument to show that Agrawal's Blocked Column algorithm complies with the secondary processing order constraint 254
17.6.5 The third interim argument to show that Agrawal's Blocked Column algorithm complies with the secondary processing order constraint 255
17.6.6 Agrawal's Blocked Column algorithm complies with the secondary processing order constraint 255

© University of Pretoria

List of Tables

2.1.5 Examples of the use of Quantification Notation 20
2.2.2 Syntax of TM Topic Entities 23
2.2.3 Description of TM Topic Entities 24
2.2.4 Description of TM Association Entities 27
2.2.5 TM Occurrence Entities 28

9.2.2 Occurrence types of attributes in the TM of algorithms 121
9.3.1 Algorithm attributes 123
9.3.4 Data structure attributes 128
9.4.2 Computational problem attributes 131
9.4.4 Computational complexity attributes 135
9.5.1 Archive attributes 138
9.5.2 Benchmark attributes 141

15.2.2 Program trace of a statement in the neat coat algorithm 211

18.4.1 Short circuit coat algorithms 265
18.7.1 Short circuit grow algorithms 271

A1 Operator Symbols 311
A2 Symbols for special objects 312
A3 Special functions 313
A4 Quantifier symbols 313
A5 Relational Symbols 314
A6 Multiplicity Symbols 314
B1 Institutions and organisations 315
B2 Research fields 315
B3 Authoritative resources 315
B4 Processes and procedures 316
B5 Topic map entities 316
B6 Computational Complexity classes 317
B7 Protocols and standards for knowledge management 318
B8 Mathematical structures 318
B9 Other Acronyms 318

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

List of Figures

2.2.1 High level architecture of topic maps 21

4.4.2 M and submatrices of M shown by extension 57

7.3.2 Two hierarchical taxonomies of one context 97
7.3.3 Taxonomy of tree acceptance algorithms 98
7.3.4 Merging the taxonomies in Figure 7.3.2 to form a semi-lattice . . . 99
7.3.5 The taxonomy in Figure 7.3.4 as a complete lattice 100
7.3.6 Types of relationships allowed between objects in a thesaurus . . . 101
7.3.8 Screen shot of a search for "thesaurus of algorithms" 103

8.1.1 The input and output of the TC problem 108

9.1.1 Relations between algorithm metadata classes 118
9.4.3 Computational complexity classes 133

10.1.1 The process to add an algorithm to the topic map 146
10.3.1 A derivation path 148
10.3.4 Extending a branch 149
10.3.5 Strating a sub-branch 150
10.3.6 Alternative derivation paths 151

11.1.1 The TCRoot and TCProblem topics 162
11.2.3 Basic derivation hierarchy of TC algorithms 164
11.3.1 Topics related to the root coat algorithm 165
11.4.1 Topics related to the root grow algorithm 168

12.2.1 Topics related to a coat algorithm that applies matrix operations . . 173
12.4.1 Topics related to Prosser's algorithm 176
12.4.2 Derivation tree of Prosser's algorithm 177
12.5.1 Topics related to a grow algorithm that applies matrix operations . 179
12.6.1 Topics related to Warshall's algorithm 181
12.6.2 Processing using Warshall's algorithm 182
12.6.3 Derivation tree of Warshall's algorithm 183

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

13.1.1 One iteration of the naïve grow algorithm 188
13.1.2 Topics related to the naïve grow row algorithm 189
13.1.3 Derivation tree of the naïve grow algorithm 191
13.2.1 Topics related to Martynyuk's algorithm 192
13.2.4 Derivation tree of Martynyuk's algorithm 195

14.1.1 Topics related to Bakers's algorithm 198
14.1.4 Derivation path of Baker's algorithm 200
14.2.1 Topics related to the monitored coat algorithm 202
14.2.4 Derivation tree showing Prosser, Warshall, Martynyuk and Baker . 204

15.1.1 Topics related to the fused coat algorithm 206
15.1.3 Derivation tree of the fused coat algorithm 208
15.2.1 Topics related to the neat coat algorithm 209
15.2.4 Derivation tree showing new variants of Prosser's algorithm 213

16.2.1 Primary constraint for processing an entry m_{xy} 218
16.2.2 Secondary constraint for processing an entry m_{xy} 219
16.2.3 Column constraint for processing an entry m_{xy} 220
16.2.4 Submatrices that has to be processed before processing m_{xy} 221
16.4.1 Topics related to the tiling algorithm 228
16.5.1 Tiling algorithm is an abstraction of Warshall's algorithm 230
16.5.2 Tiling algorithm is an alternative to Warshall's algorithm 231
16.5.3 Alternative derivation paths of Warshall's algorithm 231

17.1.1 Topics related to Warren's algorithm 234
17.1.3 Derivation tree of Warren's algorithm 236
17.3.1 Topics related to Agrawal's blocked row algorithm 239
17.3.1 Submatrices formed by the blocked row algorithm 242
17.3.3 Derivation tree of the blocked row algorithm 244
17.5.1 Topics related to Agrawal's blocked column algorithm 248
17.5.1 Subdivision of a vertical section formed by the blocked column algorithm 249
17.5.2 Submatrices of the partition formed by the blocked column algorithm 250
17.5.5 Derivation tree of Agrawal's blocked column algorithm 252

18.3.1 Topics related to the short circuit version of Prosser's algorithm . . 263
18.3.2 Derivation tree of the short circuit version of Prosser's algorithm . 265
18.5.1 Derivation tree of the coat algorithms 266
18.6.1 Topics related to the Short Circuit Warshall algorithm 267
18.6.2 Derivation tree of the short circuited Warshall algorithm 271
18.8.1 Derivation tree of the grow algorithms 272

© University of Pretoria

List of Listings

9.2.1 Core topics of the TM of A 120
9.2.3 Association type topics of the TM of A 121
9.4.2 The TC problem 132
C1 The author of this topic map (TM) 319
C2 The problem area of the transitive closure problem 319
C3 The problem area of the matrix multiplication problem 319
C4 The matrix multiplication problem 319
C5 The transitive closure problem 320
C6 Boolean matrix and its ascendants 321
C7 Basic algorithmic techniques applied by TC algorithms 322
C8 General algorithmic techniques 322
C9 Algorithmic techniques specific to TC Algorithm optimisation . . . 323
C10 Tiling strategies applied by implementations of the tiling algorithm 323
C11 Algorithmic techniques applied when operating on matrices 324
C12 The root algorithm solving the TC problem 325
C13 The root coat algorithm 325
C14 The root grow alorithm 325
C15 Matrix coat implementation 326
C16 Matrix grow implementation 326
C17 Prosser's algorithm 327
C18 Short circuited version of Prosser's algorithm 328
C19 Warshall's algorithm 329
C20 Short circuited version of Warshall's algorithm 330
C21 Fused coat algorithm 331
C22 Naïve grow implementation in row order 332
C23 Skeleton tile algorithm 332
C24 Martynyuk's algorithm 333
C25 Baker's algorithm 334
C26 Monitored coat algorithm 335
C27 Neat coat algorithm 336
C28 Warren's algorithm 337
C29 Agrawal's blocked row algorithm 338
C30 Agrawal's blocked column algorithm 339
C31 The complexity of Prosser's algorithm 340
C32 The complexity of Warshall's algorithm 340

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

C33 The complexity of Martynyuk's algorithm 340
C34 The complexity of Baker's algorithm 340
C35 The complexity of the fused coat algorithm 341
C36 The complexity of the monitored coat algorithm 341
C37 The complexity of the neat coat algorithm 341
C38 The complexity of Warren's algorithm 341
C39 The complexities of Agrawal's blocked row algorithm 342
C40 The complexities of Agrawal's blocked column algorithm 342
C41 Input data suitable to use for several TC implementations 343
C42 Implementations of Prosser's algorithm 343
C43 Short circuited implementations of Prosser's algorithm 344
C44 Implementations of Warshall's algorithm 345
C45 Short circuited implementations of Warshall's algorithm 346
C46 Implementations of Martynyuk's algorithm 347
C47 Short circuited implementations of Martynyuk's algorithm 348
C48 Implementations of Baker's algorithm 349
C49 Short circuited implementations of Baker's algorithm 350
C50 Implementations of the monitored coat algorithm 351
C51 Short circuited implementations of the monitored coat algorithm . 352
C52 Implementations of the fused coat algorithm 353
C53 Short circuited implementations of the fused coat algorithm 354
C54 Implementations of the neat coat algorithm 355
C55 Short circuited implementations of the neat coat algorithm 356
C56 Implementations of Warren's algorithm 357
C57 Short circuited implementations of Warren's algorithm 358
C58 Implementations of Agrawal's block row algorithm 359
C59 Short circuited implementations of Agrawal's block row algorithm 360
C60 Implementations of Agrawal's block column algorithm 361
C61 Short circuited implementations of Agrawal's block column algorithm 362

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 1

Introduction

This chapter positions the research presented in this thesis; highlights premises advanced and describes the processes adopted. This study is part of research to promote a software construction approach known as *Taxonomy-based software construction*. After briefly stating the research paradigm in Section 1.1, Section 1.2 gives an overview of this overarching research project. Section 1.3 describes the global problem addressed by this research, demarcates the specific domain within which a solution is proposed and mentions sub-goals pursued towards the main aim of this thesis. The research conducted here is situated in the field of *knowledge organisation* briefly discussed in Section 1.4. Section 1.5 describes the products that are delivered in this thesis. Section 1.6 justifies the selection of the topic used in this thesis to illustrate the use of the delivered artefacts and demarcates the scope of this example. The chosen example deals with the transitive closure problem. The concept *transitive closure* is introduced in Section 1.7. The concluding section of this chapter is a broad overview of the content of the remainder of this thesis.

1.1 Research paradigm

This thesis follows procedures adopted within the design science research paradigm. The definition of design science research, as introduced to information systems research by Hevner *et al.* [113], with its roots in the seminal work by Simon [225], was applied to craft knowledge in a way that enhances its usability and usefulness.

The guidelines outlined by Gregor and Hevner [104] were followed. Design science requires the creation of an innovative, purposeful artefact for a specific problem domain [8]. The research products created in this thesis apply design science to bring order to knowledge related to algorithms.

1.2 Taxonomy-based software construction

TAxonomy BAsed Software COnstruction (TABASCO) [55, 57] was first adopted at Eindhoven University of Technology as an extension of correctness-by-construction (CbC) [56]. Correctness-by-construction (CbC) is an approach for developing al-

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

gorithms inline with rigorous correctness arguments. CbC aids the development of algorithm taxonomies. Different refinements of the algorithms carried out during CbC-based design of algorithms naturally lead to the specification of a taxonomy. The constructed taxonomy provides a classification of the commonality and variability of the algorithm family and, hence, provides deep insights into their structural relationships [54].

TABASCO is an extension of the organisation and taxonomisation of algorithms started by Jonkers [130]. When TABASCO was introduced, the idea to create implemented collections of provably correct algorithms was added to the main aim of supporting industrial software construction [253]. Apart from realising the increase in the practical usability of research results, it became clear that these collections have wider applicability. The implemented algorithms as well as the information about algorithms in an identified domain that is included in such collections increases its accessibility to a broader user base including practitioners, researchers and students.

The TABASCO activities include gathering and describing information about algorithms. Through these activities the inter-relationship between the different algorithms, and consequently the algorithms themselves, become more accessible. These activities require an in-depth investigation of descriptions and implementations of algorithms.

The research reported in this thesis applies a novel approach to the description of information about algorithms that extends the ways that were previously applied to describe and store this kind of information. This creates opportunities for new ways to discover and gather information. The essential form of the TABASCO method that ensures correctness through correctness preserving transformations of algorithms remains key. The development of a derivation hierarchy of algorithms remains a core activity and serves an essential role to aid the discovery of new algorithms. The importance of providing implementations that can be used in software construction is also honoured. The algorithm collection presented in this thesis thus retains the essential aspects of previous structures while extending them to allow rich information that can be applied in a wider context.

The information presented in this thesis uses a format that is compatible with web ontologies. It adheres to trends driving the semantic web [27] and the internet of things [15]. These technologies are pushing the edge for knowledge representation in the future. If one ignores these technologies in the present, the knowledge created now may soon become inaccessible. The approach advocated here is designed to allow people and software agents alike to use the knowledge which is encapsulated in the collection. The approach provides both a user friendly interface and a machine friendly structure.

1.3 Research problem

The rapid growth of information pose problems

The ease and tempo with which information gets created and distributed in the current age, has brought about its own set of problems. Methods of organising

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

and disseminating information that were able to address the information needs of people in the past may no longer be adequate in the present. The amount of information is overwhelming and the information needs of people are constantly changing. The information itself is also changing. A diversity of new things that need to be described have come into existence. The need for new data formats to accommodate the description of new things may be more challenging than the challenges posed by the increase in the volume of data within the different classes of things. Furthermore, the need to allow for automatic processing of information is becoming more widespread. These developments create the need for innovative support to make sense of the available information and to use it appropriately. The information area dealing with algorithms shares the problems associated with the general explosion of information.

Information about algorithms needs to be organised

Many situations call for support to make sense of available information about algorithms and to simplify the use of algorithmic information. Programmers, researchers and students often require information about algorithms that solve computable problems, yet finding the appropriate information may pose significant challenges. Usually, many different algorithms may be used to solve a given problem. The availing of algorithmic information in a formal structure is useful. Professional programmers need support to select the most appropriate algorithm to be used in a given situation. Researchers and students can gain deeper insight into different aspects of an algorithm by exploring gathered knowledge about the algorithm. A formal structure may contribute to the ease of automatic processing of such information.

Addressing the problems and satisfying the need

This thesis aims to address some of the present and future problems of gathering, storing, accessing, and using information about algorithms. It is granted that this general problem can not be solved instantly or by a single idea.

This thesis explores one possible solution that has the potential to ease some of the mentioned problems. The following are aims of the TABASCO project pursued towards the aim of this thesis:

- to provide means to make algorithms more accessible;
- to describe algorithms using a uniform scientific style;
- to provide a mechanism for the comparison of algorithms;
- to describe a strategy that enables the discovery of algorithms;
- to investigate a specified category of algorithms and provide information about these algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Artefacts that support the organisation of information aimed at meeting the above-mentioned information needs are proposed. The thesis includes an illustration of how these artefacts can be applied to gather, represent and use information about algorithms.

1.4 Solution space

A solution aimed at satisfying the identified information need is created by extending ideas originating in a variety of research areas. A prominent research field which may contribute to pursuit of the goals of this thesis is the domain of knowledge organisation (KO). KO is a mature field that spans various research fields and allows for a multitude of applications. In this thesis different methods and aims of knowledge representation as an aspect of KO are considered. In a field like artificial intelligence, the representation of knowledge aims to support the generation of new knowledge, whereas in a field like information architecture, representation models focus on enhancing access to existing information.

The design of representation models is determined by two things: the aims of KO in different fields; and the content that has to be represented. Knowledge representation models within one field but covering different information domains often differ substantially. This thesis focusses on the highly specialised information domain of algorithms. Here the KO aim is to represent algorithmic information in a way that supports both the discovery and the retrieval of information by people as well as by software agents.

Due to the necessity of organising information in various disciplines, representation models have been developed in different domains. However, these developments have taken place in a scattered and independent fashion and have consequently given rise to a conglomeration of terms that tend to be used inconsistently. Gilchrist [101] states that the words *thesaurus*, *ontology*, and *taxonomy* seem to have significant overlap in meaning and can at times even be used with contradictory meanings. Pieterse and Kourie [199] clarify these and other classification-related terms.

This thesis deals with organisation, representation and manipulation of information from a *knowledge representation and engineering* perspective with the intention of creating a specialised information scheme for storage and retrieval of knowledge about algorithms. The reason for this stems from Hjørland's [114] remark that the lack of interest in specialised information schemes is problematic. He pointed out that information specialists may not have enough insight in the specialised domain while subject specialists may lack knowledge on classification and knowledge organisation. He claims that specialised databases like MEDLINE [179] and PsycINFO [13] have special classification systems that have been developed independently of methodologies of information science and he hints that their designs may be inferior. The design of the information repository presented in this thesis applies methods of information science. In doing so I attempt to avoid mistakes that may occur as a consequence of ignorance.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 7 gives a high level overview of knowledge creation, retrieval and representation. It incorporates fundamental ideas from the disciplines that informed the proposed extension of TABASCO. Various techniques can be applied to address the information needs of various role players involved in information processing. The perspective of popular standards such as Resource Description Framework (RDF) [40, 132] and Web Ontology Language (OWL) [90, 12] is machine- and logic-focussed, and such a perspective is beneficial in the era of big data. In contrast with these techniques and standards, Topic Maps (TMs) [194] put emphasis on the role that people play when dealing with information.

Although adoption of topic maps for representing information has been relatively low, this thesis nevertheless proposes their use as a representation standard for information about algorithms. They address human needs without neglecting the need for automatic processing of information. They therefore seem more powerful and easier to use than alternative representative mechanisms. TMs are, in my view, the most elegant way to represent a web ontology.

1.5 Research products

Design science artefacts can be typified as product components or development process components [201]. In this thesis, instruments of both kinds are created to support the collection, organisation and use of information about algorithms. The full cycle of gathering information, curating the information and distributing the information is covered.

The product component created in this thesis is a core thesaurus of algorithm properties. The design of this artefact is presented in Chapter 9. The management and representation of the information about the algorithms is presented as an application of topic map technology [125]. A *Topic Map* is an ISO standard for describing knowledge structures and associating them with information resources. This is, to the best of my knowledge, the first time that a topic map has been used for the management and representation of information about algorithms.

The process component is the process model presented in Chapter 10. The process model suggests the actions needed when a topic map of algorithms is created or extended. This process is an extension of the methods of taxonomisation previously applied by scholars such as Broy [43], Watson [253], Barla-Szabo *et al.* [20] and Cleophas [60].

The core thesaurus and the process model are evaluated by means of a case study that illustrates their application. The case study itself is an artefact, namely a topic map of transitive closure algorithms, called the TM of TCA, which is presented in Part III of this thesis. The transitive closure (TC) problem as specified in Section 1.7 is the underlying theme of this topic map. New knowledge regarding TC algorithms is created during the construction of this illustrative example. The TM of TCA, once completed may become a knowledge repository that can be explored when learning or conducting research about TC algorithms. It may also be used in software construction.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1.6 Choosing an illustrative example

A *subset* of algorithms solving the transitive closure problem was selected to illustrate the applicability of the research products presented in this thesis.

Cleophas and Watson [56] suggest that when selecting a domain for the application of TABASCO, the problem solved by the algorithms should be general in the sense that the solutions to the problem should have broad applicability. The domain should be mature in the sense that the derivation of algorithms in this domain can be based on a sound theoretical foundation. A variety of algorithms that solve the problem is also required, as the benefit of the method is not obvious when too few algorithms are included in a toolkit designed with TABASCO. The following arguments provide evidence that the problem of solving the transitive closure of a relation complies with these criteria and is therefore a suitable problem to benefit from the application of TABASCO.

1.6.1 Broad applicability of the transitive closure problem

Computing the transitive closure of a relation is an operation underlying many important algorithms, with applications to computer-aided design, software engineering, scheduling, databases and optimising compilers [248]. At the time that Structured Query Language (SQL) became the standard query language for relational databases, the need for solving the transitive closure as a necessary extension to relational query languages was soon observed [31]. The following is a remark about the transitive closure problem made by Dijkstra [76]

> *The question, among other things, hinges upon the assumption that the problem of the transitive closure for directed graphs in which arrows may merge, is – besides being a logical generalization of the tree-traversal – a sufficiently frequently recurring theme to give it the status of "central problem."*

Efficient transitive closure computation has been recognised as a significant subproblem in evaluating recursive database queries, as almost all practical recursive queries are transitive [46]. In compiler construction, the construction of parsing automata often relies on the computation of the transitive closure of symbols for acceptance when grammars may contain chain rules. For some practical problems in the field of syntactic analysis, it is sometimes necessary to find the transitive closure of a relation involving hundreds of nodes. Reachability analysis requires computation of the transitive closure of the connection relations of networks. Reachability is a fundamental problem that appears in several different contexts such as finite-state and infinite-state concurrent systems, computational models like cellular automata and Petri nets, program analysis, discrete and continuous systems, time critical systems, hybrid systems, rewriting systems, probabilistic systems, parametric systems and many more.

There is thus no doubt that the transitive closure problem is a generic problem that is used in a variety of situations to solve a wide spectrum of real world problems.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1.6.2 Theoretical foundation

The theory needed to solve the transitive closure problem is rooted in the mathematical concepts of set theory, relations and functions. These concepts have been studied for decades. The essence of these concepts can be found in *Principia Mathematica*, a three-volume work on the foundations of mathematics, that was published more than a century ago; in 1910, 1912, and 1913. The authors of this great work are the English mathematician and philosopher Alfred North Whitehead (1861 – 1947) and the influential British philosopher and mathematician Bertrand Russell (1872 – 1970).

The transitive closure problem is specific to the mathematical concept of a relation. This concept was defined in the writings of Russell [211] in 1903.

Transitivity is a key property of both partial order relations and equivalence relations. As such the theory of relational algebra is fundamental in understanding some solutions to the transitive closure problem. Relational algebra was first described by the English computer scientist Edgar Frank "Ted" Codd (1923 – 2003) [61, 62]. Relational algebra has well-founded semantics that are used for modeling the data stored in relational databases and defining queries on this data. Relational algebra provides a theoretical foundation for relational databases, particularly for query languages for such databases, chief among which is the Structured Query Language (SQL). Solutions to the transitive closure are considered an essential feature of relational query languages.

These theoretical underpinnings provide a sound and mature mathematical foundation for the transitive closure problem. They can be applied to derive transitive closure algorithms and to prove their correctness, as I do in this thesis.

1.6.3 Availability of variety of algorithms

Interest in algorithms to solve the transitive closure problem contributed to a multitude of publications since 1959. The following is a representative collection in chronological order that is by no means exhaustive.

1959 – 1969 Prosser [203], Roy [210], Moore [173], Baker [17], Martynyuk [162], Warshall [252], Jones [129], Karp *et al.* [135].

1970 – 1979 Purdom [204], Fischer and Meyer [91], Munro [176], Aho *et al.* [6], Tarjan [235], Martynyuk [164], Warren [251], Chandra and Merlin [49], Eve and Kurki-Suonio [87], Schnorr [218], Guibas *et al.* [108].

1980 – 1989 Ebert [84], Schmitz [215], Rao *et al.* [207], Bancilhon [19], Ioannidis [121], Italiano [127], Valduriez and Boral [240], Agrawal and Jagadish [2, 3], Ioannidis and Ramakrishnan [122], Valduriez and Khoshafian [241, 242], Ioannidis and Wong [123].

1990 – 1999 Ullman and Yannakakis [239], Agrawal *et al.* [1], Agrawal and Jagadish [5], Karp [134], Dar and Jagadish [68], Chakradhar *et al.* [48], Dar and Agrawal [67], Ioannidis *et al.* [120], Nuutila [180], Henzinger and King [112], Marchetti-Spaccamela *et al.* [160], King [138].

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2000 – 2009 Agrawal and Jagadish [4], King and Sagert [139], Gibbons *et al.* [100], Sankowski [212], Bender *et al.* [24], Demetrescu and Italiano [72], Baswana *et al.* [23], Demetrescu and Italiano [73], Roditty [208].

2010 – Sankowski and Mucha [213], Wlodzimierz *et al.* [267], Bozga *et al.* [37], Ding *et al.* [81], van Schaik and de Moor [246], Bergmann *et al.* [26], Bielecki *et al.* [28], Cheng *et al.* [53, 52], Niesink *et al.* [178], Łącki [156], Alves *et al.* [9], Konečný [146].

It is clear that sufficient algorithms exist in the domain to justify the application of TABASCO. The scope, however, has been narrowed and its application in this thesis is discussed in the next section.

1.6.4 The scope of the illustrative example

An illustration of the usage of the artefacts that are presented in this thesis is achieved without covering all the algorithms in the chosen domain. Sufficient variety in algorithms is retained despite limiting the investigation to a subset of solutions to the problem. This subset is small relative to the body of published algorithms in this domain.

The algorithms included in the investigation are mostly simplistic solutions that manipulate the adjacency matrix of a relation to determine the transitive closure of the relation. The derivation tree that is constructed in Part III includes nine abstract algorithms in order to show the derivation paths of the concrete algorithms in this tree. A total of twenty concrete algorithms are included of which thirteen are new. The previously published algorithms cover a wide date range from 1959 [203] to 1990 [5]. No distinction is made between highly cited algorithms such as the algorithm by Floyd [92] (showing a citation count of 2289 on 2014-01-14 on Google Scholar[1]) and algorithms that appeared in publications that are not easy to find, such as the algorithm by Martynyuk [164] (showing a citation count of only twelve on the same date on Google Scholar[2]).

Each of the algorithms that are included in the investigation is discussed in great depth. For each, its formal specification is given and its position in the derivation tree is determined. Mathematical arguments regarding its correctness as well as its complexity are provided and implementations are made available.

1.7 Intuitive definition of transitive closure

A relation indicates the way in which two objects are connected. In mathematics a relation, R, between two sets is defined as a collection of ordered pairs containing one object from each set. Suppose object x is from the first set and object y is from the second set. Then the objects are said to be related if the ordered pair (x, y) is in the relation, i.e. if $(x, y) \in R$, also denoted as xRy.

[1] http://scholar.google.com/scholar?cluster=14821974106251291638&hl=en&as_sdt=0,5
[2] http://scholar.google.com/scholar?as_q=&as_sauthors=VV+Martynyuk

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A relation R is said to be *transitive* if it follows from aRb and bRc that aRc. Let R be a relation that describes a specific relationship that may hold between elements of a set U. The transitive closure of this relation R is defined to be the smallest transitive relation on U that contains R. For a relation to be transitive it has to be general enough so that if the relation holds between object x and y also between y and z it is implied that it holds between x and z. The formal definition of the mathematical concept of transitive closure is given in Section 6.1.1. Here the concept is illustrated using real world examples.

The first example is about a relation between airports described in terms the existence of flights between airports. If U is a set of airports and xRy means "there is a direct flight from airport x to airport y", then R is not transitive (because there is obviously not a direct flight between any two airports). The the relation "it is possible to fly from x to y in one or more flights" is, however, transitive. If one can fly from a to b and one can also fly from b to c, it is implied that one can fly from a to c. In fact this relation is the transitive closure of R.

Another example that can be used to explain the meaning of transitive closure is the ancestor relation. If U is the set of humans (alive or dead) and R is the relation "parent of", then R is not transitive. The parent of my parent is not my parent. The smallest transitive relation that contains R is the relation "x is an ancestor of y". Any ancestor of an ancestor of mine is also my ancestor.

An interesting problem regarding relations is about the "to know" relation between people in an arbitrary community. This relation is often used as example to illustrate interesting graphs such as a "k-clique" and a "Caveman graph" [257]. The transitive extension of the "knows" relation is "is connected by a chain of people who know each other" or more concisely (yet less precisely) "is connected".

Research to determine the size of the relation of any one person with his or her acquaintances was done by Gurevitch [109]. A related problem is the small world problem which Travers and Milgram [238] formulated as follows:

> *What is the probability that any two people, selected arbitrary from a large population, such as that of the United States, will know each other?*

The notion that everyone is only six steps away, by way of introduction, from any other person in the world is popularly known as the six degrees of separation hypothesis. It was originally presented by the Hungarian author Frigyes Karinthy in a 1929 short story [133]. In 1967 Milgram [169] conducted an experiment to test the hypothesis. Chains were established by requiring randomly chosen people to pass a message via an acquaintance to a specified target person. He determined that the length of the chain from one person to another between Nebraska and Massachusetts varied from two to ten with the median at five. In 2003 Dodds *et al.* [82] conducted a similar study. They calculated a median chain length between five and seven people. The Facebook era and the rise of social networks means that people are more closely connected than ever before, with four degrees of separation having become the norm. Recently Daraghmi and Yuan [69] observed that on average only 3.868 acquaintances separate any two people on Facebook.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Although the reported experiments were conducted in specified situations, the results were generalised to such an extent that the following is commonly accepted as the truth:

> *Anyone in the world is connected to any other person in the world through a chain of acquaintances that has no more than five intermediaries* [69]

Proponents of this assumption conveniently ignore the possibility mentioned by Milgram [169] that there may be unbridgeable gaps between various groups that may cause the acquaintance circle of one given individual never to intersect with that of another individual. Travers and Milgram [238] report that only 64 of the 296 (21.6%) chains that were started were actually completed in their experiment. Similarly in Dodds *et al.*'s [82] experiment a mere 384 of 24 163 (1.6%) chains reached their targets. There are various reasons why the chains failed to complete. Yet the large number of incomplete chains may be an indication of the existence of disjoint communities. The question of whether the transitive transitive closure of the "is connected" relation includes all the people in the world in one connected unit, or consist of a number of disjoint communities remains unanswered.

1.8 Overview

Apart from this chapter (Chapter 1) and the concluding chapter (Chapter 19), the thesis comprises three parts; the prologue, a part describing generic artefacts and a part illustrating the use of these artefacts.

Part I: Prologue

The first part deals with conventions and domain knowledge to set the scene for the remainder of the thesis. The following is a broad overview of the chapters comprising Part I

Chapter 2	Specification of notational conventions used in this thesis.
Chapter 3	A discussion of the mathematical basis of the transitive closure problem.
Chapter 4	Theoretical foundation regarding matrices and their use for representing binary relations.
Chapter 5	Description of key concepts regarding paths in relations.
Chapter 6	Formal discussion of the concept of the transitive closure of a relation.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part II: Representing Algorithmic Information

The second part presents the outcome of the research in terms of a description of the research products as well as how they were created.

Chapter 7	Overview of research related to the capturing and curation of algorithmic information.
Chapter 8	Description of existing repositories of algorithmic information that informed the design of the resource description vocabulary presented and used in this thesis.
Chapter 9	The formulation of a resource description vocabulary for algorithmic knowledge.
Chapter 10	A description of the process that is applied in this thesis to generate, capture and curate algorithmic information.

Part III: Transitive Closure Algorithms

The third part is a case study using transitive closure algorithms to illustrate the application of the designed artefacts.

Chapter 11	The root algorithm and two derived abstract algorithms.
Chapter 12	Concrete algorithms that are derived using adjacency matrices to represent data.
Chapter 13	Algorithms that are derived by applying loop interchange.
Chapter 14	Algorithms that are derived by applying change monitors.
Chapter 15	Algorithms that are derived by applying loop fusion.
Chapter 16	An abstract algorithm that is derived by applying loop tiling.
Chapter 17	Concrete algorithms that are derived by specifying tiling strategies.
Chapter 18	Algorithms that are derived by applying short circuiting.

Part IV: Epilogue

The concluding part is only one chapter:

Chapter 19	Summary of achievements, future research agenda and personal reflection.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendices

Appendix A	Table of symbols used in the thesis, their meaning and where they are introduced.
Appendix B	Table of acronyms and abbreviations used in the thesis, their meaning and where they are introduced.
Appendix C	Specification of the topic map of transitive closure algorithms using linear topic map notation.

© University of Pretoria

Part I

Prologue

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Prologue Overview

This part provides theoretical preliminaries to prepare the reader for the main matter of this thesis. It also establishes the mathematical foundations of the concept of the transitive closure of a relation. The reader is expected to have a basic knowledge of the relevant mathematical terms and their meanings.

Chapter 2 states the conventions and tools that are used in this thesis. Some of the symbols, terms and concepts are defined while others are deemed sufficiently basic to need no explication. A new notation for sequences is introduced. Custom alterations to linear topic map (LTM) notation as well as to guarded command language (GCL) are explained.

The arguments used when describing the construction of the Topic Map in Part III assume a working knowledge of the mathematical problem domain of transitive closure. For this reason, Chapter 3 provides adequate mathematical domain-specific knowledge to follow the arguments relating to transitive closures in the remainder of this part as well as in arguments presented in Part III.

The algorithms that are discussed in Part III use square Boolean matrices as their data model. For this reason it is essential to have a thorough understanding of the representation of Boolean matrices and the operations to manipulate them. This is provided in Chapter 4.

Many of the arguments about the correctness of the algorithms that are discussed in Part III are based on the concept of a path as a special kind of Boolean relation. This concept, as well as a number of lemmata are presented in Chapter 5. These lemmata are used in arguments in the final chapter in this part. They are also used in arguments presented in Part III.

In the literature, mathematical formulae to calculate the transitive closure of a Boolean relation are often presented without proof or simply stated as definitions. In contrast, the final chapter in this part presents — to my knowledge, for the first time — the derivation of two of these formulae. This mathematical derivation establishes the correctness of the formulae. It is an essential cornerstone of the thesis, grounding the correctness of the algorithms presented in Part III.

© University of Pretoria

© University of Pretoria

Chapter 2

Syntax and Semantics

This chapter discusses the following notations, conventions and languages applied in this thesis. It includes a concise introduction to Guarded Command Language (GCL) and to Linear Topic Map notation (LTM) and a range of mathematical notations that are used in this thesis.

- Basic mathematical notations to express mathematical constructs. Apart from few exceptions mentioned below, these are commonly used notations and conventions.
- An overview of topic maps (TMs) to represent knowledge. The ISO 13250 standard for TMs as defined in [124, 125] is used. This standard was designed to model semantic information and to promote interoperability of knowledge repositories.
- The Linear Topic Map notation (LTM) to describe topic maps.
- The syntax and semantics of Guarded Command Language (GCL) to describe algorithms. GCL is a simple and well-defined notation for program code.
- The style that is used throughout the thesis to formulate the reasoning when having to argue about the correctness of lemmata and algorithms as well as to formulate arguments related to the attributes of algorithms.

In some cases I depart from the norm. In such cases I justify the benefits of my decision. New notations that were invented for the concise, crisp and unambiguous description of concepts specific to this theses are introduced. These include

- Notations for the extensional definition of sets, sequences, series, general quantifications as well as partitions. These notations are related to notations used by my predecessors in the Dijkstra school.
- Definition of custom extensions to LTM that serve as shorthand notations for elaborate expressions that are often used in this thesis.
- An extension of GCL for the specification of functions.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.1 Notation

2.1.1 Naming conventions and symbols

The following general naming conventions are used:

- i, j, k, and s for integer values.
- Capital letters for predicates, sets, relations and matrices.
- Lowercase letters corresponding to the capital letter used for a set, relation or matrix for its elements/entries. For example b is assumed to be an element/entry of B.
- f, g and h as well as Greek letters such as φ, ε and ψ for functions.
- Letters may be used with numerical subscripts. For example sets U_1 and U_2 are potentially different sets and v_0 and v_1 are potentially different elements of set V.
- Letters and expressions may appear in subscripts. For example p_j and p_{j+1} may refer to consecutive items in a vector P.
- A number of symbols are introduced throughout the thesis. For the convenience of the reader these are listed in tables in Appendix A.

2.1.2 Sets

A set is a collection of objects. One way of describing, or specifying the members of a set is by extension — that is, listing each member of the set. When specifying a set by extension, the members are enclosed in the bracket pair { and }. For example the set containing first five even numbers can be written as $\{2, 4, 6, 8, 10\}$

2.1.3 Special sets

In this thesis the following special symbols are introduced to refer to a few special sets often used:

- $\mathbb{B}$ denotes the set of Booleans.
  It is the set {*false*, *true*}. *false* is often denoted by 0 and *true* by 1.
- The set of *non-negative integers* = {0, 1, 2, . . . } is known as the natural numbers. This set is denoted by $\mathbb{N}$.
- $\mathbb{N}^+$ is used to denote the set of *positive integers* = {1, 2, . . . }.
- For $n \in \mathbb{N}$, $\mathbb{N}_n$ denotes the set containing the first n natural numbers. Thus $\mathbb{N}_0 = \varnothing$, $\mathbb{N}_1 = \{0\}$ and $\mathbb{N}_n = \{0, 1, 2, \ldots, n-1\}$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.1.4 Intensional definition of sets

When the members of a set follow a pattern, the set can be specified by intensional definition — that is using a rule or semantic description. In this thesis Dijkstra's [78] notation for the intensional definition of sets is used. Dijkstra emphasised the need to unambiguously identify the dummy and to delineate the scope clearly.

The notation requires the specification of three separate aspects, namely: (i) the dummy elements, (ii) the scope description, and (iii) the description of the elements of the set in terms of the dummy elements. The | character is used as separator character between the aspects. The definition, consisting of the three aspects, is enclosed in angle brackets. This notation enables versatile expression of the conditions for membership of a set.

The following example illustrates this notation:

$\langle\, i \mid 0 < u_i \leq m \mid u_i \,\rangle$

Here i is the dummy, $0 < u_i \leq m$ specifies the scope and u_i describes the elements of the set. When identifying the dummy, its type may be included. Formulae may also be used when describing the elements. If the scope is unlimited it may be omitted. For example the set of even numbers may be specified using the following construct:

$\langle\, i \in \mathbb{N} \mid \;\mid 2 \times i \,\rangle$

Since the specification of a data type may also be considered to be scope specification restricting the dummy to be an element of a specified set, the following is a legitimate specification of the same set of even numbers:

$\langle\, i \mid i \in \mathbb{N} \mid 2 \times i \,\rangle$

2.1.5 Quantification

When proving the correctness of an algorithm, the argument often involves referring to the result of a set of predicates or values, and applying an associative and commutative binary operation over all the terms in the set. For example one may want to refer to the disjunction of a set of predicates in the expression $P_0 \wedge P_1 \wedge P_2 \wedge \cdots \wedge P_{n-1}$, or to the product of a number values in the expression $a_0 \times a_1 \times a_2 \times \cdots \times a_{n-1}$. Some texts represent these respectively as $\bigwedge_{i=0}^{n-1} P_i$ and $\prod_{i=0}^{n-1} a_i$.

In contrast, I use the notation used by Dijkstra [78] that has the advantage of making the quantified variables explicit and that separates the quantified variables from the range predicates. It extends the notation used for sets described in Section 2.1.4 by specifying the quantifier symbol along with the dummy variable.

Examples of the use of variations of this notation can be found in Backhouse and Ferreira [16], Cleophas [60], van Gasteren and Dijkstra [245], Venter *et al.* [247], Watson [253]. Table 2.1.5 shows examples to illustrate how this notation is applied to specify the above mentioned quantifications.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 2.1.5: Examples of the use of Quantification Notation

Series Expression	**Summarised using quantification**
$P_0 \wedge P_1 \wedge P_2 \wedge \cdots \wedge P_{n-1}$	$\langle \forall i \mid i \in \mathbb{N}_n \mid P_i \rangle$
$a_0 \times a_1 \times a_2 \times \cdots \times a_{n-1}$	$\langle \prod i \mid 0 \leq i < n \mid a_i \rangle$

The evaluation of the expression $\langle \oplus a \mid R(a) \mid f(a) \rangle$ requires: (i) the replacement of the introduced quantified variable a in the quantified expression $f(a)$ for all possible values of a that satisfy the range predicate $R(a)$ of a; and (ii) the application of the operation specified by the quantifier symbol $\oplus$ to the obtained series.

If the domain of a quantified expression is empty, the resulting value of this expression is the unit of the quantifier. That is

$$\langle \oplus a \mid \; \mid f(a) \rangle = 1_{\oplus}.$$

Table A4 in the appendix lists the quantified binary operators and their quantifiers as well as the unit of each quantifier used in this thesis.

2.1.6 Notation for sequences

A sequence is an ordered list of objects. Unlike a set, order matters. Furthermore elements may appear multiple times at different positions in the sequence whereas in a set elements are unique. A sequence M with n entries is called an *n-tuple*. The convention is to write it as $(m_0, m_1, m_2, \ldots, m_{n-1})$ when specifying it by extension. The values of the terms in a sequence may be random, having no relationship between i and the value of the ith term in the sequence. If this is the case, such a sequence can only be described by extension.

A notation for intensional definition of sequences is introduced in this thesis for cases where a relationship between i and the value of the ith term in the sequence can be expressed in terms of i. It is adapted from the notation for intensional definition of sets as defined in Section 2.1.4. Similar to this notation for sets, the three aspects namely the dummy variables, the range predicate and the expression describing the entries are specified separated by the | character.

To indicate that it is a series, parentheses are inserted inside the angle brackets. For example the following specifies a *5-tuple* of Booleans.

$\langle\!(i \mid i \in \mathbb{N}_5, m_i \in \mathbb{B} \mid m_i)\!\rangle$

Often the value of a term in a sequence is related to its index. For instance, if P is the *7-tuple* described by extension as $P = (3, 5, 7, 9, 11, 13, 15)$, then the formula for p_i is $2 \times i + 3$. It can thus be described using the following intensional definition:

$\langle\!(i \mid i \in \mathbb{N}_7 \mid 2 \times i + 3)\!\rangle$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.1.7 Ranges and intervals

If x may assume integer values $5, 6, 7, \ldots 599$ it is customary to express this in a formula such as $5 \leq x < 600$. Recently authors such as Watson [254] uses an interval notation where the above mentioned range of values for x can be expressed as $x \in [5, 600)$. Although it is acceptable to use this notation for arbitrary totally ordered sets, Wikipedia [263] observes that this bracket notation is rarely used for integer sets. Weisstein [256] uses it only in connection with intervals that are connected portions of the real line and Simmons [224] defines it explicitly only for intervals of real numbers. Furthermore, this notation gives rise to an ambiguity in the meaning of (a, b). It may refer to a pair or to the interval $a < x < b$. To avoid this ambiguity and to adhere to the mathematical inclination to reserve the interval notation for real intervals, it was decided to refer ranges of discrete items within an interval in terms of a mathematical expression in this thesis.

2.2 Topic Map Basics

2.2.1 Definition and representation

A Topic Map (TM) is a semantic description of information. It is a structured mark-up. Unlike metadata descriptions that are commonly embedded in documents, TM descriptions are external to the information resources they describe. A topic map is an information overlay which can be constructed separate from a set of resources, identifying instances of subjects and relationships within the set of resources. There is a clean division between the TM and the resources referenced from the TM. TMs are in essence navigational tools that can be applied to enhance the findability[1] of information in information bodies. TMs are information assets in their own right, irrespective of whether or not they are connected to external information resources [194].

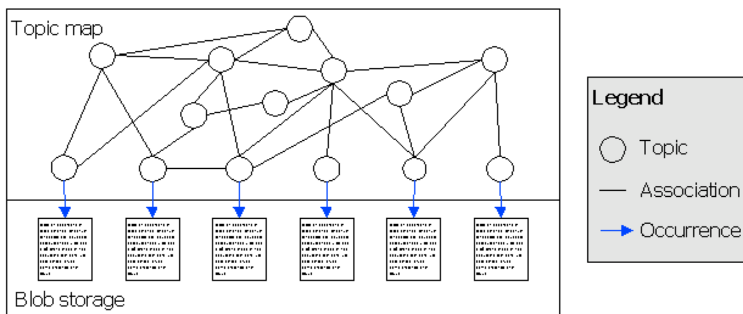


Figure 2.2.1: High level architecture of topic maps from Garshol [99]

[1] *Findability* as an Information Science term that encompasses navigability as well as the discovery of hidden information

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Figure 2.2.1 is an illustration of the TM architecture that is given by Garshol [99]. It shows a TM in the top section and icons resembling the resources that are described by this TM in the bottom section. Occurrences and topics exist on two different layers or domains. The topics and relations defined in the TM form a conceptual layer while the actual resources form a physical layer. Linking between these layers is defined in terms of topic occurrences. The high level objects constituting a TM are

- topics, indicated with circles in the diagram;
- associations, indicated with lines connecting the topics; and
- topic occurrences, indicated with arrows pointing to resources in the TM's resource store.

Notations that can be applied to describe TMs are XML Topic Maps (XTM), Linear Topic Map Notation (LTM) and Asymptotic Topic Map Notation (`AsTMa=`). XTM is the standard XML-based interchange syntax for Topic Maps [196] while LTM [98] and `AsTMa=` [21] are unofficial notations aimed to be more usable for text-based definitions. Compact Topic Map Syntax [2] (CTM) is a ISO Standard under development to replace LTM and `AsTMa=`.

I chose to use LTM to describe the TMs of algorithms defined in Chapter 9 and in Part III of this thesis. LTM was designed to specify topic maps using plain text. It is more concise and readable than XTM and less prone to ambiguities than `AsTMa=` and therefore I deem it the most suitable for the purpose of this thesis. CTM would have been better, but was not yet available when I started with my development.

I applied adaptations to LTM to suit its application in this thesis. One adaptation is to add a specification that can be applied to refer to an entity *within* the current TM as opposed to allowing only references to *external* URI's as it is specified by Garshol [98]. Another adaptation is a shorthand to refer to sections in the current text. These adaptations are discussed in Section 2.2.6 and 2.2.7. They are extensively used throughout.

The following sections define the TM concepts and introduce the reader to TMs and LTM as they are applied in this thesis. The abbreviations that are used to refer to the TM entities introduced in this section are, for convenience, also listed in Table B5 in the appendix.

2.2.2 Topics

A topic is a construct representing anything whatsoever, regardless of whether it is concrete or abstract [193]. When a topic represents a concrete thing, it is proxy for a subject with physical existence such as a book in a library, a compact disc in a music store, an artefact in a museum, an electronic document on the Internet or even a place or a person. Topics may also represent ideas and abstract concepts such as names for sets of subjects that share some attributes.

[2]http://www.isotopicmaps.org/ctm/ctm.html

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Expression 2.1 defines the syntax for a topic definition. The definition is enclosed in square brackets. TI is the topic identifier, TT is the topic type, BN; SN; DN /SC is a name triplet with identified scope consisting of a base-name (BN), sort-name (SN), display-name (DN) and a scope identifier (SC). (VN /SC) is a variant-name and its associated scope identifier. SI is a subject indicator.

```
[TI : TT = BN; SN; DN /SC (VN /SC) @SI]
```
(2.1)

The syntax for each of the entities in this expression is defined in Table 2.2.2 while their semantics are described in the next section. For the convenience of the reader the meaning of these entities are listed in Table B5 in Appendix B.

Table 2.2.2: Syntax of TM Topic Entities

Entity	**Syntax**
Topic identifier (TI)	A character string complying with XML restrictions for identifier names.
Type (TT)	The TI of a topic in the TM.
Base-name (BN)	A character string enclosed in double quotation marks.
Sort-name (SN)	A character string enclosed in double quotation marks.
Display-name (DN)	A character string enclosed in double quotation marks.
Scope identifier (SC)	The TI of another topic in the TM preceded by a forward slash.
Variant-name (VN)	A character string enclosed in double quotation marks enclosed in round brackets along with a scope identifier.
Subject indicator (SI)	A URI enclosed in double quotation marks preceded by the @ character.

2.2.3 Semantics of the topic construct

This section explains the semantics and use of each of the entities in expression 2.1. Table 2.2.3 lists these entities and gives a brief description of each. For the convenience of the reader the meaning of these entities are also listed in Table B5 in Appendix B.

A topic should at least have a topic identifier (TI). The remaining characteristics associated with a given topic are optional. If a topic is defined using only a TI, it is known as a bare topic. It is treated as a first class topic although it does not have any characteristic besides its identifier. By default a topic's TI is used as its base-name (BN), sort-name (SN) and its display-name (DN) if they are not specified.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 2.2.3: Description of TM Topic Entities

Entity	Description
Topic identifier (TI)	A unique name (identifier) used in the TM code to refer to the topic.
Type (TT)	A topic specifying the type of this topic. A topic stands in an IS-A relation with its type.
Base-name (BN)	A unique name used by people to refer to the topic.
Sort-name (SN)	A variant of the base-name that should be used when topics are sorted.
Display-name (DN)	A variant of the base-name that should be used when the topic is displayed.
Scope identifier (SC)	A topic that defines a context in which a name or variant-name should be used.
Variant-name (VN)	A variant of the base-name that is an alternative form of a base name which may be used during retrieval.
Subject indicator (SI)	An authoritative information resource that serves as unambiguous identification of the subject represented by the topic

The following is the definition of the bare topic *Person* using only a TI:

```
[Person]
```

The topic type (TT) is specified after the TI. A colon is used to separate the TI and its TT. Multiple topic types may be listed in the definition of a topic after the colon. If multiple TTs are listed, they should be separated by white space.

The topic that is defined stands in an IS-A relation with each of the TTs listed in its definition. The following are two definitions of the topic *Alan*. The first definition specifies that *Alan* is a topic of type *Person* which was previously defined. The second definition uses *Mathematician*, *Logician* and *Cryptanalyst* as TT's.

```
[Alan : Person]
[Alan : Mathematician Logician Cryptanalyst]
```

The TTs used in the second definition are the TIs of topics. It is allowable that a TI appears in a TM as a TT before it is formally defined as the TI of a Topic, or even without ever defining a topic having this TI. If this is the case, a bare topic is assumed. Bare topics may be extended using additional definitions as can be seen in later examples. In the next example the definition of *Mathematician* is extended, while a later example specifies variant names for *Cryptanalyst*.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A topic definition need not specify a topic type. If the topic type is omitted, the colon is also omitted. If a type for a topic is not specified, the default type (*Topic*) is assumed.

The name of a topic is given as a triplet consisting of the three different versions of the name separated by semicolons and preceded by the = character. The name variants for sorting and displaying the name of a topic have to be given in the specified order.

The sort-name and display-name may be omitted if they are the same as the base-name. If the display-name or both the sort-name and the display-name are omitted, redundant semicolons may be omitted. If only the sort-name is omitted, two semicolons between the base-name and the display-name are needed. The following specify names for some topics.

```
[Person ="Person"]
[Mathematician : Scientist ="Mathematician" ;; "Mathematician"]
[Alan ="Alan Mathison Turing" ; "Turing, A.M." ; "Alan"]
```

When more than one name is given, each name should be unique within its scope. The definition of each name is preceded by the = character and, with the exception of one instance specifying the name of the topic in the global scope, must have a scope identifier. Here name refers to a name definition that may contain variants for sorting and/or displaying.

The inclusion of scope identifiers enables the author of a TM to specify different names to be used in different contexts. Each context is defined by its scope. If no scope is specified the forward slash is also omitted. This will cause the name or types to be used in the unrestricted global scope. The above definitions are defined in global scope while the following definitions are limited to their specified scopes. Notice how the = sign is repeated for each of the different names that are specified.

```
[Person ="Persoon";"persoon" /Afrikaans ="Persona" /Italian]
```

In addition to different names for a topic to be used within different scopes, any number of variant-names may be specified for a topic. Variant-names represent names that are alternative forms of a base-name of a topic.

Variant-names are used in queries with the intention to increase the findability of the topic. When the TM is searched using the variant-name as keyword, the topic having that variant-name should be identified as a hit. As variant-names are only used in queries, they are not sorted or displayed, therefore, they do not have sort-names and display-names. It is required that each variant-name have a scope identifier justifying its inclusion in the TM. Each variant-name and its scope identifier must be enclosed in round brackets. If multiple variant-names are listed, they should be separated by white space. The following example defines a cryptanalyst. The definition includes two variants. This first is a layman's term while the other includes a spelling error.

```
[Cryptanalyst ="Cryptanalyst";"cryptanalyst" /English
    ("decipherer" /layman) ("cryptanalist" /incorrect)]
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The SI of a topic is an authoritative information resource that serves as unambiguous identification of the subject represented by the topic. It is used when TMs are merged to enable the identification of topics that have the same meaning. It is specified by a Uniform Resource Identifier (URI) enclosed in double quotation marks preceded by the `@` character. URI is the generic term for names and addresses that refer to resources or artefacts. A Uniform Resource Locator (URL) is one kind of URI. It is a specific character string that constitutes a reference to an electronic resource, for example to a web address or a file path of an electronic file. An example of a URI that is not a URL is the ISBN of a book that uniquely identifies the book but does not locate the book.

It is customary to use Published Subject Indicators (PSIs) to specify the SI of a topic. According to Pepper and Moore [196] a PSI is a stable resource that has been published in order to provide a positive, unambiguous indication of the identity of a subject for the purpose of facilitating TM interchange and mergeability. The following definitions specify the SI of *Person* and *Cryptanalyst* respectively using the Oxford Dictionaries[3] and WordNet [89] as PSI's:

```
[Person : Human
    @"http://oxforddictionaries.com/definition/english/person"]
[Cryptanalyst
    @"http://wordnetweb.princeton.edu/perl/webwn?s=cryptanalyst"]
```

When a topic is defined multiple times in a TM, a single topic is created with the union of the characteristics defined in the different definitions defining the topic. In the above a number of definitions for the *Person* topic was given. The following is the definition of the single *Person* topic that is defined by these definitions.

```
[Person : Human ="Person" ="Persoon";"persoon" /Afrikaans
    ="Persona" /Italian
    @"http://oxforddictionaries.com/definition/english/person"]
```

The author of a TM may decide to define topics using compound statements or by using a number of different definitions. When compound statements are used, the order of the items in the statement should be as specified in Expression 2.1.

2.2.4 Associations

An association is a relationship between two or more topics. Associations form the semantic basis of a TM. An association is a topic that defines the nature of the relationship between all of its associated topics. The definition of an association consists of an association identifier (AC) and two or more players.

Each player is a topic which plays a role in the association. Each role is a topic that describes the role of the player in the association.

[3] http://www.oxforddictionaries.com/

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Expression 2.2 shows the syntax to define an association involving n players with $n \geq 2$. The comma separated list of players participating in the association is enclosed in parentheses. AC is the association identifier. P_i is the ith player in the association and R_i is the role played by the ith player in the association.

$$\texttt{AC (P}_0 : \texttt{R}_0\texttt{, P}_1 : \texttt{R}_1\texttt{, ..., P}_{n-1} : \texttt{R}_{n-1}\texttt{) /SC} \quad (2.2)$$

Each of these should be a topic identifier (TI). Optionally an association may be scoped by appending the expression with a forward slash followed by a scope identifier (SC). The SC should also be a TI. The entities in this expression are described in Table 2.2.4. For the convenience of the reader the meaning of these TM entities are listed in Table B5 in Appendix B.

Each player in the association plays a role. For example if *Ethel* and *Julius* are defined to participate in an *isMarried* association, *Julius* may play the role of *Husband* while *Ethel* may play the role of *Wife* in this association. This is expressed in the following:

```
isMarried (Julius : Husband, Ethel : Wife)
```

Table 2.2.4: Description of TM Association Entities

Entity	**Description**
Association ID (AC)	A unique identifier used in the TM code to refer to the association.
Player (P_i)	A topic that participates in the association.
Role (R_i)	A topic that defines a role of a player in the association.
Scope (SC)	A topic that specifies a context in which this association holds.

Self association is specified with the topic involved playing two different roles. The following code shows the self relation that *Alan* killed himself:

```
murder(Alan : Murderer, Alan : Victim)
```

Associations, players and roles are topics. Ideally they should be defined elsewhere in the TM, using topic definitions as specified in Expression 2.1. If a TI, however, appears in a relation definition but is not involved in any other topic definitions, a bare topic is assumed.

Every player in an association must have a role. The syntax, however, allows for the omission of the role of a player (as well as the colon separating the player from its role). When the role of a player is not explicitly specified, the topic type of the player is assumed as the role in the association. This applies only if the player is a topic that has only one type.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

It is permissible to define the topic in-line in a relation definition. In this case a definition complying with the syntax specified in Expression 2.1 will replace a P_i or R_i instance in Expression 2.2. The following shows a family relationship involving *Alan* and others with some in-line topic definitions:

```
family(Alan : Son, John : Son ="John Turing",
   Julius : Father ="Julius Mathison Turing",
   Ethel : Mother ="Ethel Stoney";;"Ethel Sara Stoney" /Unmarried
         ="Ethel Turing";;"Ethel Sara Turing" /Married )
```

2.2.5 Occurrences

An occurrence of a topic is a link to an instance of the actual *thing* the topic represents. Thus an occurrence of a topic represents the information that is specified as relevant to a given topic.

Usually an occurrence is a pointer to a document about the topic or otherwise related to the topic. Expression 2.3 defines the syntax for specifying an occurrence.

$$\texttt{\{TI, TP, TL\} /SC} \qquad (2.3)$$

Occurrence information is given in braces. The first three pieces of information, all of which are required, appear inside the braces, separated by commas. TI is the identifier of the topic which has the occurrence. TP is the identifier of the occurrence type. TL is a locater of the occurrence and SC the scope of the occurrence. The scope is optional. Each of these entities, except the TL should be a topic identifier (TI). The TL is a reference to an object or concept in the real world[4]. Optionally an association may be scoped by appending the expression with a forward slash followed by a scope identifier (SC). The SC should also be a TI. The entities in this expression are described in Table 2.2.5.

Table 2.2.5: TM Occurrence Entities

Entity	Description
Topic identifier (TI)	The identifier of the topic which has the occurrence.
Occurrence Type (TP)	The identifier of a topic that defines the type of the occurrence.
Occurrence locator (TL)	A URI enclosed in double quotation marks or text enclosed in double square brackets
Scope (SC)	A topic that specifies a context in which this occurrence holds.

[4]In this thesis a reference to another topic is also allowed (Section 2.2.6)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The requirement to specify a type for each occurrence suggests that different types of occurrences are assumed. This allows the author of the TM to define categories of occurrences and to convey more information about the type of an occurrence and the relevance of occurrences [193]. TPs, like TTs and roles, are also topics. Although an author may treat a TP like any other TI in the TM, The topics associated with TPs are usually only defined in occurrence definitions and exist only as bare topics in the TM. Some TM authoring tools distinguish between topics and topic types. An occurrence resource can be specified either as an in-line definition or in the form of an Uniform Resource Identifier (URI). If an occurrence is specified in terms of an in-line definition, the occurrence is metadata text that resides in the TM. In this case the text is enclosed in double square brackets. A URI is enclosed in double quotation marks.

The following shows some occurrences for the topic *Alan* that is defined in Section 2.2.3:

```
{Alan, Biography,
    "http://www.biography.com/people/alan-turing-9512017"}
{Alan, Biography,
    "http://www.turing.org.uk/sources/biblio.html"}
{Alan, Quote,
    "http://www.brainyquote.com/quotes/authors/a/alan_turing.html"}
{Alan, AlmaMater, [[ King's College, Cambridge ]]}
{Alan, AlmaMater, [[ Princeton University ]]}
```

2.2.6 Topics as occurrence locators

Appendix C of this thesis is the definition of a TM containing algorithms. It is specified using LTM. When this TM was created, the need arose to treat topics defined in the TM as occurrences of other topics in the TM. To accommodate the idea to see one topic as a specific type of occurrence of another topic while adhering the TM standard, one can define relation with roles *baseTopic* and *occurrenceTopic* for each of the occurrence types that may be another topic instead of a URI or in-line definition.

As defined in the listing above `Alan` has an occurrence type named `AlmaMater` of which two occurrences are defined in-line. If one wants to include more information about say the Princeton University in the TM it would make sense to define a topic `Princeton`. In this case the in-line occurrence namely [[Princeton University]] is not associated with the topic `Princeton`. To rectify this, the in-line definition can be replaced by the definition and use of a relation specifying the same information, using the defined topic. The following is the code that specifies the Alma Mater relationship between the topic `Alan` and the topic `Princeton`.

```
[isAlmaMater] /* roles: Student, Institution */
isAlmaMater(Alan : Student, Princeton : Institution)
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

To eliminate the need to define an occurrence relation and role types for each occurrence type that may be another topic, I introduce a syntax that can serve as as shorthand to specify such relation as well as the implied roles similar to how other occurrences are specified. I relax the restriction that an occurrence locater (TL) may only be a URI's or an in-line definition. Instead of specifying a URI or an in-line definition, I allow that the TI of a topic be used as an occurrence locater. The following example shows how the information in the above example is rewritten without the explicit definition of the occurrence relation:

```
{Alan, AlmaMater, Princeton}
```

2.2.7 Custom occurrence locators

While creating the TM, I also realised that I often need to let sections in the thesis itself serve as URIs for specific types of occurrences of the topics in this TM. A special notation for TL specification that is not included in LTM is specified here to accommodate this need. The notation is specified with specific reference to entities within this thesis, but can be used generically whenever there is a need to refer the content of similar texts.

Custom TLs apply the same syntax as in-line TLs in LTM. Two variations are defined, one to refer to an external resource that also appears in the bibliography of this thesis and the other to refer to an entity in this thesis. The following describe their syntax:

```
[[ THIS-Citation:  <author> [#] ]]                    (2.4)
[[ THIS-Reference:  <reference type> #: Page # ]]      (2.5)
```

Expression 2.4 applies to citations. `<author>` represents the author of the citation and # is the citation number in the bibliography of this thesis. Expression 2.5 applies to references. `<reference type>` represents the specific reference type in the thesis. It should be one on the following: *Algorithm*, *Figure*, *Lemma*, *Section*, or *Table*.

The following illustrates the application of this syntax. It shows some occurrences of a topic *Prosser*, that is defined in the TM of TC algorithms discussed in Part III.

```
{Prosser, Publication, [[ THIS−Citation: Prosser [203] ]]}
{Prosser, Discussion, [[ THIS−Reference: Section 12.4: Page 175 ]]}
```

The use of the syntax of Expression 2.5 is extended to refer to information contained in other publications. The publication containing the information needs to be defined as topic. The defined topic has to be a unique and exact identification of the publication. The following defines *Watson10* as a topic representing the electronic version of Watson [254]:

```
[Watson10
   @"http://repository.up.ac.za/handle/2263/23648"]
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Once such a topic is created, the TI of the defined topic may be used instead of the keyword THIS in Expression 2.5. The following is an example of how this is done to refer to a section in Watson [254]. It shows the definition of the topic *Minimal* and the occurrence of a visualisation of this algorithm in this publication:

```
[Minimal : MADFA ="Minimal intermediate ADFA"]
{Minimal, Visualisation,
    [[ Watson10−Reference: Section 6.3: Page 54 ]] }
```

2.2.8 Reification

The Oxford English Dictionary [182] defines *reification* as making of something abstract into something more concrete or real. In computer science reification is the process by which an abstract idea is turned into an explicit processable data model. In conceptual modelling reification of a relationship means viewing a relation as an entity.

In TMs the same concept is applied. Anything in the TM that is not a topic can be turned into a topic in the TM by reifying it. As one can only make assertions about topics in a TM, reification makes it possible to declare any non-topic construct as a topic essentially removing this limitation. Thus relations, occurrences as well as other sub-entities may be made into topics enabling these constructs to be named and treated as a topics.

Pepper [195] explains that a name, an occurrence, an association, an association role (or even the topic map itself) can be reified in order to make assertions about the reified construct as if it is a topic.

Any construct specified using LTM notation is reified by appending the construct with a TI preceded by a $\sim$ character. After doing so, this TI may be applied like any other TI.

$$\mathtt{AC\ (P_0\ :\ \ R_0,\ P_1\ :\ \ R_1,\ \ldots,\ P_n\ :\ \ R_n)\ /SC\ \sim TI} \tag{2.6}$$

$$\mathtt{\{TI,\ TP,\ TL\}\ /SC\ \sim TI} \tag{2.7}$$

$$\mathtt{[TI\ :\ TT\ =\ BN;\ SN;\ DN\ /SC\ \sim TI\ (VN\ /SC)\ @SI]} \tag{2.8}$$

Expression 2.6 shows how Expression 2.2 in Section 2.2.4 on page 27 may be modified to reify the association. Expression 2.7 shows how Expression 2.3 in Section 2.2.5 on page 28 may be modified to reify the occurrence. Expression 2.8 shows how Expression 2.1 in Section 2.2.2 on page 23 can be modified to reify a name of a topic within the topic definition. In each case the construct preceding the $\sim$-character is the definition of a new topic identified by the specified TI that follows after the $\sim$-character. The following shows a reification of the association between *Ethel* and *Julius*. As a result of this reification a new topic named *TuringCouple* is added to the TM. The next statement uses the newly created TI to make an assertion about the income range of this couple:

```
isMarried (Julius : Husband, Ethel : Wife) ~TuringCouple
{TuringCouple, IncomeRange, [[ upper−middle−class ]]}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.3 Guarded Command Language

A necessary modus operandi when having to compare algorithms and having to show their correctness, is to use a standardised notation that facilitates correctness reasoning to present the algorithms under consideration. Guarded Command Language (GCL) has been used by most authors creating taxonomies for TABASCO[5]. Dijkstra [77] defined this language. GCL is a universal language that is well suited to be used as uniform notation for expressing algorithms. In this thesis algorithms are presented using a slightly augmented version of this GCL. This language was chosen here to describe algorithms because it is mathematically founded and designed to support demonstration of program correctness. The translation of algorithmic descriptions that is formulated in terms of this language to high level programming languages such as C++, Java and Pascal is fairly trivial.

The following sections briefly describe the semantic characterisation of this language. Augmentations that are added to the language is the use of variables, constants and functions.

2.3.1 Variables

Data structures are fundamental constructs in practical algorithms. To enable algorithms to operate on data, the data are represented in data structures that allows the algorithm to manipulate the values of the represented data. The data structures that are used by an algorithm during execution of the algorithm are called *variables*.

In the augmented GCL used in this thesis, variables are specified at the beginning of an algorithm directly after the constants and are preceded by the keyword **var**. The syntax used in this thesis to specify variables is the same as for constants.

Each variable is specified in terms of an expression to delineate the scope of the variable. A variable may assume an initial value which may be indicated by using the = sign. This syntax is not the same as the syntax given by Kourie and Watson [147]. They do not apply the optional initialisation of variables and furthermore would write **var** $x : \mathbb{N}$ to declare that x is a natural number. In this thesis as the expression **var** $x \in \mathbb{N}$ is used instead. This notation allows the specification of a variable in terms of relations other than $\in$, such as $\subset$ or $\Subset$.

Algorithm 2.3.1: Specifying variables

---

var $M_0 \in \mathbb{B}[n,\ n]$
$M_1 \in \mathbb{B}[n,\ n]$

Algorithm 2.3.1 is an example showing how to specify variables. The algorithm declares two variables respectively named M_1 and M_2 which are both specified to be elements of $\mathbb{B}[n,\ n]$. The notation used to refer to these constructs and meaning of expressions such as $U \times U$, $|U|$ and $\mathbb{B}[n,\ n]$ are explained in Chapter 3.

[5]Taxonomy based software construction (Section 1.2)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

2.3.2 Constants

The use of constants is not common among authors who use GCL as it is contrary to how one reasons in the functional programming paradigm. Constants are more often used in the imperative programming paradigm where it is primarily advocated to simplify code maintenance.

In this thesis, however, the specification of constants is introduced as a method to specify preconditions rather than its usual imperative use. Here the data structures that are assumed to have a value when an algorithm starts, and are not changed during the execution of the algorithm, are called *constants* of the algorithm.

In the augmented GCL used in this thesis, constants are specified at the beginning of an algorithm definition preceded by the keyword **const**. Each constant is specified in terms of an expression to delineate the scope of the constant. If the specific value of a constant is relevant, it is indicated by appending the specification of the constant with the = sign followed by an expression that evaluates to this specific value.

Algorithm 2.3.2: Specifying constants

---

const $R \subseteq U \times U$
$n \in \mathbb{N}^{+} = |U|$

Algorithm 2.3.2 is an example that specifies two constants. The mathematical notation used to refer to these constructs and meaning of expressions such as $U \times U$ and $|U|$ are explained in Chapter 3. The scope of the constant R is delineated in the expression $R \subseteq U \times U$. It is assumed that it already contains a value. The specific value is not relevant in this case. The scope of the constant n is delineated in the expression $n \in \mathbb{N}^{+}$. In this case its specific value is relevant. $n = |U|$ is a precondition for the algorithm.

2.3.3 Skip and abort

The **skip** command means *do nothing*. It executes in finite time and does not change the state of the machine that executes this command.

The **abort** command means *do anything*. It has an undetermined effect on the machine that executes this command.

2.3.4 Assignment (:=)

If a variable x is to be replaced by the value of an expression E, it is done by using the assignment command:

$$x := E.$$

GCL also allows what is known as *concurrent assignment*. Here a number of different variables can be substituted simultaneously. Each variable is assigned the value of a different expression. This command is denoted by a list of comma

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

delimited variables at the left hand side of the assignment operator and an equally long list of comma delimited expressions at the right hand side. Thus one is allowed to write:

$$x_0, x_1, \ldots, x_n := E_0, E_1, \ldots E_n$$

The meaning of such a command is the replacement of each variable at the left hand side of the assignment operator with the value of the corresponding expression in the right hand side of the command. It is important that the replacement takes place concurrently. Thus the following expression swaps the variable values:

$$a, b := b, a$$

2.3.5 Composition of commands

Let S_0 and S_1 be two commands. Then the composition of these commands forms the construct $S_0; S_1$. It is permissible to compose any number of commands. A command that is the composition of two or more commands is called a compound command. Single commands and compound commands are treated uniformly. In this thesis the term *command* is used to refer to a single or a compound command.

2.3.6 Guarded commands

A guarded command is a command (possibly compound) preceded by a Boolean expression and an arrow. The Boolean expression is called a *guard*. The guarded command is executed only on the condition that the guard initially evaluates to *true*. For example

$$B \rightarrow S$$

Here B is a Boolean expression and S represents a command that may be executed if the expression B initially evaluates to *true*. A guarded command set is one or more guarded commands separated by the $[\!]$ character. For example

$$B_0 \rightarrow S_0 \;[\!]\; B_1 \rightarrow S_1 \;[\!]\; \ldots \;[\!]\; B_n \rightarrow S_n$$

Here each B_i is a Boolean expression and S_i represents its corresponding command. Each S_i may be a compound command. A single guarded command and a guarded command set are treated uniformly. In this thesis the term *guarded command set* is used to refer to a single guarded command or a guarded command set consisting of arbitrary number of guarded commands. The semantics of a guarded command set is dependent on weather it is part of an alternative construct or a repetition construct. These constructs are discussed next.

2.3.7 Alternative constructs

An alternative construct is formed by enclosing a guarded command set by the bracket pair **if ... fi**. For example

$$\textbf{if } B_0 \rightarrow S_0 \;[\!]\; B_1 \rightarrow S_1 \;[\!]\; \ldots \;[\!]\; B_n \rightarrow S_n \textbf{ fi}$$

On execution, a command corresponding with a guard that evaluates to *true* is executed. This construct can give rise to nondeterminacy. When more than one guard is true when the construct is activated, it is undetermined which of the corresponding commands will be selected for activation. It is assumed that

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

all guards are defined. If the initial state is such that none of the guards are true, activation of the construct will lead to abortion of the algorithm that contains such alternative construct. This means that an alternative construct for which none of its guards are true is equivalent to the **abort** statement. Therefore, it is important to select the guards for an alternative construct in such a way that it can be shown that at least one will be true when the construct is reached in the algorithm.

Some authors such as Cleophas [60] and Kourie and Watson [147] use

as $b \rightarrow S$ **sa**

for a shortcut for

if $b \rightarrow S \;[\!]\; \neg b \rightarrow$ **skip fi**

2.3.8 Repetition constructs

A repetition construct is formed by enclosing a guarded command set by the bracket pair **do ... od** similar to the formation of an alternative construct. For example

do $B_0 \rightarrow S_0 \;[\!]\; B_1 \rightarrow S_1 \;[\!]\; \ldots \;[\!]\; B_n \rightarrow S_n$ **od**

On execution, the guards of the construct are evaluated to determine which command should be executed. The command corresponding to *any* guard that evaluates to *true* is executed giving rise to possible nondeterminacy. On completion, the construct is re-executed.

The process is repeated until the state is such that none of the guards are true. At this stage the construct will terminate. When repetition constructs are used in a program, it is important to be able to show that all the guards of the repetition construct will evaluate to *false* after a finite number of guarded commands were selected for execution and their corresponding commands were executed accordingly. This is necessary to ensure that the construct will eventually terminate and the machine executing it can reach a final state.

Finally,

for $x : P \rightarrow S$ **rof**

is used for executing command list S once for each value x initially satisfying predicate P. van den Eijnde [244] points out that this is only admissible if there is a finite number of such values for x. Often P is an expression like $x \in X$. Although P may be expressed in a way that suggest a specific order in which the x values can be chosen, this order cannot be assumed. This is contrary to how Kourie and Watson [147] see it. They apply nondeterminism on an expression like **for** $x : x \in X \rightarrow S$ **rof** only if the set X is not ordered.

2.3.9 Functions

A number of sub-program types called procedures and functions are defined by Kourie and Watson [147]. They specify a function to be a special case of a procedure that bears additional parameters. In this thesis, however, only functions are specified. Here a function is a sub-program that returns a *n-tuple* and the construct they call a procedure, is defined as a function that returns $\varnothing$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The commands that constitute the actions of the function is called the *body* of the function. In most cases this command is a compound command. To define a function using the augmented version of GCL used in this thesis, the body of the function is enclosed by the bracket pair **func ... cnuf**. The keyword **func** is followed by a *name* for the function, a formal parameter list and the specification of the nature and data structure (types) of the items in its return *n-tuple*.

The formal parameter list is written in parenthesis. It may contain any number of items. If the function has more than one parameter, the parameters are separated by commas. Unlike Kourie and Watson [147], who require only an identifier name for each parameter, each parameter is defined here by an expression such as $M_0 \in \mathbb{B}[n,n]$ or $R \subseteq U \times U$. It delineates the scope of each parameter and at the same time specifies an identifier that can be used to refer to the parameter in the body of the function. The formal parameters are assumed to be passed by value. This eliminates the need to have keywords like those used by Kourie and Watson [147] to the distinguish different behaviours.

Algorithm 2.3.9: Declaring and calling a function

---

func largest$(n \in \mathbb{N}, m \in \mathbb{N}) : \langle \mathbb{N} \rangle$
    if $(n > m) \rightarrow$ **return** $\langle n \rangle$
    $[\!]$ $(n \leq m) \rightarrow$ **return** $\langle m \rangle$
    fi
cnuf

---

const $x, y \in \mathbb{N}$
var $z \in \mathbb{N}$

$z :=$ largest(x, y)

The return type is specified as a *n-tuple* of sets, with each set delineating the scope of an element of the *n-tuple* returned by the function. This specification follows the formal parameter list in the definition of the function preceded by a colon. If the return type has more than one item, the items are separated by commas and enclosed in the angle bracket pair $\langle \ldots \rangle$. If it has one item, the angle brackets may be omitted.

When defining a function, the formal parameter list, as well as the return *n-tuple* may contain any number of items. They may even be empty. The number of formal parameters need not correspond with the number of elements in the return *n-tuple*.

A **return** command is introduced. The body of the function must have at least one **return** command. A return command consists of the keyword **return** followed by a *n-tuple* of return values. If the return type has more than one item, the values are separated by commas and enclosed in the angle bracket pair $\langle \ldots \rangle$. If it has one item, the angle brackets may be omitted. The values in this *n-tuple*

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

may be variables, constants or expressions that are defined within the scope of the function. When a **return** statement is executed these values are assigned to the corresponding elements in the return *n-tuple*. The *n-tuple* of return values in every return statement in the function must adhere to their scopes defined in the return type of the function.

All variables and constants declared within a function must be distinct.

A function may be used in any command where the return type of the function is required. When a function is used, its name should be followed by its actual parameters. The actual parameters are written in parenthesis. It is a comma separated list of variables, constants or expressions. The function is executed when a command containing the function is evaluated. An example may be an assignment command with a *n-tuple* the size of the return *n-tuple* of the function on the left hand side and the function on the right hand side. In this case the elements of the receiving *n-tuple* are listed enclosed in the bracket pair $\langle \ldots \rangle$ and separated with commas. If the receiving *n-tuple* has only on element, the angle brackets may be omitted. Algorithm 2.3.9 is an example of the definition and use of a function. This function, called `largest`, determines and returns a natural number, which is the largest of two natural numbers that are passed as parameters. The only command in this algorithm is an assignment command that assigns the return value of the function to the variable x. See Algorithm 5.4.2 (AcyclicPath) for an example of a function with a pair as return type.

2.4 Lemmata

2.4.1 Definition

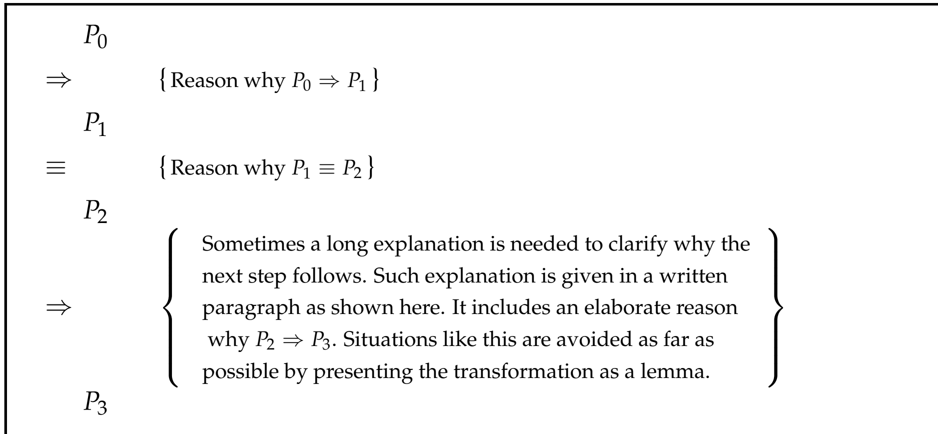
There is no technical distinction between a lemma, a proposition, and a theorem. A lemma is a proven statement, typically named a lemma to distinguish it as a truth used as a stepping stone to a larger result rather than an important statement in and of itself.

2.4.2 Notation

This thesis uses lemmata to illustrate the correctness of algorithms included in the topic map of transitive closure algorithms that is constructed in Part III. A lemma is shown to be correct by listing a series of known correct transformations of a correct statement concluding in the statement of the lemma. Each transformation is supported with a hint stating why the transformation is deemed correct. Lemmata may be used to justify steps in other lemmata. A style that resembles the style proposed by Dijkstra and Scholten [79] is used to present lemmata. Lemma 2.4.2 illustrates this style.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA



Lemma 2.4.2: Illustration of the style used in proofs

2.5 Summary

This chapter states the conventions and tools that are used in this thesis. It does not include an explication of sufficiently basic symbols, terms and concepts.

The use of the Dijkstra-like notation for sets and quantifications is well known within a small community, but the broader computer science and mathematics community may be less familiar with it. Furthermore, the Dijkstra-like notation is not used consistently by various authors. The notation is explained at a sufficiently basic level to serve the broader readership of this thesis and to clarify the variant of this notation that I use in this thesis. A new notation to specify series that is consistent with the notation for sets and quantifications is introduced (Section 2.1.6). This is needed for the concise and accurate specification of partitions (Section 4.5).

Topic maps offer a standard for knowledge representation that is not widely used. Most mathematicians and computer scientist would not have heard of them and a limited community of information scientists is familiar with their use. Linear topic map notation (LTM) is one of various notations that can be used to specify topic maps. The syntax and semantics of LTM are explained for the benefit of readers who may not be familiar with topic maps and the LTM notation. This information is also needed to explain the adaptations to LTM that I have introduced. One adaptation is to add a specification that can be applied to refer to an entity *within* the current TM as opposed to allowing only references to *external* URI's as specified by Garshol [98]. I call these *topics as occurrence locators*. Another adaptation is a shorthand to refer to sections in specified texts as well as in the current text. I call these *custom occurrence locators*. These are discussed in Section 2.2.6 and 2.2.7 respectively.

The syntax and semantics of the Guarded Command Language (GCL) used in this thesis is explained for the benefit of the reader that may not be familiar with this language and to clarify deviations and extensions introduced in this

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

thesis. In Section 2.3.1 I deviate from how other authors have dealt with variables. The section includes a clarification of the reason for this augmentation of the language that I have introduced here. Kourie and Watson [147] introduced the use of procedures and functions in GCL as the language defined by Dijkstra [77] does not include these. The definition of functions in GCL I introduce here is, however, not the same as their definitions. In Section 2.3.9 I propose only one versatile definition instead of their set of definitions. I also introduce the use of constants in Section 2.3.2. Constants in the context of this thesis are used to specify the precondition of and algorithm as well as the data structures used by the algorithms.

For completeness the style used when writing out proofs, as proposed by Dijkstra and Scholten [79], is illustrated.

Having established the syntax and semantics of the languages and tools used in the thesis, the scene is set to discuss the mathematical knowledge needed to describe the transitive closure problem discussed in the remainder of this part, to consider the TM that is specified in Part II and to follow the arguments in Part III.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

Chapter 3

Domain Knowledge

This chapter serves as an introduction to the problem domain by briefly presenting the necessary mathematical preliminaries. The reason for including this chapter in the thesis is to introduce the terminology and to state how the concepts are interpreted in this thesis. The chapter provides a single point of reference for basic concepts related to symbolic logic, sets, relations and functions used in this thesis. The reader who is familiar with these concepts may skip this chapter.

Most of the operations performed by algorithms that solve the transitive closure (TC) problem are logical operations on Boolean variables. Arguments in lemmata also often rely on logical inferences. For this reason the fundamentals of symbolic logic is briefly discussed in Section 3.1.

Set theory is a basis of modern mathematics, and notions of set theory are used in all formal descriptions. Section 3.2 is a very brief introduction to basic set theory needed for further description. Section 3.3 introduces relations. Relations and operations on relations are crucial for succinctly discussing the TC problem. Section 3.4 discusses the properties of relations needed in proofs later in this thesis. Section 3.5 illustrates different models that can be used to represent relations and the correspondence between these models. Section 3.6 introduces mathematical functions as specialised relations. The section lists known theories of functions used in arguments in this thesis.

3.1 Symbolic Logic

Symbolic logic is a mathematical model of deductive thought. In the area of proving program correctness, predicate calculus allows one to precisely state under which conditions a program gives the correct output. This section defines important terms in logic and the rules of logical equivalence that are often used in correctness arguments in Part III in this thesis.

Negation

Negation indicates the opposite, usually employing the word *not*. The negation of predicate P is expressed as $\neg P$. If P is true, then $\neg P$ is false and vice versa.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Conjunction
A conjunction is a compound predicate formed by using the word *and* to join two predicates. The conjunction of predicated P and Q is expressed as $P \wedge Q$. $P \wedge Q$ is true only if both P and Q holds.

Disjunction
A disjunction is a compound predicate formed by using the word *or* to join two predicates. The disjunction of predicated P and Q is expressed as $P \vee Q$. $P \vee Q$ is true if either P or Q or both holds.

Double negation $\neg\neg P \iff P$

DeMorgan rules
$\neg(P \wedge Q) \iff \neg P \vee \neg Q$
$\neg(P \vee Q) \iff \neg P \wedge \neg Q$

Idempotence
$P \wedge P \iff P$
$P \vee P \iff P$

Commutative rules
$P \wedge Q \iff Q \wedge P$
$P \vee Q \iff Q \vee P$

Associative rules
$(P \wedge Q) \wedge R \iff P \wedge (Q \wedge R)$
$(P \vee Q) \vee R \iff P \vee (Q \vee R)$

Distribution rules
$P \wedge (Q \vee R) \iff (P \wedge Q) \vee (P \wedge R)$
$P \vee (Q \wedge R) \iff (P \vee Q) \wedge (P \vee R)$

3.2 Sets

A set is defined as a collection of objects. Such a collection can consist of few (even 0) or many (even infinitely many) items. When a set has more than one element, the order of the items in a set has no significance.The smallest set consists of 0 items, is denoted by $\varnothing$ and is called the *empty set*. A collection with only one item is called a *singleton set*. The specification of set by extension is introduced in Section 2.1.2 while the way to specify sets by extension in this thesis is explained in Section 2.1.4.

The item x is called a member of set X if item x occurs in collection X. This is denoted by $x \in X$. Mathematics uses the term *element* instead of *item*. The expression $x \in X$ is regarded as a predicate (i.e. it has a value of *true* or *false*).

3.2.1 Size of a set

For a set U a size can be defined as the number of elements in the set. The size of a set U is denoted by $|U|$.

© University of Pretoria

3.2.2 Operations on sets

Three well-known binary operations on sets are union ($\cup$), intersection ($\cap$) and difference ($-$). The definitions are, for set X and Y

$$z \in X \cup Y \equiv z \in X \vee z \in Y$$
$$z \in X \cap Y \equiv z \in X \wedge z \in Y$$
$$z \in X - Y \equiv z \in X \wedge z \notin Y$$

3.2.3 Subset

For two sets X and Y, X is a subset of Y if every member of X is a member of Y too. In formulae $\langle \forall i : x_i \in X : x_i \in Y \rangle$, and in notation $X \subseteq Y$. An equivalent formulation of $X \subseteq Y$ is $X \cup Y = Y$.

3.2.4 Power set

The power set, $\mathcal{P}(X)$, of a given set X is the set of all subsets of X. In formulae

$$\langle\, Y \mid Y \subseteq X \mid Y \,\rangle.$$

3.2.5 Cartesian product

Given the sets $X_0,\ X_1,\ \ldots,\ X_{n-1}$. The Cartesian product $X_0 \times X_1 \times \ldots \times X_{n-1}$ in formulae $\langle \prod i : i \in \mathbb{N}_n : X_i \rangle$, of these sets is defined as

$$\langle\, x_0, x_1, \ldots, x_{n-1} \mid \langle \forall i : i \in \mathbb{N}_n : x_i \in X_i \rangle \mid (x_0, x_1, \ldots, x_{n-1}) \,\rangle.$$

For example:

$$X \times Y = \langle\, x, y \mid (x \in X) \wedge (y \in Y) \mid (x, y) \,\rangle.$$

3.3 Relations

Subsets of a Cartesian product are called relations. Given sets $X_0, X_1, \ldots, X_{n-1}$, any subset of the Cartesian product of these sets is an *n-ary* relation. n is called the *arity* of the relation. When $n = 2$, the term *binary relation* is used. If R is a binary relation, $(x, y) \in R$ is often denoted by $x\, R\, y$.

For a binary relation $R \subseteq X \times Y$, X is called the *domain* of R, and Y is called its *codomain*. A relation is said to be *homogeneous* if its domain and its codomain is the same set. Thus, the binary relation $R \subseteq X \times X$ is *homogeneous*. This relation R is also referred to as a relation *on* (universe) X. The rest of this chapter concentrates on relations on finite universes.

The *inverse* R^{-1} of R is defined by

$$R^{-1} = \langle\, x, y \mid (x, y) \in R \mid (y, x) \,\rangle$$

The *left domain* of R, denoted by $R<$, is the subset of $U \times U$ defined by

$R< = \langle\, x, y \mid (x, y) \in R \mid (x, x) \,\rangle$ Similarly the *right domain* of R is defined by

$$R> = \langle\, x, y \mid (x, y) \in R \mid (y, y) \,\rangle$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.3.1 Some special relations on finite universe U

There are three elementary relations on U:

- the *e*mpty relation $\varnothing$,
- the *c*omplete relation $U \times U$ and
- the *i*dentity relation E_U defined by $E_U = \langle\, u \mid u \in U \mid (u, u) \,\rangle$

3.3.2 Monotype and point

It is often useful to refer to specific subsets of the identity relation. For this reason we define the concepts of monotype and point.

- A subset of an identity relation on a set is called a *monotype*. If $S \subseteq E_U$ it is said that S is a monotype on U. Thus if $R \subseteq U \times U$, $R<$ and $R>$ are monotypes on U.

- If a monotype has only one element it is called a *point*. Thus, when one says *point* S on U it means that $S \subseteq E_U \wedge |S| = 1$.

For $A \subseteq U$, define E_A as follows:

$$E_A = \langle a : a \in A : (a, a) \rangle \tag{3.1}$$

Clearly $E_A \subseteq E_U$, therefore E_A is a monotype on U. In particular:

$$E_\varnothing = \varnothing \tag{3.2}$$

3.3.3 Union and intersection of relations

Because relations are sets they can be joined and intersected. Homogeneous relations are closed under union and intersection.

A property of monotypes that is of particular interest is:

$$E_{A \cup B} = E_A \cup E_B \tag{3.3}$$

3.3.4 Composition of relations

An important operation on relations is the *composition* of relations. This operation is key to the calculation of the transitive closure of relations. It is denoted by the $\circ$ symbol. It is defined as follows:

$$R \circ S = \langle\, a, b, c \mid (a, b) \in S \wedge (b, c) \in R \mid (a, c) \,\rangle$$

In other words, $R \circ S \subseteq U \times U$ is defined by the rule that says $(a, c) \in R \circ S$ if and only if there is an element $b \in U$ such that $(a, b) \in S$ and $(b, c) \in R$. Thus $R \circ S$ means that S is applied first and then R is applied. In particular fields, authors might denote by $S \circ R$ what is defined here to be $R \circ S$ [1].

[1] https://en.wikipedia.org/w/index.php?title=Composition_of_relations&oldid=697415897

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The composition operator has some interesting properties:

- It has a zero $\varnothing$, i.e. its holds that $\varnothing \circ R = R \circ \varnothing = \varnothing$ for any $R \subseteq U \times U$.
- It has a unit E_U, i.e. it holds that $E_U \circ R = R \circ E_U = R$ for any $R \subseteq U \times U$.
- It is associative, i.e. it holds that $(R \circ S) \circ T = R \circ (S \circ T)$.
  The advantage of associativity is that brackets may be omitted without changing the meaning of an expression.
- Composition distributes (from left and from right) over union. Left distribution in formulae is $(S \cup T) \circ R = (S \circ R) \cup (T \circ R)$ and right distribution is $R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$
- Composition is monotonic in both arguments; e.g. if $S \subseteq T$ then $R \circ S \subseteq R \circ T$.

The result of certain symmetric compositions of relations with special properties is predictable, for example if *point* A on U and $R \in U \times U$ then $(A \circ R \circ A)$ is either empty or equal to A. This shown by the following three lemmata. Lemma 3.3.4 is an interim step used by the lemmata in the following two sections. These observations are used in the proof of Lemma 6.4.3 which in turn is a crucial ingredient of a fundamental algorithm to calculate the transitive closure of a relation.

$$
\begin{array}{ll}
 & A \circ R \circ A \\
= & \quad \{\text{Definition of } \circ\} \\
 & \langle\, a, b, c \mid (a,b) \in A \wedge (b,c) \in R \mid (a,c)\,\rangle \circ \mathrm{A} \\
= & \quad \{\textit{Point } A, \text{ thus } a = b\} \\
 & \langle\, b, c \mid (b,b) \in A \wedge (b,c) \in R \mid (b,c)\,\rangle \circ \mathrm{A} \\
= & \quad \{\text{Definition of } \circ\} \\
 & \langle\ a, e, f \mid \\
 & \quad (a,e) \in \langle b, c : (b,b) \in A \wedge (b,c) \in R : (b,c)\rangle \wedge (e,f) \in A \mid (a,f)\,\rangle \\
= & \quad \{\textit{Point } A, \text{ thus } e = f\} \\
 & \langle\, a, e \mid (a,e) \in \langle b, c : (b,b) \in A \wedge (b,c) \in R : (b,c)\rangle \wedge (e,e) \in A \mid (a,e)\,\rangle \\
= & \quad \left\{ \begin{array}{l} \text{Since } (a,e) \in \langle b, c : (b,b) \in A \wedge (b,c) \in R : (b,c)\rangle \\ (a,e) \text{ complies with the format of expressing the} \\ \text{elements in the set with } a = b \text{ and } e = c \end{array} \right\} \\
 & \langle\, a, e \mid (a,a) \in A \wedge (a,e) \in R \wedge (e,e) \in A \mid (a,e)\,\rangle \\
= & \quad \{\textit{Point } A, \text{ thus } a = e\} \\
 & \langle\, a \mid (a,a) \in A \wedge (a,a) \in R \mid (a,a)\,\rangle
\end{array}
$$

Lemma 3.3.4: *Point* $A \Rightarrow A \circ R \circ A = \langle\, a \mid (a,a) \in A \wedge (a,a) \in R \mid (a,a)\,\rangle$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.3.5 A symmetric composition that is empty

Lemma 3.3.5 shows that $(A \circ R \circ A) = \varnothing$ if $A \nsubseteq R$

$$
\begin{array}{ll}
 & A \circ R \circ A \\
= & \qquad\qquad \{\text{Lemma 3.3.4}\} \\
 & \langle\, a \mid (a,a) \in A \wedge (a,a) \in R \mid (a,a)\, \rangle \\
= & \qquad\qquad \{A \nsubseteq R, \text{thus } (a,a) \notin R\} \\
 & \varnothing
\end{array}
$$

Lemma 3.3.5: *Point* $A \wedge A \nsubseteq R \Rightarrow A \circ R \circ A = \varnothing$

3.3.6 A symmetric composition that absorbs R

Lemma 3.3.6 shows that $(A \circ R \circ A) = A$ if $A \subseteq R$.

$$
\begin{array}{ll}
 & A \circ R \circ A \\
= & \qquad\qquad \{\text{Lemma 3.3.4}\} \\
 & \langle\, a \mid (a,a) \in A \wedge (a,a) \in R \mid (a,a)\, \rangle \\
= & \qquad\qquad \{A \subseteq R, \text{thus } (a,a) \in R\} \\
 & A
\end{array}
$$

Lemma 3.3.6: *Point* $A \wedge A \subseteq R \Rightarrow A \circ R \circ A = A$

3.3.7 Exponentiation of composition

Repeated composition of a relation with itself is called exponentiation of composition and it is denoted by a superscript. So R^3 means $R \circ R \circ R$.

The following is a recursive definition of exponentiation of relations:

$R^0 = E_U$
$R^n = R \circ R^{n-1}$ for $n \geq 1$

Note that, owing to associativity, R^n could equally well have been defined as $R^{n-1} \circ R$.

Lemma 3.3.7 shows by application of this definition that $R^1 = R$

In order to keep the formulae that built expression using several operators clean, the convention that composition has a higher priority than union and intersection is adopted; i.e. $R \cup S \circ T$ means $R \cup (S \circ T)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
& R^1 \\
= & \quad \{ \text{Definition of } R^n \} \\
& R \circ R^0 \\
= & \quad \{ \text{Definition of } R^0 \} \\
& R \circ E_U \\
= & \quad \{ E_U \text{ is a unit of } \circ \} \\
& R
\end{array}
$$

Lemma 3.3.7: $R^1 = R$

3.4 Properties of relations

3.4.1 Definitions

Consider relation R.

- R is said to be reflexive if $E_U \subseteq R$
- R is said to be antisymmetric if $R \cap R^{-1} \subseteq E_U$ — or at an element-wise level — if it holds that $(a,b) \in R \wedge (b,a) \in R \Rightarrow a = b$
- R is transitive if $R \circ R \subseteq R$ — or at a element-wise level — if it holds that $(a,b) \in R \wedge (b,c) \in R \Rightarrow (a,c) \in R$
- R is a partial order (PO) if it is reflexive, antisymmetric and transitive.
- R is idempotent if $R \circ R = R$.

3.4.2 Monotypes are idempotent

Lemma 3.4.2 shows that monotypes are idempotent.

3.4.3 Intersection preserves transitivity

If S and T are transitive, so is $S \cap T$. Lemma 3.4.3 shows this.

3.4.4 Composition preserves transitivity

The composition of transitive relation with itself is transitive. Lemma 3.4.4 shows this.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & A \circ A \\
= & \quad \{\text{Definition of } \circ\} \\
 & \langle\, a, b, c \mid (a,b) \in A \wedge (b,c) \in A \mid (a,c) \,\rangle \\
= & \quad \{\text{A is a monotype, thus } a = b = c\} \\
 & \langle\, a \mid (a,a) \in A \wedge (a,a) \in A \mid (a,a) \,\rangle \\
= & \quad \{\wedge \text{ is idempotent}\} \\
 & A
\end{array}
$$

Lemma 3.4.2: A is a monotype $\Rightarrow A \circ A = A$

$$
\begin{array}{ll}
 & (a,b) \in S \cap T \wedge (b,c) \in S \cap T \\
\Rightarrow & \quad \{\text{Definition of } \cap\text{, Definition of } \cap\} \\
 & ((a,b) \in S \wedge (a,b) \in T) \wedge ((b,c) \in S \wedge (b,c) \in T) \\
= & \quad \{\wedge \text{ is commutative and associative}\} \\
 & ((a,b) \in S \wedge (b,c) \in S) \wedge ((a,b) \in T \wedge (b,c) \in T) \\
\Rightarrow & \quad \{S \text{ is transitive, } T \text{ is transitive}\} \\
 & (a,c) \in S \wedge (a,c) \in T \\
= & \quad \{\text{Definition of } \cap\} \\
 & (a,c) \in S \cap T
\end{array}
$$

Lemma 3.4.3: S is transitive and T is transitive $\Rightarrow S \cap T$ is transitive

3.5 Representing Relations

The algorithms described in this thesis operate on binary relations. For this reason methods to store and manipulate binary relations are relevant to this thesis. This section discusses different representation models for binary relations that are commonly used. These are sets, lists, graphs and matrices.

3.5.1 Sets of pairs

A binary relation R on a set S is defined as a collection of pairs of elements of S. Listing or describing a relation in terms of a set of pairs is thus a representation model that is in direct correspondence with this definition of a relation.

For example if $S = \{a, b, c, d, e\}$, then the set $R = \{(a,b), (b,c), (c,a), (c,d)\}$ is a relation on S. This example is used as a running example for the different representation models that follows.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & (a,b) \in R \circ R \wedge (b,c) \in R \circ R \\
\Rightarrow & \{\text{Definition of } \circ, \text{Definition of } \circ\} \\
 & \langle \exists x_1 : x_1 \in U : (a,x_1) \in R \wedge (x_1,b) \in R \rangle \\
 & \wedge \langle \exists x_2 : x_2 \in U : (b,x_2) \in R \wedge (x_2,c) \in R \rangle \\
= & \{\ \exists \text{ distributes over } \wedge\ \} \\
 & \langle \exists x_1, x_2 : x_1, x_2 \in U : (a,x_1) \in R \wedge (x_1,b) \in R \wedge (b,x_2) \in R \wedge (x_2,c) \in R \rangle \\
\Rightarrow & \{\ \text{R is transitive}\ \} \\
 & \langle \exists x_1, x_2 : x_1, x_2 \in U : (a,x_1) \in R \wedge (x_1,x_2) \in R \wedge (x_2,c) \in R \rangle \\
\Rightarrow & \{\ \text{R is transitive}\ \} \\
 & \langle \exists x_2 : x_2 \in U : (a,x_2) \in R \wedge (x_2,c) \in R \rangle \\
\Rightarrow & \{\text{definition of } \circ\} \\
 & (a,c) \in R \circ R
\end{array}
$$

Lemma 3.4.4: R is transitive $\Rightarrow$ $R \circ R$ is transitive

3.5.2 Lists

Instead of storing all pairs of a relation R on a set S, one can store only the successors or predecessors of each element of S. If a relation is sparse this representation model is space efficient.

This representation model to store the successors or predecessors of each element of S is called a *successor list*. The following is the successor list representation of the running example. If an element has no successors, as is the case for d in this example, it's list is empty.

Element	Successors
a	b
b	c
c	a d
d	

A *predecessor list* can similarly be applied by storing the predecessors of each element in S. The following is the predecessor list representation of the running example:

Element	Predeccessors
a	c
b	a
c	b
d	c

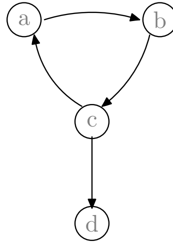
© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.5.3 Graphs

This relation can be visualised as a directed graph where the vertices correspond with the elements of S and the edges correspond with the elements of R. Each element $i \in S$ is represented by a node and each element $(i, j) \in R$ is represented by an arc pointing from the node i to the node j.

The following is the graph representation of the relation that is defined in set notation above.



3.5.4 Matrices

Relations can also be represented using an adjacency matrices i.e. a relation can be represented using a square Boolean matrix where the entry at (i, j) is 1 if $(i, j) \in R$ when represented as a set of pairs, or if there is an edge from vertex i to vertex j in the graph representation of R; otherwise the entry is 0. For example, the following is the matrix representation of the relation used as running example in this section:

	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	1	0	0	1
d	0	0	0	0

3.5.5 Correspondence between representation models

Transformations from one model to another are easily defined. Algorithms that are described in terms of manipulating successor lists and/or predecessor lists, are most easily described and implemented using linked lists. It is, however, equally feasible to use matrix representations for these algorithms as row i in the adjacency matrix of such relation represents the successor list of the ith node in the relation. Similarly, column i in the adjacency matrix of the relation represents the predecessor list of this node.

In Section 4.7 a mathematical proof is given that confirms that the adjacency matrix of a relation as defined here is indeed an isomorphism of the representation of the relation in terms of pairs. It is shown that the operations on the relation required to calculate the transitive closure of the relation corresponds with basic operations that can be performed on Boolean matrices.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.6 Functions

A function is a relation with the property that each element of its domain is related to at most one element in its codomain.

3.6.1 Defining a binary function

A binary relation f for which the following restriction on its elements holds, is called a *binary function*:

$$(x, y_0) \in f \wedge (x, y_1) \in f \Rightarrow y_0 = y_1$$

Because the second value in a pair (x, y), that is an element of a binary function f, is uniquely defined by the first value in the pair, $(x, y) \in f$ is often denoted by $f.x = y$. A binary function $f \subseteq X \times Y$, is denoted by $f : X \mapsto Y$. In this case X is the called the *domain* of f and Y is called its *range*. Note that it is not required that for each element of the domain of f there is a unique pair in f. If a unique pair is required, the function is called a total function.

Functions with higher arity are defined similarly as special kinds of relations with higher arity. This thesis does not refer to functions with higher arities. Therefore, when referring to a function without specifying its arity, it is assumed to be a binary function.

If $A \subseteq U$, then $f.A$ is defined by:

$$f.A = \langle\, a \mid a \in A \mid f.a \,\rangle$$

3.6.2 Surjection

Let f be a function $f : A \mapsto B$, then f is said to be a *surjection* (or surjective map) if, for any $b \in B$, there exists an $a \in A$ for which $b = f.a$. A surjection is sometimes referred to as being *onto*.

3.6.3 Injection

Let f be a function $f : A \mapsto B$, then f is said to be an *injection* (or injective map, or embedding) if $f.x = f.y \Rightarrow x = y$. Equivalently, $x \neq y \Rightarrow f.x \neq f.y$. In other words, f is an injection if it maps distinct objects to distinct objects. An injection is sometimes also called *one-to-one*.

3.6.4 Total function

Let f be a function $f : A \mapsto B$, then f is said to be a *total function* if f has a unique pair for each element in A. Note that the definitions for surjection, injection applies to all functions. Thus the function need not be a total function to be an injection or surjection.

3.6.5 Bijection

A total function is called *bijective* if it is both injective and surjective. A bijective function is also called a *bijection*.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

3.6.6 Inverse of a bijection

The inverse of a function, when seen as a relation as defined above, is not necessarily a function. However, a bijection f admits an inverse f^{-1} that is a function. The inverse of a bijection $f : A \mapsto B$ is the function $f^{-1} : B \mapsto A$ defined by

$$f^{-1} = \langle\, a, b \mid f.a = b \mid (b, a) \,\rangle$$

3.6.7 Operations on functions

Because functions are relations they can be joined, intersected, composed and exponentiated in the same way as relations.

3.7 Summary

This chapter presents typical textbook knowledge in a condensed fashion to serve as a concise introduction to the problem domain. Only the essential mathematical preliminaries needed to follow the arguments in the remainder of Part I and in Part III of this thesis are presented.

Familiar concepts that are key in the correctness proofs of algorithms, such as de Morgans's laws and idempotence are simply stated while less familiar concepts such as monotype and point are explained using examples.

The reader should take note that the definition of composition of relations, denoted by $\circ$, is not consistent across fields with respect to the order in which it is applied. To avoid confusion one may write $\circ_l$ and $\circ_r$ explicitly when necessary, depending whether the left or the right relation is the first one applied. This convention is followed by Kilp *et al.* [137]. In this thesis it was decided to define $\circ$ to be $\circ_l$.

The lemmata in Sections 3.4 and 3.3 extend that information about relations that is usually found in textbooks. The outcome of specific operations are proved (such as $A \circ R \circ A$ for any relation R and arbitrary set A). Also proved are interesting properties of some key operations on relations. These are used in later arguments in this thesis. The property that transitivity is preserved when these operations are performed is of particular interest. These proofs are original; i.e., they have been worked out specifically as part of this thesis to support correctness arguments in Part III.

When implementing an algorithm, the data on which the algorithm operates has to be stored and manipulated in memory. Different algorithms that calculate the transitive closure of relations may use different representation models for relations. For this reason the different models and the correspondence between them are introduced and explained in Sections 3.5. The proof of the correspondence between set representation and the use of adjacency matrices is postponed to Section 4.7 while Chapter 4 discusses the mathematical foundation and the theories of matrices that are relevant to this thesis. The algorithms discussed in Part III use adjacency matrices as their representation model.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 4

Matrices

This thesis deals with algorithms that apply operations on square Boolean matrices to determine the transitive closure of a given relation. This chapter provides the mathematical definitions of matrices, the operations on them as well as relations between them that are needed to describe the algorithms in Part III of this thesis.

Section 4.7 provides original mathematical reasoning to show how boolean operations on square Boolean matrices correspond with the mathematical definitions of relational operations shown in Section 3.3. It also shows how the properties of the boolean operations that are discussed in this chapter correspond with the properties of relational operations shown in Section 3.3. The ability to transform operations on matrices to relational operations and vice versa as well as the tight correspondence between relations and boolean matrices is used in later arguments that show the correctness of algorithms discussed in Part III of this thesis.

4.1 Definition

A *matrix* P is a non-empty finite ordered sequence of members of a set U. U is called the data type of P and the items in the sequence are called the *entries* of the matrix. In many texts P is called a *vector*.

The *size of a matrix* P is defined to be the number of entries in the sequence. Note that the entries need not be unique. The set of all matrices of size n with data type U is denoted by $U[n]$.

4.2 Notation for matrices

Since a matrix is a sequence, the notation for sequences as defined in Section 2.1.6 is applied. Lowercase letters subscripted with the elements of $\mathbb{N}$ are used to refer to the entries of a matrix. Generally the letter chosen to refer to the entries of a matrix corresponds with the uppercase letter that is used to refer to a matrix. For example if $P \in U[n]$, it is assumed that p_i refers to the ith entry of P without explicitly stating that $P = \langle\!\langle\ i \mid i \in \mathbb{N}_n \mid p_i\ \rangle\!\rangle$. Likewise q_i refers to the ith entry of a matrix Q.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

4.3 Two-dimentional matrices

4.3.1 Dimensions

The data type of a matrix may be any set, including a set of matrices. For example let $P \in U[m]$ and $Q \in M[n]$. It follows that $Q \in (U[m])[n]$. To simplify notation $(U[m])[n]$ is written as $U[m,n]$. In this case Q is called a two-dimensional matrix. For example $Q = ((2,8),(5,0),(1,4)) \in \mathbb{N}[2,3]$. Similarly an n-dimensional matrix can be defined as a matrix of matrices nested n levels deep.

An entry of a nested matrix is referred to by a variable with an index value which is an ordered string of subscripts. Each character in the subscript string corresponds with a nested level. For example $Q = ((u_{00},u_{01}),(u_{10},u_{11}),(u_{20},u_{21})) \in U[2,3]$. In the case of a n-dimensional matrix, the subscript string contains n characters.

This thesis deals only with matrices with one or two dimensions. The term *vector* is used to refer to a one-dimensional matrix and it should be assumed that when the term *matrix* is used without specifying its dimension, it refers to a two-dimensional matrix.

In the general matrix $S \in U[m,n]$, the token s_{ij} is used to represent the jth entry in the vector $\langle\!\langle\ k \mid k \in \mathbb{N}_n \mid s_{ik}\ \rangle\!\rangle$. This vector can be expressed as the ith entry in $\langle\!\langle\ t \mid t \in \mathbb{N}_m \mid \ \langle\!\langle\ k \mid k \in \mathbb{N}_n \mid s_{tk}\ \rangle\!\rangle\ \rangle\!\rangle$. This two-dimensional matrix can also be described by specifying its members by extension — that is, listing the members ordered in rows and columns and enclosed in square brackets as follows:

$$\begin{bmatrix} s_{00} & s_{01} & s_{02} & \cdots & s_{0n} \\ s_{10} & s_{11} & s_{12} & \cdots & s_{1n} \\ s_{20} & s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ s_{m0} & s_{m1} & s_{m2} & \cdots & s_{mn} \end{bmatrix}$$

By convention the order of the indices for the entries in a matrix is the row index followed by the column index. Rows are numbered top to bottom and columns are numbered left to right.

4.3.2 Alternate notation for matrix entries

In cases where the use of subscripts will lead to notationally complex expressions or confusion p_{ij} may be written as $P[i,j]$. For example when there is a need to refer to the entry $p_{k_{i+1}k_{j-1}}$ of the matrix P, it is rather written as $P[k_{i+1},k_{j-1}]$.

Another reason for using this alternate notation is when subscripts are used to refer to different matrices, for example $M_0 \in U[n,n]$ and $M_1 \in U[n,n]$ are different matrices. In such a case, subscripts are not available to refer to the entries in these matrices. Here the use of this notation is the only option available to refer to the entries of these matrices. Thus the entry in the ith row and j^{th} in matrix M_0 is represented by $M_0[i,j]$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

This alternate notation is also required when referring to the elements of a composite matrix. In Section 4.6.4 the concept of a composite matrix is introduced and an example of the use of this notation for the composite matrix is given.

4.3.3 Square matrices

A matrix is called *square* if the number of rows in the matrix is equal to the number of columns in the matrix.

In a square matrix $M \in U[n,n]$, the vector $D = \langle\!\langle\ i,j \mid i,j \in \mathbb{N}_n, i = j \mid m_{ij}\ \rangle\!\rangle$ is called the *main diagonal* of M. The main diagonal of a square matrix is the diagonal which runs from the top left corner to the bottom right corner when specifying its members by extension. For example, the following matrix has ones down its main diagonal:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A square matrix like the above in which the entries outside the main diagonal are all zero is called a *diagonal matrix*.

In a square matrix $M \in U[n,n]$, the vector $D = \langle\!\langle\ i,j \mid i,j \in \mathbb{N}_n, i + j = n \mid m_{ij}\ \rangle\!\rangle$ is called the *minor diagonal* of M. The minor diagonal of a square matrix is the diagonal which runs from the top right corner to the bottom left corner of the matrix when specifying its members by extension.

4.3.4 Boolean matrices

Recall that $\mathbb{B}$ denotes the set of Booleans. Therefore the vector $P \in \mathbb{B}[n]$ has n Boolean entries. It is called a Boolean vector. Likewise $Q \in \mathbb{B}[m,n]$ is called a Boolean matrix. In fact, an $m \times n$ Boolean matrix specifies a truth value for each of the pairs in $\mathbb{N}_m \times \mathbb{N}_n$. For $n \in \mathbb{N}^+$, a matrix $P \in \mathbb{B}[n]$ is called a *Boolean matrix* of size n.

4.3.5 Some special square Boolean matrices

The following are the definitions of special square Boolean matrices in $\mathbb{B}[n,n]$:

The *always false* matrix $\mathbb{O}_n$ defined by
$\mathbb{O}_n = \langle\!\langle\ i,j \mid i,j \in \mathbb{N}_n \mid 0\ \rangle\!\rangle$,

the *always true* matrix $\mathbb{J}_n$ defined by
$\mathbb{J}_n = \langle\!\langle\ i,j \mid i,j \in \mathbb{N}_n \mid 1\ \rangle\!\rangle$,

and the *identity* matrix $\mathbb{I}_n$ defined by
$\mathbb{I}_n = \langle\!\langle\ i,j \mid i,j \in \mathbb{N}_n \mid i = j\ \rangle\!\rangle$.

In this definition $=$ is a logical operator on numbers, i.e. $7 = 7$ evaluates to *true* and $7 = 1$ evaluates to *false*. Thus the identity matrix has 0's everywhere except in the main diagonal where it has 1's. It is a diagonal boolean matrix.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

For $A \subseteq \mathbb{N}_n$, define $\mathbb{I}_A$ as follows:
$\mathbb{I}_A = \langle\!\langle\ i,j \mid i,j \in \mathbb{N}_n \mid i = j \wedge i \in A\ \rangle\!\rangle$
For example;

$$A = \{0,3,4\} \Rightarrow \mathbb{I}_A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

For $0 \leq k < n$, $\mathbb{I}_{\{k\}}$ is a square Boolean matrix with exactly one *true* entry, where this *true* entry corresponds with $\mathbb{I}_n[k,k]$. i.e.
$\mathbb{I}_{\{k\}} = \langle\!\langle\ i,j \mid i,j \in \mathbb{N}_n \mid (i = j) \wedge (i = k)\ \rangle\!\rangle$.

4.4 Submatrices

4.4.1 Definition

A submatrix S of a matrix M is denoted by $S \subseteq M$. A submatrix S of M can be constructed by singling out some elements of M and imposing an ordering on these entries. The notation for intentional definition of a sequence as defined in Section 2.1.6 is applied when defining a submatrix. Typically when defining $S \subseteq M$, the definition of S is done in terms of the elements of M. S assumes the range predicate of M and strengthens the predicate through the specification of one or more additional predicates.

When defining a submatrix of a two-dimensional matrix M one would refer to the entries in M using the conventional index values. Thus m_{ij} would refer to the entry in the ith row and j^{th} column of M. The order imposed on the entries in the submatrix is specified in the order in which the dummy elements are given. For example; given $M \in U[n,n]$, the following two submatrices of M involving only the entries in M above its main diagonal can be defined:

$$P = \langle\!\langle\ i,j \mid i < j \mid m_{ij}\ \rangle\!\rangle$$
$$Q = \langle\!\langle\ j,i \mid i < j \mid m_{ij}\ \rangle\!\rangle$$

Both P and Q comprises the same entries in M, however, the elements of P are in row major order of M while the elements of Q are in column major order.

The definition of a submatrix allows for arbitrary ordering of entries in a submatrix while this notation does not provide means to impose an order other than row major order or column major order. This limitation is, however, not a problem as none of the algorithms discussed in this thesis requires orderings beyond these.

4.4.2 Combining, intersecting and subtracting submatrices

The binary operations that are defined for sets namely union ($\cup$), intersection ($\cap$) and difference ($-$) are applicable to submatrices.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Let P and Q be submatrices of $M \in U[n, m]$ that have the same ordering of their elements. The following defines these operations:

$$\begin{aligned} m_{ij} \in P \cup Q &\equiv m_{ij} \in P \vee m_{ij} \in Q \\ m_{ij} \in P \cap Q &\equiv m_{ij} \in P \wedge m_{ij} \in Q \\ m_{ij} \in P - Q &\equiv m_{ij} \in P \wedge m_{ij} \notin Q \end{aligned}$$

If the elements of two submatrices have different orderings, the operations are undefined.

The following are the definitions of M and a number of submatrices of M shown in Figure 4.4.2.

$$\begin{aligned} M &= \langle\!\langle\ i, j \mid i \in \mathbb{N}_5, j \in \mathbb{N}_5 \mid m_{ij}\ \rangle\!\rangle \\ S &= \langle\!\langle\ i, j \mid 0 < i < 3 \mid m_{ij}\ \rangle\!\rangle \\ T &= \langle\!\langle\ i, j \mid i = 4 \mid m_{ij}\ \rangle\!\rangle \\ V &= \langle\!\langle\ i, j \mid 0 < j < 3 \mid m_{ij}\ \rangle\!\rangle \end{aligned}$$

M

$$\begin{bmatrix} m_{00} & \mathbf{m}_{01} & \mathbf{m}_{02} & \mathbf{m}_{03} & \mathbf{m}_{04} \\ m_{10} & m_{11} & \mathbf{m}_{12} & \mathbf{m}_{13} & \mathbf{m}_{14} \\ m_{20} & m_{21} & m_{22} & \mathbf{m}_{23} & \mathbf{m}_{24} \\ m_{30} & m_{31} & m_{32} & m_{33} & \mathbf{m}_{34} \\ m_{40} & m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$$

S

$$\begin{bmatrix} m_{10} & m_{11} & \mathbf{m}_{12} & \mathbf{m}_{13} & \mathbf{m}_{14} \\ m_{20} & m_{21} & m_{22} & \mathbf{m}_{23} & \mathbf{m}_{24} \end{bmatrix}$$

T

$$\begin{bmatrix} m_{40} & m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix}$$

V

$$\begin{bmatrix} \mathbf{m}_{01} & \mathbf{m}_{02} \\ m_{11} & \mathbf{m}_{12} \\ m_{21} & m_{22} \\ m_{31} & m_{32} \\ m_{41} & m_{42} \end{bmatrix}$$

Figure 4.4.2: M and submatrices of M shown by extension

If P is the submatrix of M defined in Section 4.4.1, the elements of $M \cap P$, $S \cap P$ and $V \cap P$ are shown in blue in Figure 4.4.2. The elements of M, S and V shown in black can respectively be defined as $M - P$, $S - P$ and $V - P$. It can be observed that $T \cap P = \varnothing$ and $T - P = T$.

4.4.3 Commonly used submatrices

Commonly used submatrices of a matrix $P \in U[m, n]$, are the submatrices comprising a row, a column or a diagonal of the matrix P. The submatrix $\langle\!\langle\ j \mid j \in \mathbb{N}_n \mid p_{ij}\ \rangle\!\rangle$ is called the ith *row* of P. Note that in this definition i is a constant. The shorthand $P[i, \star]$ is sometimes used to refer to $\langle\!\langle\ j \mid j \in \mathbb{N}_n \mid p_{ij}\ \rangle\!\rangle$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Similarly, the vector $\langle\!\langle\ i \mid i \in \mathbb{N}_m \mid m_{ij}\ \rangle\!\rangle$ is a submatrix of $P \in U[m,n]$ that singles out the entries belonging to a specific column in the matrix. It is called the jth *column* of P and may be referred to using the expression $P[\star, j]$.

If P is square, the submatrix $\langle\!\langle\ i,j \mid i = j \mid m_{ij}\ \rangle\!\rangle$ is called the *diagonal* of P.

$P_{\mathbb{I}}$ may be used as a shorthand for $\langle\!\langle\ i,j \mid i = j \mid p_{ij}\ \rangle\!\rangle$.

4.5 Partitions

A partition of a set X is a division of X as a union of non-overlapping and non-empty subsets called cells [86]. More formally, the subsets comprising the partition are both collectively exhaustive and mutually exclusive with respect to the set being partitioned.

Here the concept of a partition of a set is extended to a new concept, namely a partition of a matrix. An important requirement when specifying this concept for matrices is that ordering of elements in matrices matters. For this reason we specify that the cells comprising the partition of a matrix M should be submatrices of M, thus the order of the elements in each cell matters. Furthermore, we specify that the cells themselves in the partition should also ordered. The partition of a matrix is thus a *sequence* of submatrices that are both collectively exhaustive and mutually exclusive with respect to the matrix being partitioned.

The context where partitions of matrices are used differs from the context where partitions of sets are used. Set partitions are applied in probability theory. When partitioning sets to determine probabilities, the number of different subsets in a given partition is important. Therefore the requirement that the subsets, that are considered, should not be empty plays a crucial role. On the other hand, when forming a partition of a matrix in the context of this thesis, the number of submatrices is known. The submatrices are formed with the purpose of specifying an ordering of the entries in the matrix. Their union should comprise the matrix to ensure that all entries are included. They should also be mutually exclusive to avoid unnecessary repetition. There is, however, no practical reason why they have to be non-empty. The inclusion of a number of empty submatrices in a partition in the context of its application in this thesis has no effect.

A partition P of a matrix M is specified as a sequence $\langle\!\langle\ t \mid p_t \subseteq M \mid p_t\ \rangle\!\rangle$ that complies with the following:

$$\langle \bigcup t \mid p_t \in P \mid p_t \rangle = \mathrm{M}$$
$$s \neq t \wedge p_s \in P \wedge p_t \in P \Rightarrow p_s \cap p_t = \varnothing$$

Andrews [10] uses the notation $\lambda \vdash n$ to indicate that the sequence λ is a partition of the number n. Although the meaning of a partition in the context Andrews used it, is different from the meaning of Erdös and Rado's [86] partition of a set and the meaning of a partition of a matrix used in this thesis, the same notation is applied. Thus $P \vdash M$ is a shorthand for $P = \langle\!\langle\ t \mid p_t \subseteq M \mid p_t\ \rangle\!\rangle$ complies with the requirements to be a partition of M as defined here.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The algorithms in Chapters 16 and 17 perform an operation on the elements of a square Boolean matrix in an order that is specified in terms of a partition of the matrix.

4.6 Operations

This section considers matrices with entries in U, where the data type U is equipped with two binary operators say $+$ and $\times$ with $+ : U \times U \mapsto U$ and $\times : U \times U \mapsto U$ such that $\times$ distributes over $+$. In this section these operators are promoted to the matrix level.

4.6.1 Addition

The + operator is defined on two matrix operands whose number of rows and number of columns correspond. Given two matrices $P, Q \in U[m, n]$, the sum $S = P + Q$ is defined as a matrix $S \in U[m, n]$ where

$$S = \langle\!\langle\, i, j \mid i \in \mathbb{N}_m \wedge j \in \mathbb{N}_n \mid p_{ij} + q_{ij} \,\rangle\!\rangle.$$

4.6.2 Multiplication

The $\times$ operator is defined for matrices where the number of rows in the first matrix is equal to the number of columns in the second matrix. Given the matrices $P \in U[m, n]$ and $Q \in U[n, s]$, the product $T = P \times Q$ is defined as a matrix $T \in U[m, s]$ where

$$T = \langle\!\langle\, i, k \mid i \in \mathbb{N}_m \wedge k \in \mathbb{N}_s \mid \langle \textstyle\sum j \mid j \in \mathbb{N}_n \mid p_{ij} \times q_{jk} \rangle \,\rangle\!\rangle$$

4.6.3 Attributes of operations

Matrix addition is element-wise addition of the entries in the matrices and will inherit the properties, such as commutativity and associativity of the + operator on the datatype of the matrix.

Matrix multiplication, on the other hand, is not element-wise multiplication of the entries and is generally not commutative. In fact, if $T = M_1 \times M_2$, it is likely that $M_2 \times M_1 \neq T$. If M_1 and M_2 are not square, $M_2 \times M_1$ is not even defined.

It can be shown that matrix multiplication is associative and that matrix multiplication distributes over matrix addition from left as well as from right. It is assumed that matrix multiplication has higher priority than matrix addition. For example $M_0 + M_1 \times M_2$ means $M_0 + (M_1 \times M_2)$.

4.6.4 Composite matrices

A composite matrix is an expression specifying a matrix in terms of operations on two or more matrices. For example; the matrix that is the result of multiplying matrix M_0 with matrix M_1 can be specified as the matrix $(M_0 \times M_1)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The alternate notation described in Section 4.3.2 is used to refer to the entries in composite matrices. For example the entry in the ith row and the jth column of a composite matrix such as $M_0 + M_1$ is denoted by $(M_0 + M_1)[i, j]$.

4.6.5 Addition and multiplication of Boolean matrices

The operators $\vee$ and $\wedge$ on Boolean values satisfy the fact that $\wedge$ distributes over $\vee$. This allows us to promote these operators to the matrix level with the following definitions for these operations:

For matrices $P \in \mathbb{B}[m, n]$ and $Q \in \mathbb{B}[m, n]$, the sum $P + Q \in \mathbb{B}[m, n]$ is defined by

$$P + Q = \langle\!\langle\, i, j \mid i \in \mathbb{N}_m \wedge j \in \mathbb{N}_n \mid p_{ij} \vee q_{ij} \,\rangle\!\rangle$$

For matrices $P \in \mathbb{B}[m, n]$ and $Q \in \mathbb{B}[n, s]$ the product $P \times Q \in \mathbb{B}[m, s]$ is defined by

$$P \times Q = \langle\!\langle\, i, k \mid i \in \mathbb{N}_m \wedge k \in \mathbb{N}_s \mid \langle\, \exists j \mid j \in \mathbb{N}_n \mid p_{ij} \wedge q_{jk} \,\rangle \,\rangle\!\rangle$$

Note that $\exists$ is the quantifier symbol for the commutative associative binary operation $\wedge$ as discussed in Section 2.1.5 and shown Table A4 in the appendix.

The special square Boolean matrices $\mathbb{I}_n$ and $\mathbb{O}_n$ defined in Section 4.3.5 are units for these operations. $\mathbb{I}_n$ is a unit for matrix multiplication and $\mathbb{O}_n$ is a unit for matrix addition, i.e. $M \times \mathbb{I}_n = \mathbb{I}_n \times M = M$ and $M + \mathbb{O}_n = \mathbb{O}_n + M = M$ for all $M \in \mathbb{B}[n, n]$.

4.6.6 Exponentiation

Repeated multiplication of a square matrix with itself is called exponentiation and it is denoted by a superscript. So M^3 means $M \times M \times M$. Exponentiation of a matrix $M \in \mathbb{B}[n]$ is defined by the following recursive definition:

$$M^0 = \mathbb{I}_n$$
$$M^i = M \times M^{i-1} \text{ for } i \geq 1$$

The following shows by application of this definition that $M^1 = M$:

$$\begin{aligned} M^1 &= M \times M^0 \\ &= M \times \mathbb{I}_n \\ &= M \end{aligned}$$

Note that, owing to associativity, M^i could equally well have been defined as $M^{i-1} \times M$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

4.6.7 Multiplying with $\mathbb{I}_A$

The composite matrices involving multiplication of a square Boolean matrix with the special matrix $\mathbb{I}_{\{k\}}$ that is defined in Section 4.3.5 is of particular interest. The following shows that all the entries in $M \times \mathbb{I}_{\{k\}}$ are 0, except for the entries in the kth **column**. The entries in this column are equal to those in the kth column of M. For example the following calculation show this for $M \in \mathbb{B}[4,4]$ and $k = 2$:

$$M \times \mathbb{I}_{\{2\}} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & m_{02} & 0 \\ 0 & 0 & m_{12} & 0 \\ 0 & 0 & m_{22} & 0 \\ 0 & 0 & m_{32} & 0 \end{bmatrix}$$

Note that $(M \times \mathbb{I}_{\{2\}})[2,1] = 0$ was calculated by evaluating the expression

$$(m_{20} \wedge 0) \vee (m_{21} \wedge 0) \vee (m_{22} \wedge 0) \vee (m_{23} \wedge 0)$$

Similarly $(M \times \mathbb{I}_{\{2\}})[0,2] = m_{02}$ was calculated by evaluating the expression

$$(m_{00} \wedge 0) \vee (m_{01} \wedge 0) \vee (m_{02} \wedge 1) \vee (m_{03} \wedge 0)$$

Likewise the entries in $\mathbb{I}_{\{k\}} \times M$ are 0, except for the entries in the kth **row** that correspond with the entries in the kth row of M.

A property that is of particular interest is:

$$\mathbb{I}_{\{i,j\}} = \mathbb{I}_{\{i\}} + \mathbb{I}_{\{j\}} \tag{4.1}$$

For $A \subseteq \mathbb{N}_n$, the following expression can thus be applied to calculate $\mathbb{I}_A$:

$$\mathbb{I}_A = \langle\, \sum a \mid a \in A \mid \mathbb{I}_{\{a\}} \,\rangle$$

4.7 From relations to Boolean matrices

In Section 3.5.4 it was mentioned that a relation can be represented in terms of its adjacency matrix. This is a fundamental concept in this thesis as it deals with algorithms that determine the transitive closure of a given homogeneous relation. The algorithms discussed in Part III of this thesis achieve this through the application of operations on two-dimensional Boolean matrices.

Using matrices to perform the needed calculations instead of performing them on the relations is possible because homogeneous relations and two-dimensional Boolean matrices are isomorphic.

In this section the mathematical definition of an isomorphism is given. This concept is then used to show the 1-to-1 correspondence between a homogeneous relation and a two-dimensional Boolean matrix through a series of isomorphisms. First relations are expressed as predicates. Then the correspondence between universal relations and relations on $\mathbb{N}_n$ is established. The combination of these two is applied to define an isomorphism that can be applied to transform a relation to a square Boolean matrix.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

4.7.1 Isomorphic sets

Two sets X and Y are isomorphic, notation $X \cong Y$, if there is a function, say $\varphi : X \mapsto Y$ and a function, say $\psi : Y \mapsto X$ such that $\varphi \circ \psi = E_Y$ and $\psi \circ \varphi = E_X$, i.e. each element of X is in a 1-to-1-correspondence with an element of Y. As an example, a finite set U is isomorphic to $\mathbb{N}_{|U|}$. Any enumeration of U can be applied. The 1-to-1-correspondence between such enumeration and the elements of $\mathbb{N}_{|U|}$ establishes that these sets are isomorphic.

4.7.2 From relations to predicates

The first step to show the correspondence between relations and Boolean matrices is to show that a 1-to-1-correspondence between the homogeneous relations on U and predicates on $U \times U$ can be constructed. One such correspondence that is completely determined by type considerations, is given by a pair of functions:

$$\varphi : \mathcal{P}(U \times U) \mapsto (U \times U \to \mathbb{B})$$
$$\psi : (U \times U \to \mathbb{B}) \mapsto \mathcal{P}(U \times U)$$

φ is defined to be a function that assigns an element of $U \times U \to \mathbb{B}$ to each subset of $U \times U$. Recall subsets of a Cartesian product are called relations. Thus this function assigns an element of $U \times U \to \mathbb{B}$ to each relation $R \subseteq U \times U$. The element assigned to a relation R is denoted by $\varphi.R$. However, $\varphi.R$ is an element of $U \times U \to \mathbb{B}$. Hence, it has to assign a truth value to each pair $(x, y) \in U \times U$.

The value that is assigned to (x, y) by $\varphi.R$ is denoted by $(\varphi.R).(x, y)$. This value is true if and only if $(x, y) \in R$. The definition of φ can thus be expressed as follows:

$$(\varphi.R).(x, y) \equiv (x, y) \in R$$

Conversely, ψ is defined to be a function that assigns a subset of $U \times U$ to each element of $U \times U \to \mathbb{B}$. Let p be an element of $U \times U \to \mathbb{B}$. The relation assigned to p by ψ is denoted by $\psi.p$. $\psi.p$ is a relation on U. In other words $\psi.p$ is a set of pairs in $U \times U$.

$\psi.p$ is defined by specifying that (u, v) is an element of $\psi.p$ if and only if $p.(u, v)$. The definition of ψ can thus be expressed as follows:

$$(u, v) \in \psi.p \equiv p.(u, v)$$

An isomorphism between $\mathcal{P}(U \times U)$ and $(U \times U \to \mathbb{B})$ is formed by φ and ψ. As a consequence, this isomorphism enables a predicate view on relations. Having an alternative view on objects has some advantages, but in practical situations these objects will be subject to operations in the original view.

The next objective is to find operations in the alternative view whose behaviour corresponds to the operations on the original view. i.e. subsets, union, intersection and composition of relations have to be expressed in predicates.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The aim is to find a compositional way to express these as predicates, i.e. operations, say $\hat{\subseteq}, \hat{\cup}, \hat{\cap}, \hat{\circ}$ are sought such that

$$\begin{array}{lcl} \varphi.(R \subseteq S) & = & \varphi.R \mathbin{\hat{\subseteq}} \varphi.S \\ \varphi.(R \cup S) & = & \varphi.R \mathbin{\hat{\cup}} \varphi.S \\ \varphi.(R \cap S) & = & \varphi.R \mathbin{\hat{\cap}} \varphi.S \\ \varphi.(R \circ S) & = & \varphi.R \mathbin{\hat{\circ}} \varphi.S \end{array}$$

It is possible to derive definitions for these predicate operations. Lemma 4.7.2 shows this for $\hat{\circ}$.

$$\begin{array}{ll} & \varphi.(R \circ S).(u,v) \\ = & \{\text{Definition of } \varphi\} \\ & (u,v) \in R \circ S \\ = & \{\text{Definition of } \circ\} \\ & \langle\, \exists z \mid \;\mid (u,z) \in R \wedge (z,v) \in S \,\rangle \\ = & \{\text{Definition of } \varphi\} \\ & \langle\, \exists z \mid \;\mid (\varphi.R).(u,z) \wedge (\varphi.S).(z,v) \,\rangle \\ = & \{\text{Definition of } \hat{\circ}\} \\ & (\varphi.R \mathbin{\hat{\circ}} \varphi.S).(u,v) \end{array}$$

Lemma 4.7.2: $\varphi.(R \circ S) = \varphi.R \mathbin{\hat{\circ}} \varphi.S$

By using the result of Lemma 4.7.2, and similar derivations for $\hat{\subseteq}$, $\hat{\cup}$ and $\hat{\cap}$ the following definitions are produced:

$$\begin{array}{l} p \mathbin{\hat{\circ}} q = \langle\, \exists z \mid \;\mid p.(u,z) \wedge q.(z,v) \,\rangle \\ p \mathbin{\hat{\subseteq}} q = \langle\, \forall u,v \mid u,v \in U \mid p.(u,v) \Rightarrow q.(u,v) \,\rangle \\ (p \mathbin{\hat{\cup}} q).(u,v) \;=\; p.(u,v) \vee q.(u,v) \\ (p \mathbin{\hat{\cap}} q).(u,v) \;=\; p.(u,v) \wedge q.(u,v) \end{array}$$

In the special case when dealing with homogeneous relations on $\mathbb{N}_n$ these predicates coincide with the square Boolean matrices $\mathbb{B}[n,n]$. It is clear that the operations $\hat{\subseteq}$, $\hat{\cup}$ and $\hat{\circ}$ respectively coincide with $\sqsubseteq$ (the subrelation-matrix relation defined in Section 4.8), matrix addition (defined in Section 4.6.1) and matrix multiplication (defined in Section 4.6.2).

4.7.3 From relations on U to relations on $\mathbb{N}_n$

In the previous section a neat 1-to-1-correspondence between universal relations (i.e. relations on a universal set U) and predicates was established. However, the quest is to find the correspondence between homogeneous relations and two-dimensional Boolean matrices.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

In this section the correspondence between homogeneous relations on U and relations on $\mathbb{N}_n$ is explored to create a route to find the correspondence between homogeneous relations on U and square Boolean matrices.

In Section 4.7.1 it was mentioned that finite sets are isomorphic to a begin segment of the naturals. This means that when the elements of a finite set are numbered, one can use the numbers to refer to the elements instead of referring to the elements themselves. This isomorphy can now be extended to homogeneous relations on finite sets as follows:

Fix an enumeration of U in bijection
$\varepsilon : U \mapsto \mathbb{N}_n$
and consider the functions
$\gamma : \mathcal{P}(U \times U) \mapsto \mathcal{P}(\mathbb{N}_n \times \mathbb{N}_n)$ and
$\delta : \mathcal{P}(\mathbb{N}_n \times \mathbb{N}_n) \mapsto \mathcal{P}(U \times U)$
defined by
$\gamma.R = \langle\, u, v \mid (u, v) \in R \mid (\varepsilon.u\, , \, \varepsilon.v) \,\rangle$
$\delta.Z = \langle\, i, j \mid (i, j) \in Z \mid (\varepsilon^{-1}.i\, , \, \varepsilon^{-1}.j) \,\rangle$

γ and δ establish an isomorphism. Hence relations on $\mathbb{N}_n$ give an alternative view on relations on U. The advantage of this transformation is that changing the view is only a change in appearance of the objects and not in the operations involved.

The domain of γ is the codomain of δ and vice versa. As both the domains and the codomains of these functions are relations, the operations on the elements in these sets are the same in the domain as in the codomain. i.e.

$$\begin{array}{lcl} \gamma.(R \subseteq S) & = & \gamma.R \subseteq \gamma.S \\ \gamma.(R \cup S) & = & \gamma.R \cup \gamma.S \\ \gamma.(R \cap S) & = & \gamma.R \cap \gamma.S \\ \gamma.(R \circ S) & = & \gamma.R \circ \gamma.S \end{array}$$

4.7.4 Transforming from a relation to a Boolean matrix

The isomorphisms that were established in the previous two subsections provide for a transformation from a relation to a Boolean matrix. In Section 4.7.2 a 1-to-1-correspondence between the homogeneous relations on U and the homogeneous relations on $U \times U \to \mathbb{B}$ was established. In Section 4.7.3 a trivial transformation from $U \times U$ to $\mathbb{N}_n \times \mathbb{N}_n$ was established. They are now combined to form a transformation from relations on $U \times U$ to square Boolean matrices.

Per definition an isomorphism is a bijection. Since the composition of two bijections is a bijection [118], it follows naturally that isomorphisms are closed under composition. Both the transformations given in the preceding sections are isomorphisms, therefore their composition is an isomorphism. Having established these isomorphisms we can now transform any relation to a Boolean matrix with will defined operations that corresponds with operations on the relation.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

For finite U of size n, introduce the following:

$$\Phi : \mathcal{P}(U \times U) \mapsto \mathbb{B}[n,n] \text{ defined by } \Phi = \varphi \circ \gamma$$
$$\Psi : \mathbb{B}[n,n] \mapsto \mathcal{P}(U \times U) \text{ defined by } \Psi = \delta \circ \psi$$

The following holds:

$$\begin{array}{lcll}
\Phi.(R \circ S) & = & \Phi.R \times \Phi.S & \\
\Phi.(R \cup S) & = & \Phi.R + \Phi.S & \\
\Phi.(R^m) & = & (\Phi.R)^m & \\
\Phi.E_U & = & \mathbb{I}_n & \\
\Phi.E_{\{a\}} & = & \mathbb{I}_{\{k\}} & \text{where } \varepsilon.a = k \\
\Phi.E_A & = & \mathbb{I}_B & \text{where } B = \langle a : a \in A : \varepsilon.a \rangle
\end{array}$$

Lemmata 4.7.5 and 4.7.6 serves as examples of how each of these correspondences follows trivially from the definitions of the entities and the transformations involved.

4.7.5 Union of relations vs addition of matrices

When transforming relations to square Boolean matrices, the union of the relations corresponds with the addition of matrices. Lemma 4.7.5 confirms this.

$$\begin{array}{ll}
& \Phi.(R \cup S) \\
= & \{\text{Definition of } \Phi\} \\
& \varphi \circ \gamma.(R \cup S) \\
= & \{\text{Definition of } \gamma\} \\
& \varphi.(\gamma.R \cup \gamma.S) \\
= & \{\text{Definition of } \varphi\} \\
& \varphi \circ \gamma.R \ \hat{\cup} \ \varphi \circ \gamma.S \\
= & \{\text{Definition of } \hat{\cup}\} \\
& \varphi \circ \gamma.R + \varphi \circ \gamma.S \\
= & \{\text{Definition of } \Phi\} \\
& \Phi(R) + \Phi(S)
\end{array}$$

Lemma 4.7.5: $\Phi(R \cup S) = \Phi(R) + \Phi(S)$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

4.7.6 Units for relation composition and matrix multiplication

Lemma 4.7.6 shows that the Φ transformation transforms the unit for composition of relations to the unit for matrix multiplication.

$$
\begin{array}{ll}
 & \Phi(E_A) \\
= & \{\text{Definition of } E_A \text{ — Section 3.3.2 (Equation 3.1, Page 44)}\} \\
 & \Phi(\langle a : a \in A : (a,a)\rangle) \\
= & \left\{\begin{array}{l}\text{A set can be expressed as the union of} \\ \text{the singleton sets of its elements}\end{array}\right\} \\
 & \Phi(\langle \bigcup a : a \in A : \{(a,a)\}\rangle) \\
= & \{\text{Definition of } E_A\} \\
 & \Phi(\langle \bigcup a : a \in A : E_{\{a\}}\rangle) \\
= & \{\text{Lemma 4.7.5 repeated}\} \\
 & \langle \sum a : a \in A : \Phi(E_{\{a\}})\rangle \\
= & \{\Phi.E_{\{a\}} = \mathbb{I}_{\{\varepsilon.a\}}\} \\
 & \langle \sum a : a \in A : \mathbb{I}_{\{\varepsilon.a\}}\rangle \\
= & \left\{\begin{array}{l}\text{B} = \langle\, a \mid a \in A \mid \varepsilon.a \,\rangle, \\ \text{Definition of } \mathbb{I}_B \text{ — Section 4.3.5}\end{array}\right\} \\
 & \mathbb{I}_B
\end{array}
$$

Lemma 4.7.6: $\Phi(E_A) = \mathbb{I}_B$ where $B = \langle a : a \in A : \varepsilon.a\rangle$

4.7.7 Transforming from a matrix to a relation

Through the application of similar reasoning it can be established that the following duals of these correspondences are also valid:

$$
\begin{array}{lcll}
\Psi.(M_0 \times M_1) & = & \Psi.M_0 \circ \Psi.M_1 & \\
\Psi.(M_0 + M_1) & = & \Psi.M_0 \cup \Psi.M_1 & \\
\Psi.(M^k) & = & (\Psi.M)^k & \\
\Psi.\mathbb{I}_n & = & E_U & \\
\Psi.\mathbb{I}_{\{k\}} & = & E_{\{a\}} & \text{where } \varepsilon^{-1}.k = a \\
\Psi.\mathbb{I}_B & = & E_A & \text{where } A = \langle b : b \in B : \varepsilon^{-1}.b\rangle
\end{array}
$$

The transformations namely $\Phi : \mathcal{P}(U \times U) \mapsto \mathbb{B}[n,n]$ and $\Psi : \mathbb{B}[n,n] \mapsto \mathcal{P}(U \times U)$ introduced in this section provide a 1-to-1-correspondence between relations on $U \times U$ and square Boolean matrices. If R is a binary relation, $\Phi(R)$ refers to the square Boolean matrix representing R. $\Phi(R)$ is called the adjacency matrix of relation R. If M is a square Boolean matrix, $\Psi(M)$ refers to the binary relation represented by M. M is called the adjacency matrix of the relation $\Psi(M)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The operations on relations correspond with well defined operations on square Boolean matrices. This correspondence forms the basis for showing the correctness of the TC algorithms described in Chapter 12.

4.8 Relative strength of Boolean matrices

For matrices $P \in \mathbb{B}[n,n]$ and $Q \in \mathbb{B}[n,n]$, the notation $P \sqsubseteq Q$ is used to denote that $\Psi(P) \subseteq \Psi(Q)$. $P \sqsubseteq Q$ thus means that P is the adjacency matrix of a sub-relation of $\Psi(Q)$. P is said to be *stronger than* Q. $P \sqsubseteq Q$ is formally defined as follows:

$$P \sqsubseteq Q \equiv \langle\, \forall i,j \mid i \in \mathbb{N}_m \wedge j \in \mathbb{N}_n \mid p_{ij} \Rightarrow q_{ij} \,\rangle.$$

This definition is needed to apply the mathematical reasoning about relations to Boolean matrices. In particular the reasoning in Section 11.3 that shows the correctness of Algorithm 11.3.1 (Coat) is transformed in Section 12.2 in arguments to show that Algorithm 12.2.2 (MatrixCoat) is correct.

Note that definition of the relative strength of two Boolean matrices is only defined if the matrices have the same dimensions. If $P \sqsubseteq Q$ then P and Q have the same dimensions but P is sparser than Q is such a way that $p_{ij} \Rightarrow q_{ij}$

The difference between $\subseteq$ and $\sqsubseteq$ when comparing matrices is illustrated with examples using the Boolean matrices defined as follows:

$$\begin{aligned}
S &= \langle\!\langle\, i,j \mid i \in \mathbb{N}_3, j \in \mathbb{N}_3, i = j \mid s_{ij} \,\rangle\!\rangle \\
P &= \langle\!\langle\, i,j \mid i \in \mathbb{N}_4, j \in \mathbb{N}_4, i = j \mid p_{ij} \,\rangle\!\rangle \\
Q &= \langle\!\langle\, i,j \mid i \in \mathbb{N}_4, j \in \mathbb{N}_4, i \geq j \mid q_{ij} \,\rangle\!\rangle \\
T &= \langle\!\langle\, i,j \mid i \in \mathbb{N}_4, j \in \mathbb{N}_4, i + j = 4 \mid t_{ij} \,\rangle\!\rangle
\end{aligned}$$

$S \subseteq P \sqsubseteq Q$: $S \subseteq P$ because S can be formed by removing the last column and last row from P. $P \sqsubseteq Q$ because each entry that is 1 in P is also 1 in Q.

$$\underset{S}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \subseteq \underset{P}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}} \sqsubseteq \underset{Q}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}$$

$S \not\subseteq T \not\sqsubseteq Q$: $S \not\subseteq T$ because there is no way in which rows and/or columns can be removed from T to form S. $T \not\sqsubseteq Q$ because there are entries that are 1 in T but 0 in Q.

$$\underset{S}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \not\subseteq \underset{T}{\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}} \not\sqsubseteq \underset{Q}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}}$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

It is straightforward to verify that if $k \in A$ and $A \subseteq \mathbb{N}_n$ that the following holds for the special Boolean matrices $\mathbb{I}_n, \mathbb{I}_A$ and $\mathbb{I}_{\{k\}}$ defined in Section 4.3.5:

$\mathbb{I}_{\{k\}} \sqsubseteq \mathbb{I}_A \sqsubseteq \mathbb{I}_n$.

4.9 Summary

This chapter gives basic definitions of vectors and matrices to clarify terminology and the notational conventions used in this thesis. The symbols used to refer to the special square Boolean matrices are explained in Section 4.3.5. These symbols are newly defined for use in this thesis.

The concept of a submatrix defined in Section 4.4.1 as well as the operations on such submatrices defined in Section 4.4.2 are new concepts. They were defined to enable concise reference to specific portions of a matrix. The concept of a submatrix is an extension of the concept of a subset, with a further specification that the elements need to be ordered. This definition of a submatrix is general and can be applied to matrices of any arity. It allows for the specification of arbitrary ordering of elements in submatrices.

The proposed notation, however, provides only for defining submatrices of two-dimensional matrices and orderings that corresponds closely with the ordering of the supermatrix of the defined submatrix. This limitation is not a problem, as the algorithms discussed in this thesis do not require definitions of submatrices beyond these. This notation is applied to define special square Boolean matrices denoted by $\mathbb{I}_A$ and $\mathbb{I}_{\{k\}}$ in Section 4.3.5.

In Section 4.5, the concept of a partition of a set defined by Erdös and Rado [86] is extended to a new concept, namely to a partition of a matrix. In addition, a notation is introduced to state that a given sequence is a partition of a specific matrix. It is needed for specifying the algorithms in Chapters 16 and 17 and for providing arguments to show their correctness.

Section 4.7 includes an original mathematical development to establish a transformation from a relation R to a corresponding Boolean matrix, called the adjacency matrix of the relation R, denoted by $\Phi(R)$. Section 4.7.4 establishes the opposite transformation, namely a transformation from a square Boolean matrix M to a binary relation, called the relation represented by M, denoted by $\Psi(M)$. These transformations provide a 1-to-1-correspondence between relations on $U \times U$ and the adjacency matrices of these relations. The transformations form the basis for showing the correctness of the TC algorithms described in Chapter 12.

Lastly an ordering of Boolean matrices denoted by $\sqsubseteq$, called relative strength, is introduced. This ordering of Boolean matrices corresponds with an ordering of relations defined in terms of subsets. The ordering of relations is used in mathematical reasoning about relations involving subrelations. In particular this corresponding ordering of Boolean matrices is applied in Section 11.3 to show that the transformation from Algorithm 11.3.1 (Coat) into Algorithm 12.2.2 (MatrixCoat) preserves the correctness of the algorithm.

The next chapter introduces the concept of a path in a relation. Operations that can be applied to paths and their properties are discussed.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 5

Paths

The ground concepts about relations and their properties needed in proofs later in this thesis are established in Section 3.4 while Chapter 4 dealt with matrices as a representation model for relations. In Chapter 4 a transformation between relations and matrices is established. This chapter builds on these concepts to define the concept of a path in a relation and to investigate the properties of operations that can be applied to paths.

Often authors define a path in a relation $R \subset U \times U$ in terms of a subset of R that complies with some restrictions. In contrast to this view, it was decided to define a path P in R in terms of a vector in U. The idea of representing a path in terms of a vector and then of providing a process for translating the vector representation to a representation in terms of pairs, is somewhat unconventional. This idea, however, resonates well with the intuitive meaning of a path, being a sequence of connected edges in a relation which is uniquely identified by sequence of elements in the domain of the relation. A path, P, is thus specified by listing elements of U in a specific order. This notation specifies a subset of R where every pair of successive entries in P represents an element of R and the sequence of edges formed in this way is a path.

This chapter features definitions, facts that arise from these definitions, an algorithm for the construction of an acyclic path as well as a number of original lemmata. These are needed to support arguments in the discussions related to the correctness of algorithms or algorithmic complexity of algorithms that are covered in Part III.

5.1 Relational equivalent and length of a vector

For vector $P = \langle\!\langle\ i \mid i \in \mathbb{N}_n \wedge (n > 0) \mid p_i\ \rangle\!\rangle$, define the relational equivalent of P, denoted by $\mathcal{R}(P)$, by

$$\mathcal{R}(P) = \langle\ i \mid i \in \mathbb{N}_{n-1} \mid (p_i, p_{i+1})\ \rangle$$

Note that $\mathcal{R}(P)$ is not defined if P is a sequence containing only one element. Thus when $P = \langle\!\langle\ i \mid i \in \mathbb{N}_0 \mid p_i\ \rangle\!\rangle = (p_0)$ then $\mathcal{R}(P)$ is undefined.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following hold:

- $\mathcal{R}(P)$ is a binary relation on $\mathbb{N}_n \times \mathbb{N}_n$ where $(a,b) \in \mathcal{R}(P)$ means that a is directly succeeded by b in the vector P.
- The length of a vector P, denoted by $\ell(P)$, is defined as $|\mathcal{R}(P)|$
- P is a *sequence* that may contain duplicate entries, while $\mathcal{R}(P)$ is a *set* that has unique elements. The third example below illustrates this.

Examples:

$P = (0,1,3,6)$
$\Rightarrow \mathcal{R}(P) = \{(0,1),(1,3),(3,6)\}$
$\Rightarrow \ell(P) = 3$

$P = (0,1,2,3,4,2,5)$
$\Rightarrow \mathcal{R}(P) = \{(0,1),(1,2),(2,3),(3,4),(4,2),(2,5)\}$
$\Rightarrow \ell(P) = 6$

$P = (0,1,2,3,1,2,4)$
$\Rightarrow \mathcal{R}(P) = \{(0,1),(1,2),(2,3),(3,1),(2,4)\}$
$\Rightarrow \ell(P) = 5$

If $P \in U[n]$ has no duplicate elements $\ell(P) = |\mathcal{R}(P)| = n-1$. However, if $P \in U[n]$ has duplicates, $\mathcal{R}(P)$ may have less than $n-1$ elements. Thus, it can be concluded that

$$P \in U[n] \Rightarrow \ell(P) \leq n-1 \quad (5.1)$$

5.2 Definitions

5.2.1 Defining a path

Consider $R \subseteq U \times U$ and a vector $P = (p_0, p_1, \ldots p_{m-1}) \in U[m]$ with $m > 0$. P is a *path* from p_0 to p_{m-1} in R if and only if the relational equivalent of P is a subset of R; i.e.

$$P \text{ is a } \textit{path} \text{ from } p_0 \text{ to } p_{m-1} \text{ in } R \iff \mathcal{R}(P) \subseteq R.$$

The requirement that $m > 0$ ensures that a path P has at least two entries, otherwise $\mathcal{R}(P)$ may be undefined. Note that the concept of a zero length path is undefined. Also note that each entry p_i in the vector P has to be an element of the domain (and co-domain) of the relation R in which the path P is defined.

5.2.2 Super-paths and sub-paths

Given $P = \langle\!\langle\ i \mid i \in \mathbb{N}_n \mid p_i\ \rangle\!\rangle$ is a path in R then $Q = \langle\!\langle\ j \mid j \in \mathbb{N}_{n-1} \mid q_j\ \rangle\!\rangle$ is a *sub-path* of P if Q is a path in $\mathcal{R}(P)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following hold:

- $Q \Subset P$ may be used as shorthand to denote that Q is a sub-path of P
- If $Q \Subset P$ then P is a *super-path* of Q, denoted $Q \Supset P$.
- A sub-path should have at least two entries and strictly fewer entries than its super-path. A path of length 1 cannot have sub-paths.

5.2.3 Sub-paths sharing end-nodes with their super-paths

The four infix string operators $\upharpoonleft, \upharpoonright, \downharpoonleft, \downharpoonright$ defined by Watson [253] to apply to strings, are adapted here to apply to paths. Assume $P \in U[n]$ i.e. $P = \langle\!\langle\, i \mid i \in \mathbb{N}_n \mid p_i \,\rangle\!\rangle$. Further assume $k \in \mathbb{N}_n$. These operators in $U[n] \times \mathbb{N}_n \to U[n]$ are defined as follows:

- $P \upharpoonleft k = \langle\!\langle\, i \mid i \in \mathbb{N}_k \mid p_i \,\rangle\!\rangle$ i.e. the leftmost k entries of P.
- $P \upharpoonright k = \langle\!\langle\, i \mid (i \in \mathbb{N}_n) \wedge (i > (n-k)) \mid p_i \,\rangle\!\rangle$ i.e.the rightmost k entries of P.
- $P \downharpoonleft k = \langle\!\langle\, i \mid (i \in \mathbb{N}_n) \wedge (i > k) \mid p_i \,\rangle\!\rangle$ i.e. the rightmost $(n-k)$ entries of P
- $P \downharpoonright k = \langle\!\langle\, i \mid i \in \mathbb{N}_{n-k} \mid p_i \,\rangle\!\rangle$ i.e. the leftmost $(n-k)$ entries of P.

The four operators are pronounced 'left take', 'right take', 'left drop' and 'right drop' respectively. It provides a notation to specify a sub-path of a given path that either has the same starting node or the same ending node as the given path.

5.3 Concatenation

5.3.1 Definition

The $\|$ operator is defined for vectors. Given the vectors $P \in U[n]$ and $Q \in U[m]$, the concatenation $T = P \| Q$ is defined as the vector $T \in U[n+m-1]$ where

$$t_i = \begin{cases} p_i & \text{if } i < n; \\ q_{i-n} & \text{if } i \geq n. \end{cases}$$

Concatenation is not commutative.

Given a relation $R \subseteq U \times U$ and two paths P and Q where $P \in U[n]$ in R from p_0 to p_{n-1} and $Q \in U[m]$ in R from q_0 to q_{m-1}. In general $(P \| Q)$ is not a path in R. Only if it is known that $(p_{n-1}, q_0) \in R$ it can be said that $(P \| Q)$ is a path in R from p_0 to q_{m-1} . Since it cannot be generally known if $(p_{n-1}, q_0) \in R$, the truth of $\mathcal{R}(P \| Q) \subseteq R$ is usually unknown.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.3.2 Adjacent paths

Two paths P and Q in a relation R are called *adjacent* if $p_{n-1} = q_0$. i.e. the one path starts where the other path stops.

It can be shown that if P and Q are adjacent, $(P \downharpoonleft 1) \parallel Q = P \parallel (Q \downharpoonright 1)$ and that this vector is a path in R from p_0 to q_{m-1}.

Lemma 5.3.2 shows that $\mathcal{R}((P \downharpoonleft 1) \parallel Q) \subseteq R$. It can thus be concluded that $(P \downharpoonleft 1) \parallel Q$ is a path in R from p_0 to q_{m-1}.

Let $T = (P \downharpoonleft 1) \parallel Q$.
It is shown for all possible values of $i \in \mathbb{N}_n$ that $(t_i, t_{i+1}) \in R$

Case $0 \leq$ i $<$ n - 2 :

$(t_i, t_{i+1}) \in R$

$=$ { Definition of t_i in $(P \downharpoonleft 1) \parallel Q$ }

$(p_i, p_{i+1}) \in R$

$=$ { $(P \downharpoonleft 1)$ is a path in R }

true

Case i = n - 1 :

$(t_i, t_{i+1}) \in R$

$=$ { Definition of t_i in $(P \downharpoonleft 1) \parallel Q$ }

$(p_{n-2}, q_0) \in R$

$=$ { $p_{n-1} = q_0$ }

$(p_{n-2}, p_{n-1}) \in R$

$=$ { P is a path in R }

true

Case n - 1 $<$ i $<$ m+n-2 :

$(t_i, t_{i+1}) \in R$

$=$ { Definition of t_i in $(P \downharpoonleft 1) \parallel Q$ }

$(q_{i-(n-1)}, q_{(i+1)-(n-1)}) \in R$

$=$ { Q is a path in R }

true

Lemma 5.3.2: $\mathcal{R}(P \downharpoonleft 1) \parallel Q) \subseteq R$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.4 Cyclic paths and loops

5.4.1 Definition

Given P is a path from p_0 to p_{m-1} in R, then:

P is *cyclic* $\iff \langle\, \exists i,j \mid i \neq j \mid p_i = p_j \,\rangle$.

P is *acyclic* $\iff \langle\, \forall i,j \mid i \neq j \mid p_i \neq p_j \,\rangle$.

This means that the path P is acyclic if the path do not cross, and is cyclic if it crosses anywhere, even multiple times.

P is a *loop* $\iff p_0 = p_{m-1}$.

No assumptions are made about whether the sub-paths $P \downharpoonright 1$ and $P \downharpoonleft 1$ of a loop P are cyclic or acyclic. That is, if we disconnect the start or the end, there may or may not be cycles in the disconnected sub-path.

5.4.2 Constructing an acyclic sub-path of a cyclic path

Given a path $Q \in U[m]$ from q_0 to q_{m-1} in R with $q_0 \neq q_{m-1}$, that is cyclic, then it is possible to construct an acyclic path from q_0 to q_{m-1} in R. This is done by creating a path $P \Subset Q$ where P is a path that skips sub-paths of Q that are loops. Algorithm 5.4.2 (AcyclicPath) shows this.

When given a cyclic path between two distinct elements in a relation R, this algorithm can be applied to construct an acyclic path between these elements in R. Since the algorithm constructed P by removing possible sub-paths from Q it follows that $\mathcal{R}(P) \subset \mathcal{R}(Q)$ and consequently that

$$\ell(P) < \ell(Q) \tag{5.2}$$

The following examples are program traces to illustrate the application of this algorithm to construct acyclic paths that are sub-paths of given cyclic paths. Each sub-path that is constructed has the same starting element and ending element as its super-path from which it was constructed.

Example 1:

$P = (3, 5, 2, 2, 3, 0, 5, 7)$

{The call to **indexOfDuplicate** returns $< 2, 3 >$}

$P = (3, 5, 2, 3, 0, 5, 7)$

{The call to **indexOfDuplicate** returns $< 1, 4 >$}

$P = (3, 5, 7)$

Example 2:

$P = (9, 4, 5, 6, 3, 0, 7, 8, 6)$

{The call to **indexOfDuplicate** returns $< 3, 8 >$}

$P = (9, 4, 5, 6)$

Example 3:

$P = (3, 9, 1, 7, 5, 2, 9, 7, 0, 8)$

{The call to **indexOfDuplicate** returns $< 1, 6 >$}

$P = (3, 9, 7, 0, 8)$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 5.4.2: Constructing an acyclic path

Pre: $\{\ \exists i, j : 0 \leq i, j < n \wedge i \neq j \wedge p_0 \neq p_{m-1} : p_i = p_j\ \}$
func indexOfDuplicate$(P \in U[n]) : \langle \mathbb{N}_m, \mathbb{N}_m \rangle$
  for $i : 0 \leq i \leq n \rightarrow$
    for $j : i < j \leq n \rightarrow$
      as $(p_i = p_j) \rightarrow$ **return** $\langle i, j \rangle$ **sa**
    rof
  rof
cnuf

const $Q \in U[m]$
var $P \in U[n]$

$P, n := Q, m$
do $(P$ is $cyclic) \rightarrow$
  $\langle i, j \rangle :=$ indexOfDuplicate(P)
  for $k : 1 < k < (n - j) \rightarrow$
    $p_{i+k} := p_{j+k};$
  rof
  $n := i + (n - j);$
od

Note that the loops in **indexOfDuplicate** need not be executed in a deterministic order. If the path is cyclic and contains more than one duplicate, it may return the index of any one of the duplicates. The above example may as well be executed as follows:

Example 4:
$P = (3, 9, 1, 7, 5, 2, 9, 7, 0, 8)$
{The call to **indexOfDuplicate** returns $< 3, 7 >$}
$P = (3, 9, 1, 7, 0, 8)$

No claims other than being acyclic and maintaining the endpoints are made about the constructed path. It might be possible to create more than one such path. Furthermore, the constructed path need not be the shortest path.

5.5 Relation exponentiation and paths

The lemmata discussed in this section shows a correspondence between relation exponentiation and the existence of paths in the relation. They are used in the verification of Algorithm 12.4.1 (Prosser) in Section 12.4.3.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.5.1 Correspondence between elements in R^t and paths in R

Every element of a relation $R \subseteq U \times U$ corresponds with a path $P \in U[2]$ in R. When R is composed with itself the resulting relation is $R \circ R = R^2$. Every $(a,b) \in R^2$ requires adjacent paths $(a,c) \in U[2]$ and $(c,b) \in U[2]$ in R and that the path $(a,c,b) \in U[3]$ in R can be formed. Thus every element in R^2 requires the existence of a path $(a,c,b) \in U[3]$ in R. Lemma 5.5.1 is a generalisation of this observation. It shows that given a relation $R \subseteq U \times U$ it holds that for every element of $R^t, t \geq 1$ there is a path $P \in U[t+1]$ in R.

This Lemma can be proven using induction on t:

Case t = 1 :

$$(a,b) \in R^1$$

$\equiv$ { Definition of R^1, Definition of $\mathcal{R}$ }

$$(a,b) \in R \wedge \mathcal{R}((a,b)) = \{(a,b)\}$$

$\equiv$ { Definition of subset }

$$\mathcal{R}((a,b)) \subseteq R$$

$\equiv$ { $P = (a,b) \in U[2]$ }

$$\langle\ \exists P \mid P \in U[2], p_0 = a, p_1 = b \mid \mathcal{R}(P) \subseteq R\ \rangle$$

Case t = r + 1 (Assume the lemma hold for t = r) :

$$(a,b) \in R^{r+1}$$

$\equiv$ { Definition of exponentiation of a relation }

$$(a,b) \in (R^r \circ R)$$

$\equiv$ { Definition of composition of relations }

$$\langle \exists c : c \in U : (a,c) \in R^r \wedge (c,b) \in R \rangle$$

$\equiv$ { Induction assumption }

$$\langle\ \exists Q, c \mid Q \in U[r+1], q_0 = a, q_r = c \mid \mathcal{R}(Q) \subseteq R \wedge (c,b) \in R\ \rangle$$

$\equiv$ { Apply Lemma 5.3.2 to construct a new path $P = (q_0, q_1, \ldots, q_r, b)$ Thus $\mathcal{R}(P) = \mathcal{R}(Q) \cup \{(q_r, b)\}$ with $q_r = c$ }

$$\langle\ \exists P \mid P \in U[r+2], p_0 = a, p_{r+1} = b \mid \mathcal{R}(P) \subseteq R\ \rangle$$

Lemma 5.5.1: $(a,b) \in R^t \equiv \langle\ \exists P \mid P \in U[t+1], p_0 = a, p_t = b \mid \mathcal{R}(P) \subseteq R\ \rangle$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

5.5.2 Maximum length of an acyclic path in R

Lemma 5.5.2 shows that if $P \in U[n]$ is an acyclic path in $R \subseteq U \times U$, it follows that $\ell(P) < |U|$.

$$
\begin{array}{ll}
 & P \in U[n] \text{ is an acyclic path in } R \subseteq U \times U \\
= & \qquad \{\text{Definition of an acyclic path}\} \\
 & \mathcal{R}(P) \subseteq R \wedge \langle \forall i, j : i \neq j : p_i \neq p_j \rangle \\
\Rightarrow & \qquad \{\text{Definition of } \mathcal{R}(P) \text{ and the entries in } P \text{ are unique}\} \\
 & \langle\, i \mid i \in \mathbb{N}_n \mid p_i \,\rangle \subseteq U \\
\Rightarrow & \qquad \{A \subseteq B \Rightarrow |A| \leq |B|\} \\
 & n \leq |U| \\
= & \qquad \{n - 1 < n\} \\
 & n - 1 < |U| \\
\Rightarrow & \qquad \{\text{ Equation 5.1}\} \\
 & \ell(P) < |U|
\end{array}
$$

Lemma 5.5.2: $P \in U[n]$ is an acyclic path in $R \subseteq U \times U \Rightarrow \ell(P) < |U|$

5.6 Summary

The concept of a path is prominent in many of the arguments that show the correctness of TC Algorithms. For this reason, mathematical notation, properties of paths and operations on paths are provided in this chapter.

The definition of a path and features of paths, such as being cyclic or acyclic, are not standardised in the literature. In this chapter these concepts are defined in an unique way. Instead of defining a path in terms of edges in a graph, it is defined in terms of elements in the domain of a relation. A transformation between these methods of defining a path is given to show that the meaning is preserved. This alternative way of defining a path is consistent with other concepts defined in this thesis. In particular, notational elements are used that cohere with the notation used in the rest of the thesis. The definition of a path in context of a given relation gave rise to elegant definitions of sub-paths and super-paths. To my knowledge, these definitions are novel. They, however, preserve the meaning of these concepts as defined by other mathematicians in different contexts.

The four infix operations defined in Section 5.2.3 are adapted from Watson's [253], where they were first defined. However, in Watson's [253] thesis and in publications related to his thesis they are treated as string operators. Here they are used as path operators. Thus, although the same idea as Watson is used, it is applied in a different and novel context. These operations were introduced to have

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

a concise notation to refer to a sub-path that is at one of the ends of its super-path in arguments about the concatenation of paths in Section 5.3. They were also used in arguments to show the correctness of TC Algorithms in Part III of this thesis.

Algorithm 5.4.2 (AcyclicPath) was written to serve as a *proof by construction* of the fact that for every cyclic path in a relation connecting two nodes, one can find a shorter acyclic path connecting the same nodes. This fact is used in Section 13.2.2, when verifying the correctness of Algorithm 13.2.1 (Martynyuk).

The chapter includes three original lemmata that were proven in order to support correctness arguments in this thesis. Lemma 5.3.2 is used in Lemma 5.5.1, Lemmata 5.5.1 and 5.5.2 are used in Section 12.4.3 to verify the correctness of Algorithm 12.4.1 (Prosser) and Lemma 5.5.2 is used in Section 13.2.2 to verify the correctness of Algorithm 13.2.1 (Martynyuk).

The next chapter uses the mathematical preliminaries that are given in this chapter and the chapters preceding it to derive the mathematical formulae for computing the transitive closure of a relation.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 6

Transitive Closure

In Section 1.7 the concept of a relation was introduced and the intuitive meaning of the transitive closure of a relation was explained. In this chapter the transitive closure is formally defined. The mathematical reasoning to show the equivalence of two different transitive closure definitions is presented. These lead to two methods to construct the transitive closure of a given binary relation.

Section 6.5 derives formulae to compute the transitive closure of a relation. Although the ideas are not new, the arguments provided and the Lemmata in this section are original.

6.1 Formal Specification

6.1.1 Definition

The transitive closure of a relation R is denoted by R^+. The transitive closure of a binary relation $R \subseteq U \times U$ is the minimal transitive relation $R^+ \subseteq U \times U$ that contains R. i.e.

1. $R \subseteq R^+$.
   This can be expressed as $(a, b) \in R \implies (a, b) \in R^+$
2. R^+ is transitive.
   This can be expressed as $(a, b) \in R^+ \wedge (b, c) \in R^+ \implies (a, c) \in R^+$
3. R^+ is minimal.
   This can be expressed as $(S \textit{ is transitive}) \wedge R \subseteq S \implies R^+ \subseteq S$

6.1.2 Precondition

The precondition for an algorithm that calculates the transitive closure R^+ of a homogeneous binary relation R on universe U is simply $R \subseteq U \times U$. The *coat algorithm* discussed in Section 11.3 assumes the weaker precondition that R is finite while the further requirement that U is finite is needed for algorithms using adjacency matrices to represent the binary relations they manipulate.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.1.3 Postcondition

Based on the definition of transitive closure, the postcondition for an algorithm that calculates the transitive closure R^+ of a homogeneous binary relation R on universe U is

$$
\begin{aligned}
&((a,b) \in R \implies (a,b) \in R^+) \\
&\wedge((a,b) \in R^+ \wedge (b,c) \in R^+ \implies (a,c) \in R^+) \\
&\wedge (S \text{ is transitive} \wedge R \subseteq S \implies R^+ \subseteq S)
\end{aligned}
$$

6.2 Complexity

The simplest algorithm to calculate transitive closure is to represent the relation as a graph as described in Section 3.5.3 and to perform a breadth-first or depth-first search from each vertex and keep track of all vertices encountered. Doing n such traversals gives a $\Theta(n(n+m))$ algorithm, which degenerates to cubic time if the graph is dense [226]. The complexity of the algorithms discussed in Part III in this thesis range from $\Omega(n^3)$ to $O(n^4)$.

Transitive closure is also known to be a problem in the class NC, implying that it can be solved in poly-log time with a polynomial number of processors. [3].

6.3 Mathematical preliminaries

The discussions of the ways to construct the transitive closure (TC) in Section 6.5 rely on knowledge of a number of theoretical mathematical concepts of relational theory and order theory. These include theories of functions, partial orders and fixed points. Some of these concepts were briefly discussed in previous chapters while some remaining concepts and theories are mentioned in Sections 6.3.2 to 6.3.5. Interested readers are referred to comprehensive mathematical resources such as Weisstein [258] and Russell [211] for detailed discussions on the topics used in the arguments that follow.

6.3.1 Kleene closure

In mathematical logic and computer science, the Kleene closure is a unary operation usually applied to sets of strings or on sets of symbols or characters. This operation was introduced by Stephen Kleene [265] to characterise certain automata, where the Kleene closue of a denoted by a^* means *zero or more* instances of a. If a is a set of strings, then a^* is defined as the smallest superset of a that contains the empty string and is closed under the string concatenation operation. If a is non-empty and contains at least one non-empty string, then a^* is a countably infinite set [33]. In this thesis the Kleene closure of a relation R denoted by R^* is defined to be the smallest superset of R that contains the empty relation and is closed under the relation composition operation defined in Section 3.3.4. The correspondence between Kleene closure and the transitive closure of a relation, denoted by R^+, is a key element in the derivation of algorithms to calculate the transitive closure of a relation.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

6.3.2 Bounds and supremum

Given $A \neq \varnothing$ and a subset $S \subseteq A$ together with a binary relation $R \subseteq A \times A$ that is reflexive and transitive. $a \in A$ is an *upper bound* of S with respect to R, if $s \in S \Rightarrow sRa$. The *lower bound* of S with respect to R is defined dually as $b \in A$ for which $s \in S \Rightarrow bRs$. The *supremum* of S with respect to R is the least upper bound of a set S, denoted $Sup.S$. It is defined as an upper bound of S, which is also a lower bound of the set of upper bounds of S. When it exists (which is not required by this definition), it is unique.

6.3.3 Partially ordered sets

A binary relation R is called a partial order if it is reflexive, antisymmetric and transitive. A partially ordered set (or poset) is a set taken together with a partial order on it. Formally, a partially ordered set is defined as a pair $P = (X, R)$, where X is called the ground set of P and R is the partial order of X. Often authors refer to the ground set of P to being a partially ordered set with respect to R.

A partially ordered set $P = (U, R)$ is called a directed complete partial order (DCPO) if R is a partial order and if each of the subsets of U has a supremum with respect to R. $P = (U, R)$ is called a complete partial order (CPO) if it is a DCPO with U having a least element with respect to R. The least element of a CPO is called the *bottom* of the CPO, denoted by $\bot$.

6.3.4 Continuous functions

A function $f : X \mapsto X$ is *continuous* if given $\varnothing \neq S \subseteq X$ the following holds:
$Sup.(f.S) = f.(Sup.S)$.

6.3.5 Least fixpoint

Sets may be defined in terms of operations, for example; the elements of X in the following definition are defined in terms of the operation f on the elements of U.

$$X = \langle u \mid u \in U \mid f(u) \rangle \tag{6.1}$$

Assume f is a binary homogeneous function, then an element $x \in X$ for which $x = f(x)$ holds in this definition is called a *fixpoint* and the smallest value for which it holds is the *least fixpoint* of the defining operation.

The Knaster-Tarski fixpoint theorem [44] specifies the conditions on the kind of operations that guarantee the existence of such a fixpoint. It states that if the domain of the function that defines the operation is the ground set of a CPO, say $(X, \preccurlyeq)$, and the operation is defined by a continuous function $f : (X, \preccurlyeq) \mapsto (X, \preccurlyeq)$, then a least fixpoint exists and that its explicit expression is $\langle \oplus i \mid i \in \mathbb{N} \mid f^i . \bot \rangle$, where $\oplus$ refers to the supremum that comes with $\preccurlyeq$. It is common practice to denote the least fixpoint of f by μf.

For the pair of relations $R, S \in \mathcal{P}(U \times U), R \cup S \in \mathcal{P}(U \times U)$ is an upper bound of this pair. Furthermore, $\subseteq$ is a partial order and the union of the elements

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

of any subset of $\mathcal{P}(U \times U)$ is a supremum of this subset. Therefore, the set of homogeneous relations $\mathcal{P}(U \times U)$ together with the reflexive, transitive relation $\subseteq$ is a directed set, i.e. $(\mathcal{P}(U \times U), \subseteq)$ is a DCPO. $(\mathcal{P}(U \times U), \subseteq)$ also has a least element, namely the empty relation $\varnothing$. Therefore $(\mathcal{P}(U \times U), \subseteq)$ is a CPO.

Consider the CPO $(\mathcal{P}(U \times U), \subseteq)$. The continuous function on this CPO considered here is the function $g : (\mathcal{P}(U \times U), \subseteq) \mapsto (\mathcal{P}(U \times U), \subseteq))$, defined by the following expression:

$$g.X = S \cup X \circ R \tag{6.2}$$

In this function $X \subseteq U \times U$ and it is assumed that $R \subset U \times U$ and $S \subset U \times U$. The least fixpoint μg of this chosen continuous function g on $(\mathcal{P}(U \times U), \subseteq)$ is a relation known as $S \circ R^*$, where R^* is the reflexive transitive closure of R. Per definition $S \circ R^*$ is the lower bound of the following set with respect to $\subseteq$:

$$\langle X \mid X \subseteq U \times U \mid X = S \cup X \circ R \rangle \tag{6.3}$$

According to the Knaster-Tarski theorem, this lower bound exists and the following is the explicit form to calculate it:

$$S \circ R^* = \langle \bigcup i \mid i \in \mathbb{N} \mid g^i . \perp \rangle \tag{6.4}$$

Since $\perp = \varnothing$, $g^i . \perp$ can be expressed as $g^i . \perp = S \cup \langle \bigcup k \mid 0 \leq k \leq i \mid R^k \rangle$. Thus, the explicit expression to calculate $S \circ R^*$ is given by:

$$S \circ R^* = \langle \bigcup i \mid i \in \mathbb{N} \mid S \cup \langle \bigcup k \mid 0 \leq k \leq i \mid R^k \rangle \rangle \tag{6.5}$$

The above definition of the relation $S \circ R^*$ can now be applied to formulate a series of definitions by considering special instances of this definition where the value of S is known. An example creating a definition is to replace $S = E_U$ in the above definition. Owing to $E_U \circ R^*$ being a fixpoint of the function defined in Equation 6.2 we have $E_U \circ R^* = g.(E_U \circ R^*) = E_U \cup (E_U \circ R^*) \circ R$ i.e.

$$R^* = E_U \cup R^* \circ R. \tag{6.6}$$

It also leads to the derivation of the following properties of R^* and R^+ that are relevant in further discussions:

$$R^+ = R \circ R^* \tag{6.7}$$
$$R^* \circ R = R \circ R^* \tag{6.8}$$
$$R^* = E_U \cup R \circ R^* \tag{6.9}$$
$$R^+ = R \cup R \circ R^+ \tag{6.10}$$
$$(R \circ S)^* \circ R = R \circ (S \circ R)^* \tag{6.11}$$
$$\varnothing^* = E_U \tag{6.12}$$
$$\mathit{point}\ A \text{ on } U \Rightarrow A^* = E_U \tag{6.13}$$
$$R \subseteq R^+ \tag{6.14}$$
$$(R \cup S)^* = R^* \circ (S \circ R^*)^* \tag{6.15}$$
$$R^* \circ R \subseteq R^* \tag{6.16}$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

To establish each of these properties would involve the definition of an appropriate continuous function on $(\mathcal{P}(U \times U), \subseteq)$ and applying the Knaster-Tarski fixpoint theorem to show that the explicit form to calculate the lower bound of the result constitutes the required property. The establishment of Equation 6.6 serves as an example of how this can be done.

There are many continuous functions on $(\mathcal{P}(U \times U), \subseteq)$. In Section 6.5 two such functions that lead to the derivation of formulae to calculate R^+ are considered.

6.4 Lemmata involving Kleene closure

Typically infinite steps are needed to determine the value of an expression that involves Kleene closure. In Sections 6.4.1 and 6.4.2 two Lemmata are presented in which the values of expressions involving Kleene closures are exact, while Section 6.4.3 derives a formula to calculate the value of $h(A \cup u)$ where h is a specific function chosen to derive formula to construct the transitive closure as discussed in Section 6.5.2.

6.4.1 The value of $R \circ (E_\varnothing \circ R)^*$

Lemma 6.4.1 shows that the value of $R \circ (E_\varnothing \circ R)^* = R$. This lemma is used in the derivation of a recursive definition of R^+ in Section 6.5.2.

$$
\begin{array}{ll}
 & R \circ (E_\varnothing \circ R)^* \\
= & \quad \{ E_\varnothing = \varnothing \} \\
 & R \circ (\varnothing \circ R)^* \\
= & \quad \{ \varnothing \circ R = \varnothing \} \\
 & R \circ \varnothing^* \\
= & \quad \{ \text{Property (6.12)} \} \\
 & R \circ E_U \\
= & \quad \{ E_U \text{ is the identity relation on } U \times U \} \\
 & R
\end{array}
$$

Lemma 6.4.1: $R \circ (E_\varnothing \circ R)^* = R$

6.4.2 The value of $R \circ (E_U \circ R)^*$

The extreme case of a similar expression that involves Kleene closure containing the unity relation can also be determined without infinite extension. Lemma 6.4.1 shows that the value of $R \circ (E_U \circ R)^* = R^+$. This lemma is needed for the derivation of a recursive definition of R^+ in Section 6.5.2.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & R \circ (E_U \circ R)^* \\
= & \quad \{ E_U \text{ is the identity relation on } U \times U \} \\
 & R \circ R^* \\
= & \quad \{ \text{Property (6.7)} \} \\
 & R^+
\end{array}
$$

Lemma 6.4.2: $R \circ (E_U \circ R)^* = R^+$

6.4.3 Calculating $h.(A \cup \{u\})$

The construction of the recursive formula to calculate R^+ in Section 6.5.2 require that the value of $h.(A \cup \{u\})$, where h is defined in Expression 6.18. Lemma 6.4.3 shows that the value of $h.(A \cup \{u\})$ can be determined if the value of $h.(A)$ is known.

6.5 Constructing the transitive closure of a relation

One way of constructing R^+ is by starting with the relation R and systematically adding elements to it until a state is reached where the resulting relation is R^+. This method is discussed in Section 6.5.1. Another way of constructing R^+ is by defining a function f that ultimately maps U to R^+. The algorithm starts with the empty set and systematically grows the argument to U. This method is discussed in Section 6.5.2. A third way to calculate R^+ is by starting with an empty relation, the trivial transitive relation, and systematically adding the elements of R to it in such a way that its transitivity is maintained. The process is continued until the resulting relation contains all of R, hence being R^+. This method is beyond the scope of this thesis.

6.5.1 Calculating R^+ with exponentiation of R

The application of the Knaster-Tarski fixpoint theorem applied to the least fixpoint of the function considered in this section leads to a definition of the transitive closure of a relation with an explicit form where R^+ is computed via repeated exponentiation of R.

When substituting S with E_U in Equation 6.5, the expression on the left hand side of the equation namely $E_U \circ R^*$ may be simplified to R^* as E_U is the unit of the composition operator. This special case of the equation yields the following explicit form to calculate R^*:

$$R^* = \langle \bigcup i \mid i \in \mathbb{N} \mid E_U \cup \langle \bigcup k \mid 0 \leq k \leq i \mid R^k \rangle \rangle$$

A special application of the above formula yields a method to calculate $\varnothing^*$:

$$\varnothing^* = \langle \bigcup i \mid i \in \mathbb{N} \mid E_U \cup \langle \bigcup k \mid 0 \leq k \leq i \mid \varnothing^k \rangle \rangle = E_U.$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$h.(A \cup \{u\})$

$=$ { Definition of h }

$R \circ (E_{A \cup \{u\}} \circ R)^*$

$=$ { Property (3.3) }

$R \circ ((E_A \cup E_{\{u\}}) \circ R)^*$

$=$ { Composition distributes from right over union. }

$R \circ ((E_A \circ R) \cup (E_{\{u\}} \circ R))^*$

$=$ { Property (6.15) }

$R \circ ((E_A \circ R)^* \circ ((E_{\{u\}} \circ R) \circ (E_A \circ R)^*)^*)$

$=$ { Composition is associative }

$(R \circ (E_A \circ R)^*) \circ (E_{\{u\}} \circ (R \circ (E_A \circ R)^*))^*$

$=$ { Definition of h (twice) }

$h.A \circ (E_{\{u\}} \circ h.A)^*$

$=$ { Property (6.9) }

$h.A \circ (E_U \cup (E_{\{u\}} \circ h.A) \circ (E_{\{u\}} \circ h.A)^*)$

$=$ { Composition distributes from left over union }

$(h.A \circ E_U) \cup h.A \circ ((E_{\{u\}} \circ h.A) \circ (E_{\{u\}} \circ h.A)^*)$

$=$ { E_U is the identity relation on $U \times U$, composition is associative. }

$h.A \cup h.A \circ E_{\{u\}} \circ h.A \circ (E_{\{u\}} \circ h.A)^*$

$=$ { Lemma (3.4.2) (twice) }

$h.A \cup h.A \circ E_{\{u\}} \circ E_{\{u\}} \circ h.A \circ (E_{\{u\}} \circ E_{\{u\}} \circ h.A)^*$

$=$ { Property (6.11) (twice) }

$h.A \cup h.A \circ E_{\{u\}} \circ (E_{\{u\}} \circ h.A \circ E_{\{u\}})^* \circ E_{\{u\}} \circ h.A$

$=$ {
$E_{\{u\}} \not\subseteq h.A \implies (E_{\{u\}} \circ h.A \circ E_{\{u\}}) = \varnothing$ (Lemma 3.3.5)
$E_{\{u\}} \subseteq h.A \implies (E_{\{u\}} \circ h.A \circ E_{\{u\}}) = E_{\{u\}}$ (Lemma 3.3.6)
Apply Property 6.12 if $E_{\{u\}} \not\subseteq h.A$
and Property 6.13 otherwise.
}

$h.A \cup h.A \circ E_{\{u\}} \circ E_U \circ E_{\{u\}} \circ h.A$

$=$ { E_U is the identity relation on $U \times U$ }

$h.A \cup h.A \circ E_{\{u\}} \circ E_{\{u\}} \circ h.A$

Lemma 6.4.3: $h.(A \cup \{u\}) = h.A \cup h.A \circ E_{\{u\}} \circ E_{\{u\}} \circ h.A$

© University of Pretoria

Lemma 6.5.1 shows the derivation of the expression $R^+ = R \cup R \circ R^+$ from the function defined in Expression 6.2 if S is substituted with R.

$$
\begin{array}{ll}
 & g.X = S \cup X \circ R \\
\Rightarrow & \qquad \{\text{Substitute } S \text{ with } R\} \\
 & g.X = R \cup X \circ R \\
\Rightarrow & \qquad \{\text{Substitute } X \text{ with } R \circ R^*\} \\
 & g.(R \circ R^*) = R \cup (R \circ R^*) \circ R \\
\Rightarrow & \qquad \{\ R \circ R^* \text{ is a fixpoint of this function}\} \\
 & R \circ R^* = R \cup (R \circ R^*) \circ R \\
= & \qquad \{\ \circ \text{ is associative}\} \\
 & R \circ R^* = R \cup R \circ (R^* \circ R) \\
= & \qquad \{\text{Property 6.8}\} \\
 & R \circ R^* = R \cup R \circ (R \circ R^*) \\
= & \qquad \{\text{Property 6.7 twice}\} \\
 & R^+ = R \cup R \circ R^+
\end{array}
$$

Lemma 6.5.1: Derivation of $R^+ = R \cup R \circ R^+$

The following explicit form to calculate R^+ can thus be derived from the explicit form to calculate $S \circ R^*$ in Equation 6.5 by replacing S with R and applying the fact that $R^+ = R \circ R^*$.

$$R^+ = \langle \bigcup i \mid i \in \mathbb{N} \mid R \cup \langle \bigcup k \mid 0 \leq k \leq i \mid R^k \rangle \rangle \tag{6.17}$$

Characteristic of the explicit form to calculate R^+ arrived at here, is the fact that μg is computed via repeated exponentiation of R. When applying this definition the transitive closure of the relation $R \in U \times U$ is essentially computed through exponentiation of R. The calculation starts with R and through repeated composition constructs the limit R^+. The application of this function to derive an algorithm to calculate R^+ is discussed in Section 11.3.

6.5.2 A recursive formula to calculate R^+

Define a continuous function $h : (\mathcal{P}(U \times U), \subseteq) \mapsto (\mathcal{P}(U \times U), \subseteq))$ on the complete partial order (CPO) $(\mathcal{P}(U \times U), \subseteq)$ as follows:

$$h.A = R \circ (E_A \circ R)^* \tag{6.18}$$

Assume $R \subset U \times U$ and $A \subseteq U$ in the above expression that defines h.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

In Sections 6.4.1 and 6.4.2 two extreme properties of this continuous function were established. Lemma 6.4.1 shows that $R \circ (E_{\varnothing} \circ R)^* = R$ and that therefore $h.\varnothing = R$; while Lemma 6.4.2 shows that $R \circ (E_U \circ R)^* = R^+$ and therefore that $h.U = R^+$. In Section 6.4.3 Lemma 6.4.3 expresses $h.(A \cup \{u\})$ in terms of $h.A$ and $E_{\{u\}}$ for $u \in U$ and $u \notin A$. To summarise:

$$\begin{array}{lcl} h.\varnothing & = & R \\ h.U & = & R^+ \\ h.(A \cup \{u\}) & = & h.A \cup h.A \circ E_{\{u\}} \circ E_{\{u\}} \circ h.A \end{array}$$

Ultimately this function has the right properties to support the following recursive definition of R^+ that can be applied to construct R^+.

$$R^+ = R \circ (E_U \circ R)^* \tag{6.19}$$

The application of this function to achieve this is discussed in Section 11.4.

6.6 Summary

This chapter provides the mathematical details about transitive closures of binary relations. The formal specification of the transitive closure problem is given and the complexity of its solution is discussed.

In contrast with most texts that specify the transitive closure of a relation in terms of one of the two formulae to calculate transitive closure that are given here in Section 6.5, the formal specification of transitive closure is given here in terms of the intuitive meaning of transitive closure as was explained in Section 1.7 in Chapter 1, i.e. only in terms of first principles.

The mathematical preliminaries given in Section 6.3 are advanced topics given for the sake of clarifying terminology and establishing results that are needed. In particular in Section 6.3.5 the Knaster-Tarski fixpoint theorem is introduced. The application of the Knaster-Tarski fixpoint theorem simplifies the derivation of properties of Kleene closures. Some basic properties of R^+ and R^* that can be derived through the application of this theory are listed. These are used in correctness arguments in Section 6.4 and later in the thesis.

In Section 6.4 lemmata 6.4.1, 6.4.2 and 6.4.3 are proposed. I provide their proofs using results derived from the Knaster-Tarski fixpoint theorem. These lemmata are used in Section 6.5 to derive two alternative formulae (6.17 and 6.19) for computing the transitive closure of a relation. These formulae are commonly regarded as (recursive) definitions for the transitive closure of a relation, i.e. it is assumed that each of these formulae indeed derive from $R \subseteq U \times U$ the minimal transitive relation $R^+ \subseteq U \times U$. The derivation of formulae 6.17 and 6.19 via the lemmata of Section 6.5 is, to the best of my knowledge, the first formal proof of their validity.

The establishment of these formulae serves as the mathematical proof of the correctness of the algorithms presented in Chapter 11. The rest of the algorithms discussed in Part III of this thesis are derived by applying correctness preserving transformations of these algorithms. The correctness of all the algorithms in this thesis are therefore based on the foundation established in this chapter.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part II

Representing Algorithmic Information

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Representing Algorithmic Information Overview

The aim of this part is to specify metadata for describing algorithms adequately and precisely. The specification is a description of data elements, called attributes, that are relevant to algorithms and their relations with one another.

Chapter 7 gives an overview of research that has been conducted with respect to gathering and capturing information about algorithms. It includes a discussion of literature that deals with processes to gather algorithmic information. Various representation models that were previously applied to capture and manipulate this kind of information are analysed. The chapter concludes with an argument to justify the decision to present the information about algorithms in this thesis as a Topic Map.

Chapter 8 is a discussion of selected collections of algorithmic information. Sections in Chapter 8 highlight aspects of the collections that informed the specification of the metadata discussed in Chapter 9.

The information that results from the reviews in Chapters 7 and 8 is used to specify metadata for algorithms. This is provided in Chapter 9. Based on this metadata, a Topic Map of algorithms is concurrently defined.

Finally a process to gather data about algorithms and capture it in the proposed Topic Map is described in Chapter 10.

Chapter 10 outlines a process for generically gathering and capturing algorithmic attributes. Such a process inevitably generates information about the algorithm under consideration. The approach is illustrated in Part III by extending the Topic Map that had been defined in Chapter 9.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 7

Representation methods and models

7.1 Introduction

This chapter provides the background that supports the specification of metadata for the representation of algorithmic information described in Chapter 9. It also contributes to the development of the process model for gathering algorithmic information adopted in this thesis and described in Chapter 10. It extends the work of other researchers who have described applicable structures and processes.

The topic of knowledge representation is addressed in different ways by a spectrum of disciplines ranging from Information Architecture (IA) to Artificial Intelligence (AI) [199]. IA is a discipline dealing with the problem of finding information in information sources such as document collections. It is about how to organise these information resources so that users can actually find what they are looking for [97]. In AI on the other hand, knowledge representation schemata are used to support programming tools and techniques with the aim to automate the discovery, generation and use of knowledge [227].

This chapter gives a high level unified overview of knowledge creation, retrieval and representation incorporating fundamental ideas from different disciplines. Knowledge organisation necessitates the construction of hierarchical relations between entities through the use of abstraction and its converses as well as techniques to combine multiple hierarchies. To this end the techniques of abstraction, refinement , enrichment and facet analysis are briefly discussed in Section 7.2. In Section 7.3 the structure of a number of schemata for knowledge creation and organisation are investigated.

The aim is to make an informed decision about what methods to use to gather information about algorithms, what schema to use to represent the information and what technique to use to specify the data about algorithms in this thesis. The decisions to allow a combination of of techniques to identify relations between algorithms, to use a thesaurus schema and to use a topic map as representation standard for representing information in this thesis is justified as a concluding subsection in the respective sections of this chapter.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.2 Techniques to order programs in a hierarchy

7.2.1 Abstraction

Jonkers [130] described a method to use abstraction when classifying algorithmic problems and their solutions. He proposed the systematic ordering of problems and their solutions at various levels of abstraction through a process he dubbed *systematic generalisation*. When applying Jonker's method to create a taxonomy of algorithms, relations between algorithms that use different algorithmic techniques to solve the same problem are established by identifying the fundamental nature of the operational steps taken to solve the problem and by taking the essential similarities and differences of these steps into account.

The following authors are among those who have used abstraction to classify algorithms solving specified problems: Darlington [70], Broy [43] and Merritt [167] to classify sorting algorithms; Marcelis [159] to classify attribute evaluation algorithms; Barla-Szabo *et al.* [20] to classify graph representations; Watson [253], Bosman [36], van de Rijdt [243], Cleophas *et al.* [59] to classify various pattern matching algorithms; Kouwenberg [149] to classify Lempel-Ziv Compression Algorithms; Ketcha Ngassam *et al.* [136] to classify DFA-based string processors; Schnorr [217] to classify polynomial time lattice basis reduction algorithms; Cleophas [60] to classify tree acceptance algorithms; and Watson [254] to classify algorithms that construct minimal acyclic finite automata. These authors mainly focused on the operational steps of an algorithm itself and to some extent take into account the restrictions placed on the data on which the algorithm operates.

When building an abstraction hierarchy of algorithms, one usually starts by comparing concrete algorithms. Abstractions are sought that describe common features of the algorithms under consideration.

7.2.2 Refinement

The converse of abstraction in this context is called refinement. Morgan [174] formally specifies refinement of programs in steps that transform an abstract program to an executable program. An abstract program is defined in terms of the specification of pre- and postconditions. He defines a relation called refinement, denoted by $\sqsubseteq$, between programs in terms of actions to strengthen the postcondition or weakening the precondition.

According to Watson [253] the construction of a taxonomy using abstraction, as defined by Jonkers [130], to classify algorithms can be done top-down by applying refinement as defined by Morgan [174]. When applying this method, one places a naïve algorithm, whose correctness can easily be shown, at the root. Each algorithm that is then added to the taxonomy has to be specified in terms of a refined descendant of an algorithm already in the taxonomy. These refinements may be related to algorithmic techniques or data restrictions. If it is proven that the transformations preserve correctness, the correctness of all algorithms in the taxonomy follows from the correctness of the starting point and correctness preserving transformations [60, 253].

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.2.3 Enrichment

Kourie [148] formalised the concept of abstraction of entities in terms of their properties. He defined abstraction in terms of removal of properties and considers the addition of a property to be either a refinement or an enrichment. In this context the concept of specialisation, the opposite of abstraction, encapsulates refinement as well as enrichment. These ideas are described in the context of object oriented programs in [147]. Despite being formulated in terms of classes and subclasses, the concepts are applicable in any programming paradigm. When the added property entails weakening preconditions or strengthening postconditions of existing entities, it is considered a refinement whereas the addition of independent properties constitutes enrichment. His approach leads to the definition of a strict partial ordering of entities similar to how Morgan [174] defined the relation $\sqsubseteq$ between programs. This partial ordering is useful when constructing a classification of algorithms. It is sad that the notion of enrichment was not widely adopted in computer science and software modeling. Authors often fail to distinguish between refinement and enrichment and treat the terms specialisation and refinement as synonyms.

Banach *et al.* [18] observe that many practitioners deem the formal program derivation using formal refinement inadequate in the face of the demands of real applications. They introduce the concept of retrenchment to accommodate practical development steps that may fail to adhere strictly to the formal definition of refinement as the sole method of passing from abstract to concrete models. These steps are typified as specification constructor tasks. Instead of weakening preconditions or strengthening postconditions, they elaborate specification details. According to Kourie's [148] definition, these are indeed enrichments. Gruner [106] finds Kourie's trivalent logical model problematic. On the other hand, Kovács [150] praises it and emphasises its importance for a formal definition of refinement in the context of knowledge management.

7.2.4 Facet analysis

The theory of facet analysis was introduced by Ranganathan [205, 206], who developed it because he was dissatisfied with the inability of traditional bibliographic classification systems at the time to allow for the expression of compound subjects [228].

Facet analysis is a categorisation technique that labels concepts according to an *a priori* defined classification system. It creates a number of inter-related yet orthogonal classifications, rather than having only one classification. When applying facet analysis, the analyser specifies relevant facets that can be used for the classification of objects in a chosen universe and furthermore defines a classification scheme for each facet. The versatility of the system to support the creation of compound subjects is thus greatly increased. When defining facets, each facet should describe a distinguishing characteristic of the objects in the context. The formation of sub-concepts can be applied to each facet, as well as to different concepts across facets.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The multiple overlapping hierarchies created through facet analysis have the advantage that they may produce unintended relationships between objects that could lead to discoveries that may not have been conceived otherwise [94]. Each facet represents a separate classification system, which can either be faceted into sub-facets or described in terms of an array of classes.

An important aspect of facet analysis is the application of what is termed citation order. It requires categorisation to follow strict rules of ordering. When an object is classified, the description of its position in the system represents the characteristics of the object. This description should always mention the involved facets and classes in the same prescribed order. This contributes to the predictability of the expression of compound concepts [42].

7.2.5 Justifying the use of a range of techniques

The application of abstraction and its converses, namely refinement and enrichment, is useful for identifying some of the attributes of algorithms and for creating implicit relations between algorithms. Both abstraction and facet analysis propose a bottom-up approach specify hierarchical relations while the top-down approach defined in Watson's process incorporates the approaches of Morgan [174] and of Kourie [148]. In practice alternation between a top-down approach and a bottom-up approach seems to be the norm.

The correctness of algorithms in a taxonomy is ensured by describing the taxonomy as if only a top-down approach was used. In this way each algorithm is described in terms of a well-defined sequence of refinements or enrichments that are applied to the root algorithm to arrive at the described algorithm. This does not prevent one to apply a bottom-up approach to identify abstractions before embarking on constructing derivations to prove the correctness of an algorithm.

Applying only abstraction, refinement and enrichment could, however, be problematic at times. For example, when a linear derivation path consisting of a number of techniques that can be applied to derive an algorithm is identified, the most appropriate order in which these techniques have to be applied may not be evident at first. If this is the case, the taxonomist has to decide which one of the alternative transformation paths that are formed by reordering the derivations is the most appropriate. It is, furthermore, possible that algorithms may have attributes that are not related to derivation. Examples of such attributes may be the computational complexity of the algorithm or the year of its first publication. Facet analysis provides a way to include such attributes in a taxonomy. It propose the notion to classify algorithms according to a range of classifications.

Application of abstraction, refinement and enrichment, typically results in a classification that constitutes one facet of a multifaceted classification. When using facet analysis other facets, that may be orthogonal to the derivation hierarchy, can be added. This leads to a structure that allows storage of richer information while maintaining the option to consider the information according to a single facet of choice.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.3 Representation models

7.3.1 Taxonomy

The word taxonomy is constructed by combining the Greek words τάξις, (*taxis* – meaning 'order') and νόμος, (*nomos* – meaning 'law' or 'science') [260]. The Webster dictionary has defined taxonomy since its 1828 version [181]. It, however, originated much earlier as a term used to refer to the systematic categorisation and naming of living organisms. Carl Linnaeus, who is known as the father of modern taxonomy, published a first edition of his *Systema Naturae* in the Netherlands in 1735 [259] and already used the term. The use of the term taxonomy is no longer confined to the classification of natural objects. It has become a general term used to refer to the categorisation of *anything*. Examples of well known hierarchical taxonomies are the Linnaean taxonomy of biological organisms, the Dewey Decimal Classification system for cataloguing books, as well as taxonomies used for geospatial classification, for regions, countries, provinces, and cities [111].

A taxonomy is created by grouping objects in a domain into categories in such a way that objects that are grouped together share some attributes. The set of shared attributes constitutes a concept in the domain. Often subgroups of groups can be defined several levels deep. Taxonomies are structures defining parent-child (*IS-A*) relations between the concepts that are so formed. When two concepts are in a hierarchical relation with one another the super-concept is called the *hypernym* of the sub-concept and the sub-concept is called the *hyponym* of the super-concept. It ensures the inheritance of properties from super-concepts to sub-concepts. Taxonomies can be classified according to the restrictions placed on the hypernyms and hyponyms in the taxonomy. This section explains the differences between three types of taxonomies.

7.3.2 Strictly hierarchical taxonomy

A strictly hierarchical taxonomy is a taxonomy in which a concept may have multiple hyponyms, but each concept should have only one hypernym. The position of every object in the taxonomy is uniquely determined. Mathematical models to describe and reason about hierarchical taxonomies were described by Brainerd [38], Schock [219] and Thomason [237].

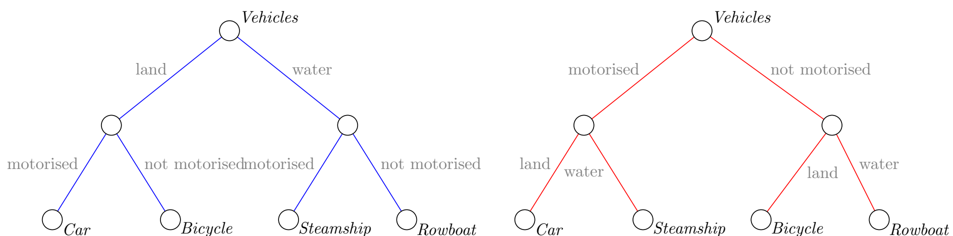


Figure 7.3.2: Two hierarchical taxonomies of one context, adapted from Frank [95]

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Figure 7.3.2 shows two different hierarchical taxonomies for a domain consisting of four objects: *Car, Bicycle, Steamship* and *Rowboat* using facets *land/water* and *motorised/not motorised*. These taxonomies have the same base objects but different interim concepts owing to using orthogonal facets, namely *terrain* and *propulsion* when classifying the vehicles.

As illustrated by this example, different hierarchical taxonomies result when the attributes used in the taxonomy are applied in a different order. It can be observed that the two taxonomies in Figure 7.3.2 have exactly the same shape when the objects *Bicycle* and *Steamship* in the image on the right are swapped. The objects in these taxonomies were listed here in the same order to simplify comparison and to highlight the differences. This simple example considers only two facets. Categorisation may, however, involve many more facets creating more variations. No wonder Broughton [41] states that hierarchical classification can cause difficulties when dealing with complex objects.

7.3.3 Duplication in strictly hierarchical taxonomies

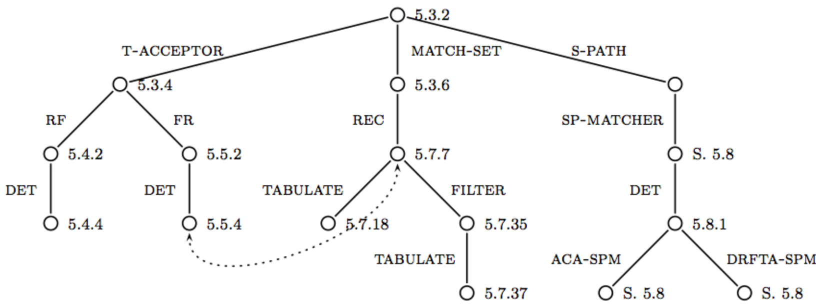


Figure 7.3.3: Taxonomy of tree acceptance algorithms [60]

The abstraction and refinement strategies described in Sections 7.2.1 and 7.2.2 can be applied in principle to identify hierarchical relations between algorithms. The construction of strict hierarchical taxonomies of algorithms, however, may pose problems. When a taxonomy has branches to differentiate algorithms that apply fundamentally different strategies, it is likely that these branches contain duplication. To illustrate this, consider the taxonomy of tree acceptance algorithms shown in Figure 7.3.3. It contains branches labelled T-ACCEPTOR and S-PATH which both contains DET as a branch.

It may sometimes be possible to derive an algorithm in multiple ways. For example in Figure 7.3.3 the node labelled 5.7.7 is connected to the node labelled 5.5.4 to indicate that they are essentially the same [60]. Duplication as well as difficulties to depict an algorithm that may be derived in different ways can be avoided when lattices are used instead of strict hierarchies.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.3.4 Semi-Lattice

In mathematics a lattice is a non-empty, partially ordered set along with two binary operations that are idempotent, commutative and associative, and satisfy the absorption law. The study of lattices is called lattice theory. Lattices offer a natural way to formalize and study the hierarchical ordering of objects. The only difference between a taxonomy and a lattice, as defined here, is the higher level of mathematical rigour applied in lattices.

Lattices are used to address difficulties that are experienced owing to the order of attribute consideration and multiple derivation paths, one can abandon the restriction on a taxonomy to be strictly hierarchical. The resulting structure is a semi-lattice. It is a hierarchical structure in which objects may have multiple hypernyms.

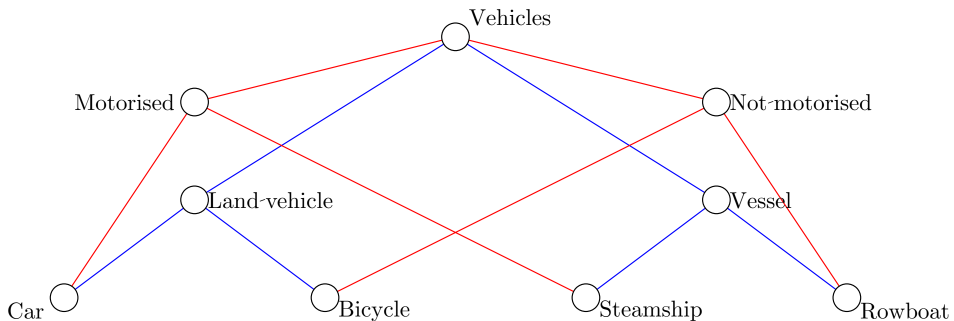


Figure 7.3.4: Merging the taxonomies in Figure 7.3.2 to form a semi-lattice

Figure 7.3.4 shows a semi-lattice taxonomy of the same domain that was taxonomised in Figure 7.3.2. It can be observed that this semi-lattice consists of a combination of the above mentioned two hierarchical taxonomies. It features the combined set of concepts formed in the hierarchical taxonomies. The previously unlabelled interim concepts in Figure 7.3.2 are now labelled with names representing these concepts. When forming concepts, it is expected from the taxonomist to assign labels to the formed concepts.

An important feature of the semi-lattice structure is that the order in which the attributes of the object are considered is irrelevant whereas the order in a strictly hierarchical structure is relevant. With a hierarchy a selected *permutation* of the attributes of an object determines its position in the taxonomy, whereas the position of an object in a semi-lattice is determined by the *combination* of the attributes concerned. The semi-lattice structure, therefore, has a more robust and predictable representation than a strict hierarchy. The use of semi-lattices increases the findability of information in the structure.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.3.5 Complete lattice

A concept lattice is a complete lattice in the mathematical sense of the word. In a concept lattice objects may have multiple hypernyms, and any two objects in the lattice, should have a single least common hypernym and a unique greatest common hyponym. The complete lattice in Figure 7.3.5 was drawn using the same domain as in the previous examples. Complete lattices are equipped with an algebraic structure which supports the automation of the use of the knowledge it embodies and allows computation [234].

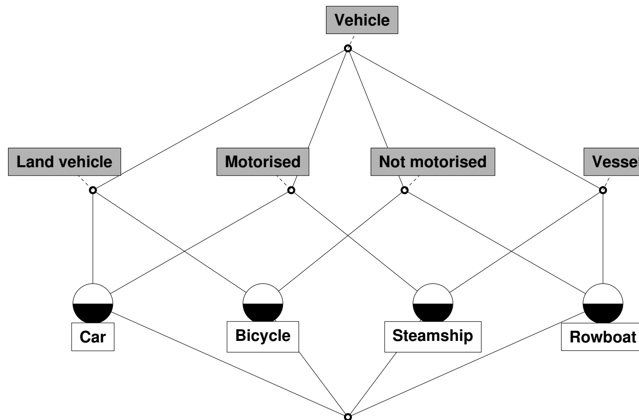


Figure 7.3.5: The taxonomy in Figure 7.3.4 as a complete lattice

Cleophas *et al.* [58] suggested an approach for taxonomy creation based on concept lattices as an enhancement of previous approaches. They constructed a concept lattice using the data in an existing taxonomy of keyword pattern matching algorithms that was previously constructed using abstraction and refinement. They showed that the concept lattice they constructed is comparable with the original taxonomy. It also had the ability to combine attributes that were previously duplicated in different parts of the taxonomy. This revealed similarities that were previously less obvious. Cleophas *et al.* [58] pointed out that attribute exploration techniques often used in the manipulation of concept lattices, described by Ganter [96], could potentially aid the discovery of new algorithms or highlight less obvious consequences.

7.3.6 Thesaurus

"Thesaurus" is a Latin word, which is the latinisation of the Greek word Θησαυρός (thĕsauros), literally meaning *treasure store*, generally meaning a collection of things which are of great importance or value [262]. The term is currently more often used to refer to a classified list of terms and their synonyms in a particular field. This change in meaning from a treasure collection to a dictionary of synonyms was

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

instigated by the publication of Roget's [209] book, titled *Thesaurus of English words and phrases*. Most likely the meaning of the word thesaurus in this title was chosen to describe the collection as a valuable resource of words and phrases for practical application. When Roget's book was first published in 1852, Roget described it as a *classed catalogue of words*. This book was widely used and gradually its title became synonymous with its intent, just as *google* nowadays is often used as a verb to signify searching the internet.

A thesaurus, when used for information retrieval, provides a controlled language used for indexing. It also provides alternative terms that can be used to improve recall [7, 111, 206]. A thesaurus is defined here to be a classification of concepts within a selected domain that allows for the definition of different types of semantic relations between its elements. A thesaurus can be contrasted with a taxonomy in terms of the types of relations that are definable in these entities. A taxonomy only allows for the definition of hierarchical relations (hypernymy and hyponymy), whereas a thesaurus allows for the definition of a number of other types of relations.

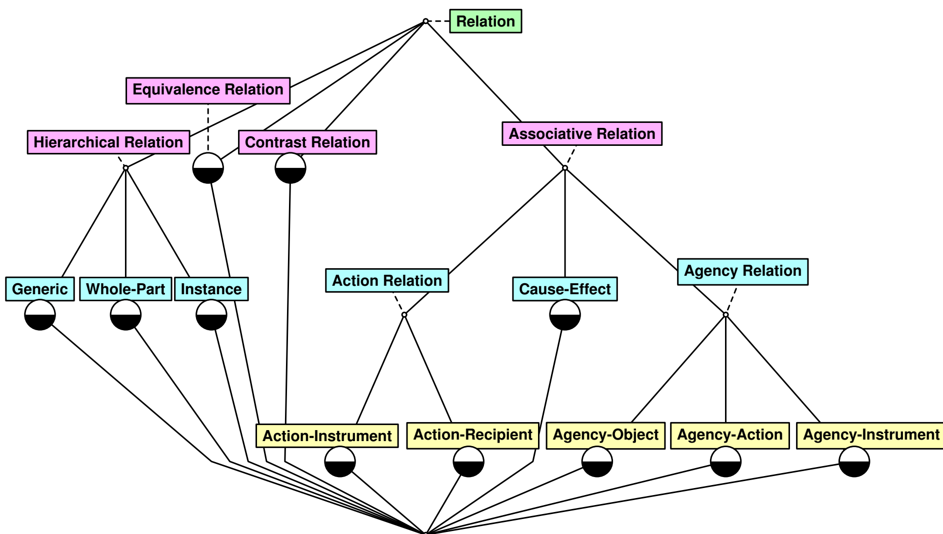


Figure 7.3.6: Relationship types based on relations identified by Tegarden [236]

The types of semantic relations that can be specified between objects in a thesaurus can be classified in the following four main types: *equivalence, hierarchical, associative* and *contrast*. The choice of what relations should be or may be included in a thesaurus depends largely on the purpose of the specific thesaurus. Figure 7.3.6 is a lattice showing a number of possible relationships that can be defined between objects in a thesaurus identified by Tegarden [236] as an application of relation element theory [47, 266].

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A wide variety of graphical displays can be used to visualise the relational information in thesauri. Such a display may consist of the thesaurus items connected to one another by different kinds of arrows representing the different types of relationships between the items. One may, for example, use tree diagrams [11], lattices [202] or hyperbolic trees [154].

7.3.7 Ontology

The word ontology is derived from the two Greek words όντος (*ontos* – meaning 'to be') and λογία (*logia* – meaning 'science', 'study' or 'theory') [261]. It is the philosophical study of the nature of being, existence or reality. As philosophical studies, ontology and epistemology are related. Epistemology is concerned with the nature and limitations of knowledge. Both these branches of philosophy deal with the nature of knowledge albeit with different perspectives. Ontology concerns the organisation of knowledge whereas epistemology deals with the sources of knowledge [107]. Ontology focusses on how knowledge can be represented. It aims to determine what entities exist and it philosophises about how these entities relate to one another, and how they can be classified.

The development of computer applications to perform intelligent actions necessitated the design of innovative knowledge representation models. One such model, a semantic network, was proposed by Sowa [227]. It is a graphic notation for representing knowledge in patterns of interconnected nodes and arcs. It provided the foundation for the design and development of formal systems for representing knowledge in structures that can support computerised reasoning. The development of ontologies occurred in knowledge-based systems (KBSs) which is a discipline within AI. It is important to note that a KBS uses its knowledge base not only to sensibly store and retrieve information, but also for reasoning [35]. In this context, the term *ontology* has been used for at least 30 years. A publication by McCarthy [165] in 1980 uses this term to refer to *the things that exist* in a domain description. However, it was only popularised more than a decade later, in the early 1990's, when Thomas Gruber at the Stanford Knowledge Systems Lab proposed specifications promoting inter-operability of different AI systems.

Thus an ontology is a thesaurus in the sense that it contains **concepts**, their **attributes** and their **relations**. The most prominent aspect that differentiate an ontology from other information systems is the inclusion of **inference rules** that can be manipulated by an intelligent controller, also known as an inference engine. The semantic expressiveness of ontologies exceeds that of other information repositories because of the availability of inference rules as well as the tendency to contain more detailed information about concepts, deeper hierarchical levels of concepts, and richer relationships between concepts. This view of ontologies differs from those knowledge workers who use of the term "ontology" in contexts where Pieterse and Kourie [199] regard the term "thesaurus" would be more appropriate.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.3.8 Justifying the creation of a thesaurus

Previously structures such as taxonomies [253], catalogues [85] and concept lattices [58] were used to organise information about algorithms. The relations between algorithms in these structures are limited to hierarchical relations. It seems appropriate to allow other types of relations. To achieve this, the it is recommended that a thesaurus schema be employed.

It appears that the compilation of a thesaurus of algorithms has not yet been attempted. Searches on 22 July 2013 using Gigablast[1], Yahoo![2], bing[3], and Google[4] for *"thesaurus of algorithms"* each produced no hits. Figure 7.3.8 is the result shown by bing. A previous google search on 14 Oct 2012 for this search string produced a single hit. It is a conference proceeding [39] containing fourteen papers in which the word *thesaurus* was used only once in one of the papers. In subsequent searches on 20 Aug 2016 three of these search engines produced the same single hit while Gigablast came up empty. This time the hit is in a book which was published in 2014. This book is on mobility data management and has thirteen chapters. Here also, the word *thesaurus* was used only once in one of the chapters [191]. The word is used to refer to the comprehensive collection of algorithms, methods and techniques that is the general result of research in mobility data management and not to refer to an artifact that contains these entities.

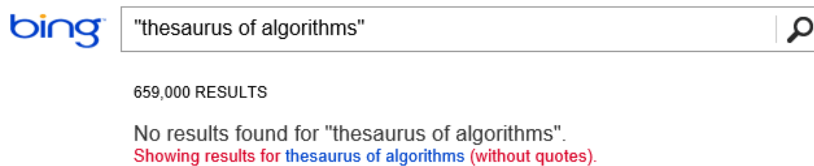


Figure 7.3.8: Screen shot of a search for "thesaurus of algorithms"

Without the quotation marks, these searches produced results ranging from definitions of the concept "algorithm" found in thesauri to algorithms to compile or visualise thesauri. No apparent collection of algorithms that is presented as a thesaurus could be identified. Despite the fact that thesauri have not, to my knowledge, been used to classify or taxonomise algorithmic information, their use in this context seems to be an appropriate extension of the existing norm. Thesauri can more easily deal with the variety of attributes that can be used to describe algorithms than other structures. They can also more easily deal with arbitrary relations specified between algorithms and their elements. It is believed that some of the collections of algorithms use such structure without explicitly calling it a thesaurus. For example the structure of the Dictionary of Algorithms and Data Structures (DADS) (Section 8.2) complies with the definition of a thesaurus.

[1] https://www.gigablast.com/
[2] http://search.yahoo.com/
[3] http://www.bing.com/
[4] https://www.google.co.za/

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

7.4 Representation

Because of the need for knowledge representation in all disciplines, many technologies and tools have been developed to support knowledge representation, both in industry and academia. To represent a thesaurus of algorithms a feasible technology is needed. After considering various representation technologies it was decided to use topic maps. It is beyond the scope of this thesis to evaluate alternative technologies. Instead a comprehensive description of the standard that was chosen for use in this thesis, namely topic maps, is given followed by a section giving arguments in favour of using it.

7.4.1 Topic maps (TMs)

Topic Maps (TMs) evolved from indices. Two (of several) definitions of index in the Merriam-Webster online dictionary [166] are appropriate in the context of this thesis. The first is a list (as of bibliographical information or citations to a body of literature) arranged usually in alphabetical order of some specified datum (as author, subject, or keyword). An example of this kind of index is the index card catalogues that were common in libraries before electronic indexing became popular. The other definition offered in the Merriam-Webster online dictionary is a list of items (as topics or names) treated in a printed work that gives a page number where each listed item can be found in the printed work. An example of this is the typical back-of-book index, which Pepper [194] describes as a concise and accurate *map* to the content of the book.

When a number of independent indices to the same corpus exists, a more comprehensive index to this corpus can be created by merging these indices. Also, when there is a need to merge different information sources into one, their respective indices need to be merged. The problem of how to merge indices gave rise to the development of the ISO/IEC 13250 standard [124], known as *the Topic Maps Standard*. It provides a standardised notation for interchangeably representing information about the structure of information resources. Pepper [195] argues that indices, glossaries, and thesauri can all be represented as TMs.

When building a TM one creates a description of the knowledge in a selected domain by formally declaring topics, and by linking the relevant parts of the information set to the appropriate topics as aptly described by Ogievetsky [183]:

> *In other words, to make sense of the labyrinths of the information world, topic map authors collect and structure networks of pointers into the multi-dimensional information universe, distinguishing and classifying subjects they want to talk about by representing them as topics and assigning these topics categorised characteristics that presumably belong to, describe, relate to, and/or elucidate those subjects.*

TMs provide means to structure unstructured information by providing an external markup mechanism that imposes structure on an unstructured corpus without having to alter its original form.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

TMs cater for detailed descriptions of the concepts (subjects) in the map in the form of attributes (called types) [193]. An example of an implementation of an index as a TM is the *Mother Encyclopaedia* of Polish Scientific Publishers reported by Ksiezyk [151] in 1998.

The application of the topic map standard promotes interoperability of applications that operate on indices. It supports the creation of automated applications to use multiple independent indices for information retrieval without physically merging them. This enables the creation of comprehensive indices to huge information repositories such as the Internet itself.

7.4.2 Justifying the use of TMs

Following through on the decision that a thesaurus structure is the schema of choice for the representation of algorithmic information, a suitable model for the definition of such a thesaurus in this thesis is needed.

Knowing that TM technology is designed for the representation of indices and thesauri, it seems to be suitable for the purpose of this thesis. It constitutes an elegant and yet powerful standard to use for describing the kind of semantic information that is envisioned to be part of any repository of information about algorithms.

TMs support the main features required for the representation of algorithmic information; they allow multiple concurrent hierarchies and provide for the definition of arbitrary relations and constraints. The following are some attractive properties of TMs that contributed to the decision to use this technology in this thesis as the standard for the representation of a thesaurus of algorithms:

- TM technology is published as an ISO standard [124] emphasising accessibility and portability of information resources. Bearing in mind that the algorithm repository in this thesis is created to enable programmers to share and reuse algorithmic solutions, it is essential that the information represented here should be accessible and portable.

- TMs enable the structuring of unstructured resources of any kind [155]. It is therefore relatively easy to incorporate any information about algorithms despite the unconventional character of some of the items one might like to include.

- The TM technology was developed to be able to merge independent indices to the same corpus in order to solve the problem of providing living master indexes [29]. The TM paradigm supports the automatic federation of diverse metadata sources [177]. If different algorithm repositories are defined as TMs, the procedure to combine them to form a larger and more comprehensive TM is already defined by this technology and can thus be automated.

- They are non-intrusive as the semantic information contained in a TM is represented by external, independently maintained metadata [119]. A TM provides a semantically rich knowledge layer over its associated resources. In essence a TM represents the knowledge embedded in a document corpus.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- The ability to use scope specifiers to define context allows the designer of a TM to organise the resources according to different points of view simultaneously [124]. This provides the tools to create a repository that can support users with diverse individual needs and views to find the optimal algorithmic solution to a problem in a given scenario.

Owing to the above mentioned advantages offered by TMs, it was decided to use TMs as the preferred technology for specifying an algorithm thesaurus.

7.5 Summary

This chapter is a review of work related to the work done in this thesis. It forms the foundation on which the artifacts that are created in this thesis are built.

The process model presented in Chapter 10 is an extension of techniques discussed in Sections 7.2. Although most of the work discussed in this section is well known and widely cited, it includes some forgotten gems such as the distinction between refinements and enrichments described by Kourie [148] and concepts in information science, introduced by Ranganathan [205], that has to my knowledge not been used in the context of classification of algorithms. The review is discussed using a fresh perspective to highlight the aspects that are specific to the work done in this thesis.

Representation models that are often used for knowledge organisation include taxonomies, lattices, thesauri and ontologies. These terms tend to be used inconsistently. Often different meanings are attached to the same terms. In Section 7.3 the meaning of these terms as used in this thesis is clarified. Based on this, the decision to create a thesaurus is justified. The discussion and definition of the terms discussed in this section is also the subject of a publication authored by me and my supervisor [199] in an accredited journal that publishes, among others, research articles that discuss problems of terminology with respect to specific fields.

The final section in this chapter lists technologies that were considered before deciding to use topic maps (TMs) to represent a thesaurus of algorithms. The section introduce topic maps (TMs) and justify its use in this thesis. TM technology is published as an ISO standard [124]. It is a specification standard that can be used to create taxonomies, thesauri and elementary ontologies. The core artefact produced in this thesis is a TM. This TM is described in Chapter 9 and in Part III.

The next chapter joins this chapter to provide background information for the specification of the metadata in Chapter 9. Where this chapter discusses processes and structures, Chapter 8 focus on the content i.e. the vocabulary needed to be able to describe algorithmic information.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 8

Existing algorithm repositories

This chapter joins Chapter 7 to provide background information for the specification of the metadata in Chapter 9. Where Chapter 7 discusses processes and structures, this chapter focusses on the content — i.e. the vocabulary needed to be able to describe algorithmic information. The metadata specified in Chapter 9 describe the core of a thesaurus of algorithms — i.e. the attributes of algorithms that are essential to describe most algorithms. This metadata is based on the fundamental attributes of algorithms described in this chapter.

Many repositories and textbooks were studied for the purpose of gathering information about attributes of algorithms that should be incorporated in the thesaurus of algorithms; specifically those that should be in the core of such thesaurus. For this reason, the investigation focused on the identification of attributes needed for the description of basic algorithms.

The structure and content of a number of algorithm collections informed and influenced the specification of the metadata needed for the description of algorithms and their related concepts. The collections that were reviewed include the famous multi-volume collection by Knuth [141, 142, 143, 144, 145] and various introductory books on algorithm design such as Manber [158], Kaldewaij [131] and Kleinberg and Tardos [140]. In order to create focus the algorithm attributes, which are incorporated in the metadata specified in Chapter 9, are reviewed in this chapter in context of their practical use in selected online algorithm repositories. These collections were chosen because they are publicly available for perusal and suitable to serve as examples of different kinds of algorithm repositories.

The the chosen repositories are discussed to highlight the aspects of these collections that played a role in specification of the metadata in Chapter 9, rather than a description of the repositories *per se*. The discussions revolve around the identification of algorithm attributes and related artefacts, used in these repositories, that should be incorporated in the core thesaurus of algorithms.

Every section in this chapter contains an overview of the relevant features of the chosen repository, followed by a discussion of how the structure and attributes identified in the repository influenced the specification of the metadata described in Chapter 9.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

8.1 The algorithm design manual

The Algorithm Design Manual is a textbook authored by Skiena [226]. Skiena also maintains a website hosting an electronic version of the book[1] which provides easy access and links to a wealth of references and downloadable implementations. The textbook is divided into two parts. One part deals with techniques that can be applied to design algorithms while the second part comprises a catalogue of algorithms, implementations and associated information. It is a catalogue of algorithmic problems that arise commonly in practice. A total of 75 problems including topics like sorting, bin packing, solving linear equations, network flow, shape similarity, string matching, convex hull, and many more are discussed.

8.1.1 Structure

The structure of entries in the second part of the book comprising a catalogue is of particular interest. Each entry in the catalogue deals with a specific problem and contains the following items:

- A graphical illustration of a typical instance of the problem in terms of two visualisations of a data model that is processed to solve the problem. One diagram visualises the input while the other visualises the required output. Figure 8.1.1 is the illustration of the TC problem in this textbook.

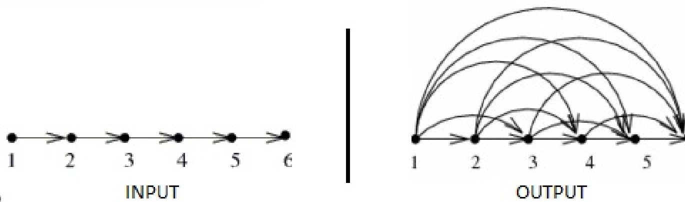


Figure 8.1.1: Visualisation of the transitive closure problem in Skiena [226].

- A formal textual description of the problem that is solved by the algorithm and its input data. The following is the formal description of transitive closure as it appears in this textbook.

> **Input description:**
> A directed graph $G = (V, E)$.
> **Problem description:**
> Construct a graph $G' = (V, E')$ with edge $(i, j) \in E'$ iff there is a directed path from i to j in G.

- A general discussion of the problem domain and possible scenarios where the practical problem reduces to solving the defined problem.

[1] `http://www.cs.sunysb.edu/~algorith/`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- A discussion stating the issues that need to be considered and how each of these issues can be addressed if needed.
- A discussion of possible alternative algorithms that can be applied to solve the problem. For each of these a concise textual description of the algorithm is given along with references to more detailed discussions of the specific algorithm.
- References to available implementations of the discussed algorithms. Each of these may include a description indicating the quality and usefulness of the implementation as well as the programming language of the implementation.
- Additional notes telling the history of the problem and referring to results primarily of theoretical interest. It contains a rich source of references to textbooks and academic publications about the problem and algorithms that solve the problem.

8.1.2 Contribution to the metadata specification

The metadata specification in Chapter 9 is largely based on the structure of the catalogue entries in the second part of Skiena's [226] book.

- Instead of only an informal visualisation of the precondition and postcondition of algorithms that solve the problem, the metadata specification provides for the formal expression of the precondition and postcondition as well as for any number and any kind of visualisation of the problem.
- The formal expression of the precondition and postcondition of the problem in the metadata specification is comparable with the textual description of the problem as it appears in Skiena's [226] book. The metadata specification adds a requirement to have an authoritative published subject indicator (PSI) for the problem.
- The metadata specification does not allow for general discussions of the problem domain or identification of issues related to the problem. This omission is only because it is beyond the scope of this thesis. The focus here is on detailed information about algorithms, rather than detailed information about problems and problem areas.
- Exactly as is in Skiena's [226] book, the metadata specification provides for a concise textual description of the algorithm for alternative algorithms that can be applied to solve the problem, as well as references to more detailed discussions of these algorithms. Additionally the metadata specification expands considerably on the level of detail about algorithms required, when compared to how Skiena [226] deals with alternative algorithms.
- The metadata specification provides for notes about the problem, which may be as comprehensive as in Skiena's [226] book.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- Similar to the treatment of references to available implementations here, the metadata specification allows for inclusion of a description indicating the quality and usefulness of the implementation as well as the programming language of the implementation. Additionally the metadata specification provides for referring to supporting artifacts that may be needed by the implementation as well as options to add references to visualisations and performance measurements of the implementation.

In summary, the metadata specification provides for the inclusion of most of the items of the catalogue entries of Skiena's [226] book. It scales down on requirements for problems, but considerably extends on aspects related to algorithms and implementations. It honours the relation between algorithmic problem and algorithmic solutions advocated in Skiena's [226] book.

8.2 NIST dictionary of algorithms and data structures

The *National Institute of Standards and Technology (NIST) Dictionary of Algorithms and Data Structures (DADS)* [32] is a website that was created in September 1998 and ever since has been maintained by Paul E. Black. Currently it contains about 1400 entries covering general algorithms and data structures. The design is minimalistic and special care has been taken to ensure guaranteed fast access with any browser. It is a dictionary containing concise definitions of algorithms, algorithmic techniques, data structures, archetypal problems, and related terms.

The coverage of detail about algorithms in DADS is limited when compared with the information provided by the repositories discussed in Sections 8.1 and 8.3. This is because DADS aims to provide definitions rather than explanations or discussions. It was a design decision to have short entries in DADS. This is related to the purpose of DADS namely to be a reference source. Because the entries are brief, the web pages are ranked high by Google and other search engines. The brevity also enhances the usability of DADS on small-screen devices.

Black [32] observed that entities included in DADS are sometimes known by different names or their names have different spelling options. This is something that is supported in topic maps as it is allowed to specify any number of names for a topic.

8.2.1 Structure

Each entry is classified as one of the following type:

- classic problem
- definition
- data structure
- algorithm
- algorithmic technique

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Each entry is also classified as belonging to an area such as Graphs, Numeric Computation, Parallel, Searching, etc.

The entries in DADS are classified according to multiple hierarchies and therefore complies with the definition of a faceted classification scheme. The structure of DADS allows for the definition of a variety of relationship types between different entries through cross-referencing. Because the structure of DADS allows the specification of hierarchical relations as well as other types of relations, DADS complies with the definition of a thesaurus (Section 7.3.6)

Each entry consists of a number of fields. The following describes these fields. The first three are required while the others are only used where applicable.

- A brief textual definition.
- A signature of the author of the entry. If the author has a profile page, the signature is a link to this page. Otherwise it links to a page that lists the contributors.
- A text showing how to cite the entry in academic writing.
- A note containing additional information or an explanation.
- Typical thesaurus-like links for example to broader terms, narrower terms, or aggregate relations.
- Links to other external resources
- Links to implementations.

8.2.2 Contribution to the metadata specification

Much of the structure specified in the metadata specified in Chapter 9 was borrowed from the structure of DADS.

- The metadata specification incorporates a brief textual definition, additional notes, references to more information, links to external resources, and links to implementations, all as in DADS, but with the necessary modifications.
- The thesaurus-like relations that are applied in DADS are inherent to TMs. Therefore these relations form a natural part of the metadata specification as a consequence of the decision to use TMs as the standard of choice for this specification.
- Similar to DADS, the metadata specification supports maintaining information about the author of entries and other information needed to cite the entry. Additionally the metadata specification provides specifically that a reference to the publication that first introduced a problem or algorithm may be included. Unlike DADS, the metadata specification does not explicitly include detail about how to cite an entry. It is, however, suggested that the site hosting an algorithm repository based on the metadata specification should provide a function to generate a complete citation for an entry when requested.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

- The classification categories of DADS (classic problem, definition, data structure, algorithm, or algorithmic technique) were all adopted as first class items in the metadata specification.

- The feature to index items under different names that is applied in DADS drew my attention to the importance of this feature. It is not a unique feature; it is advocated by Diamantini and Potena [74] and the repository discussed in Section 8.1 also applies it. The metadata specification does not explicitly cater for this feature. It is, however, implicitly assumed owing to TMs supporting this feature.

The structure of DADS significantly shaped the structure of the metadata specification when considering the organisation of the information with hierarchical as well as other types of relations. Owing to the inclusion of a variety of relation types, it should be called the *Thesaurus of Algorithms and Data Structures (TADS)*, but then it will not have its memorable name.

Most of the items of the entries in DADS are similar to items in *The Algorithm Design Manual* discussed in Section 8.1. These items are included in the metadata specification. Important additional features borrowed from DADS are the need to support easy citation of the items in the repository as well as the formalism applied when relations between items are specified.

8.3 Handbook of exact string matching algorithms

The *Handbook of exact string matching algorithms* by Christian Charras and Thierry Lecroqis was published in 1997. It is a collection of algorithms solving the problem of finding occurrences of a string (more generally called a pattern) in a text. This collection is published both in PDF format [50] and as a website[2]. This repository is of particular interest as it applies a comprehensive and consistent structure to represent the information about the different algorithms. The structure supports comparison of algorithms.

In contrast with the collections discussed in Sections 8.1 and 8.2, which each covers a range of problems each having only one or an unclassified list of only a few algorithms that solve the problem, this collection covers a single problem and offers a classification of numerous algorithms that solve the problem.

An introductory chapter summarises a classification of these algorithms in broad categories and highlights the similarities and differences of the algorithms in terms of the algorithmic techniques they apply. The short descriptions of the algorithms indicate how the algorithms relate to one another. The introductory chapter in Charras and Lecroq's [50] book includes detailed information about the data structures used by the algorithms in each of the categories. Each remaining chapter covers one algorithm.

[2] `http://www-igm.univ-mlv.fr/~lecroq/string/`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

8.3.1 Structure

Each chapter that covers one algorithm comprises the following sections:

- A textual description elaborating the main features of the algorithm.
- The program list of an implementation using the C programming language.
- A sequence of pictures to visualise the execution of the algorithm.
- Main features of the algorithm.

The features of each algorithm are given in the form of a list with items such as:

- Time and space complexities, and requirements for both preprocessing and processing phases.
- Expected number of comparisons.
- Bounds of the algorithm's delay.
- Relation with other algorithms. For example being a variation or refinement of another algorithm.
- Restrictions such as requiring an ordered alphabet or that it is most efficient for long patterns using a small alphabet.
- Data structures used.
- Algorithmic techniques used.
- Advantages such as, for example, that it is easy to implement or that it can easily be adapted to become an approximate string matching algorithm.

8.3.2 Contribution to the metadata specification

The metadata specification provides for the inclusion of most of the above items. The following items in the design of the metadata specification are borrowed from Charras and Lecroq's [50] book and are not shared by the collections discussed in Sections 8.1 and 8.2:

- The data structures used by the algorithm.
- The theoretical space and time complexities of algorithms.
- Visualisations of the algorithm.

The metadata include a variety of existing data structures and allow addition of new data structures. A structure for the specification of the computational complexity of problems, data structures and algorithms are specified. It allows for the specification of any number of complexities to allow complexities such as worst case, average case and best case or any other case.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Complexities can be specified for space and time as well as other resources. It also allows the specification of multiple instances of a variety of kinds of illustrations, including kinds that are yet to be discovered.

The following items are also supported by the other repositories previously discussed:

- Restrictions on the input data that may be required by the algorithm, for example requiring the data to be sorted. These are part of the precondition of an algorithm.
- Specifying relations between algorithms.
- Specifying the algorithmic techniques used by algorithms.
- Pointing to implementations of algorithms.

The following paragraphs describe the decisions that were made about items that are part of the structure of Charras and Lecroq's [50] book but which are not explicitly part of the metadata specification:

- The expected number of comparisons of the algorithm is not explicitly supported in the metadata specification. Although it is hard to think of a useful algorithm that does not make comparisons, the number of comparisons is not always relevant. Counting comparisons usually forms part of reasoning to support the specified computational complexity of an algorithm. Thus, this attribute is indirectly supported by allowing inclusion of a justification item associated with each specified complexity.
- The bounds on the delay of an algorithm do not form part of the metadata specification. Although it is relevant for all string matching algorithms, this attribute is not applicable to all kinds of algorithms. It is argued that this delay relates to pre-processing required by the algorithm. When adhering to the metadata specification, algorithms that require pre-processing should be treated as compound algorithms. Then pre-processing is seen as a sub-algorithm that can be treated as an algorithm in its own right. All attributes that can be specified for an algorithm may thus be specified for the sub-phases of a compound algorithm.
- It was decided not to support explicit specification of distinguishing advantages of an algorithm. Firstly because it is likely that such information would be a subjective opinion whereas the metadata specification aims to support describing objective facts about algorithms. Furthermore, such information can be included without having an explicit field for it simply by including it as part of a discussion of the algorithm.

The attributes of algorithms that provide for the appealing and consistent structure of the chapters in this book contributed to significant enrichment of the specified matadata. The most notable enrichments are the addition of support for a variety of visualisations and support for formal analysis of the computational complexity of algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

8.4 The Canterbury algorithm repository

The website of the Department of Computer Science and Software Engineering of the University of Canterbury hosts a relatively small algorithm repository[3].

When compared with the repositories discussed in Section 8.1 and 8.2 it covers similar content when considering the kind of algorithms included in the repository. It is, however, not nearly as comprehensive as DADS or Skiena's [226] book. On 20 August 2016 it contained 14 algorithms and 20 data structures.

8.4.1 Structure

It contains the following items per algorithm:

- A fairly comprehensive textual explanation of the algorithm.
- An explanation of the problem to be solved by specifying the precondition in the form of a graphical representation of a typical data instance. It also illustrates the working of the algorithm in a sequence of pictures.
- Downloadable source code of an implementation in C or C++. Makefiles and test harnesses needed to compile and run these programs are included.
- Some benchmark results.

8.4.2 Contribution to the metadata specification

The idea to include access to code that can be executed to produce benchmark results as seen here is applied in the metadata specification. This idea is also supported by Diamantini *et al.* [75] who mentioned that the ability to characterise algorithms by performance indexes may be handy when algorithms have to be evaluated. In the metadata specification this idea was taken a step further by allowing storage of benchmark results.

Another item that was observed on this website that is included in the metadata specification is the availability of additional items, such as test harnesses and makefiles, that may be needed to be able to use given implementations.

8.5 Summary

This chapter discussed *The Algorithm Design Manual* by Skiena [226], the *Handbook of exact string matching algorithms* by Charras and Lecroq [50], the *National Institute of Standards and Technology (NIST) Dictionary of Algorithms and Data Structures (DADS)* by Black [32] and *The Canterbury algorithm repository* hosted on the website of the Department of Computer Science and Software Engineering of the University of Canterbury.

[3]http://www.cosc.canterbury.ac.nz/research/RG/alg/repository.shtml

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

These repositories are discussed to illustrate the kind of attributes and associated artefacts that are needed to describe algorithms and to demonstrate how these attributes are used. They serve as backdrop to a discussion about the identification of attributes and the decisions about how they are included in the metadata described in Chapter 9.

The collection of repositories of algorithmic information discussed in this chapter is by no means comprehensive. The repositories were carefully chosen from a larger collection as a representative sample that is small enough to support the intention of this chapter.

The analysis of these repositories in terms of their structure and information items informed the design and implementation of the topic map described in Chapter 9.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 9

Specification of a TM of algorithms

This chapter describes the primary artefact produced as an outcome of the design science research conducted in this thesis. This artefact that has been created makes information about algorithms that solve computable problems more accessible to a variety of users. Exploration of knowledge about algorithms may lead to deeper understanding of the algorithms themselves, to the problems they solve, to the advantages and disadvantages of using the algorithms in different situations, to more insight into the techniques the algorithms use, etc. When the information is stored using a well defined structure the retrieval of the information for practical application is simplified.

The chapter applies the information in Chapters 7 and 8. Chapter 7 surveyed methods that have been described to find algorithmic information. Chapter 8 is an investigation of existing algorithm repositories. Both these revealed important elements of the metadata needed to describe algorithms. The gathered information is applied in this chapter to specify the metadata and to construct a TM based on the metadata.

The TM that is constructed in this chapter could form the core for any TM containing information about a specific set of algorithms and the problems it solves. It can also be extended or modified to form TMs dedicated to a larger subsection of the universe of algorithms. It aims to be applicable for algorithm collections constructed for a variety of applications: for professional use; for educational purposes; or for research goals. In service of this vision to support a variety of uses, the metadata has been selected to contain rich information about algorithms, including theoretical facts, discussions, implementations and visualisations.

Theoretically it should be possible to integrate different collections of algorithms that are defined as independent TMs based on these metadata with one other. This is possible because the TM technology was designed with integration in mind [193], and technology to integrate TMs has already been developed [29, 177].

A basic understanding of TMs and linear topic map notation (LTM) is assumed. The TM of algorithms is defined using LTM. These were described in Section 2.2.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

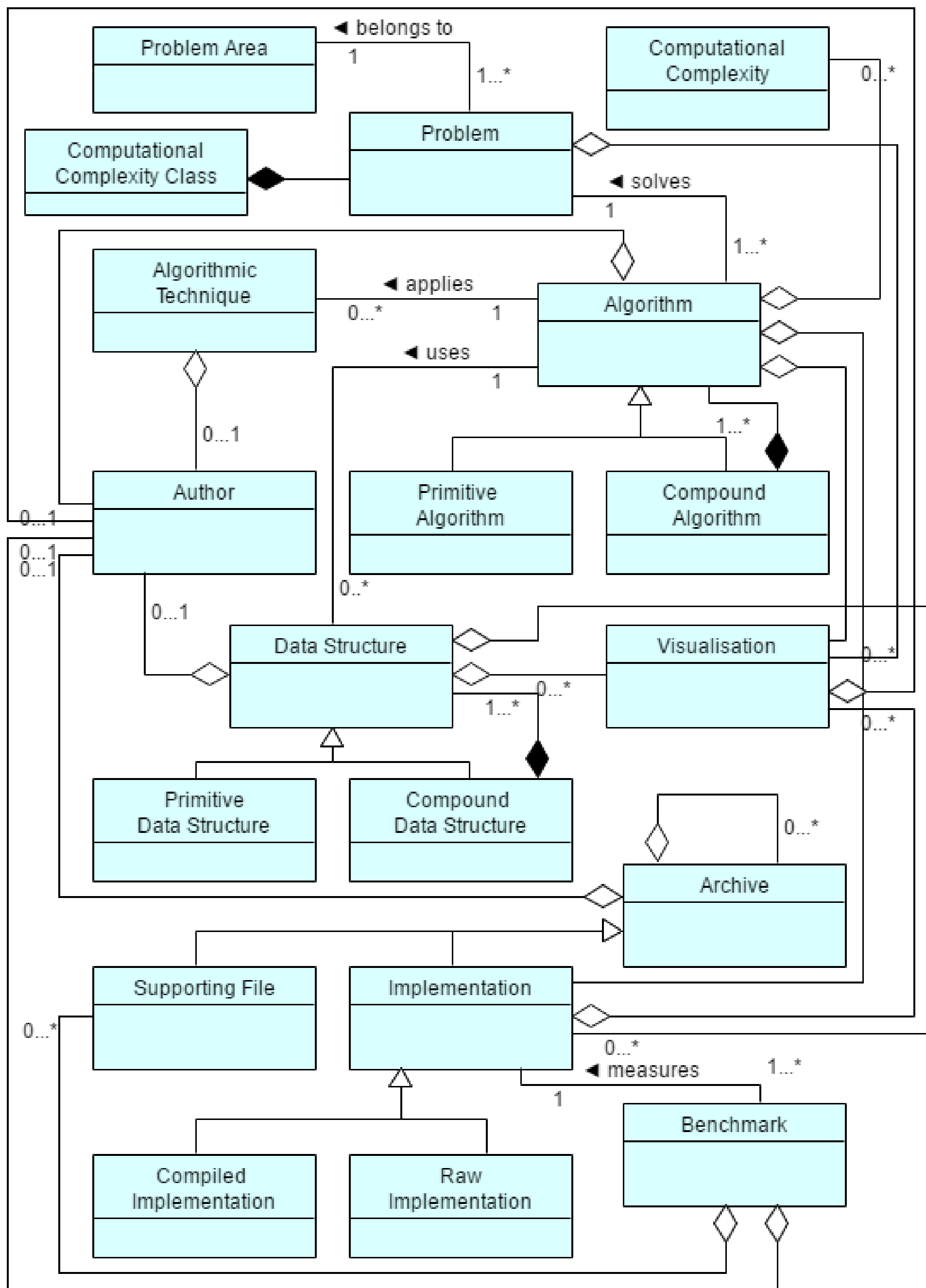


Figure 9.1.1: Relations between algorithm metadata classes

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

9.1 Concepts

This section lists the concepts that are needed to describe and classify algorithms. It explores the relations that may exist between algorithms and their attributes. The concept of an algorithm is central. Everything included in the TM of A is directly or indirectly related to an algorithm. The following terms were identified as concepts to be used in the TM of A:

1. Algorithm
   - Primitive algorithm
   - Compound algorithm
2. Algorithmic technique
3. Data structure
   - Primitive data structure
   - Compound data structure
4. Problem area
5. Problem
6. Computational complexity class
7. Computational complexity
8. Archive
   - Implementation
   - Supporting file
9. Visualisation
10. Benchmark
11. Author

The UML class diagram shown in Figure 9.1.1 was drawn for the purpose of this thesis to show the identified concepts and how they relate to one another. The acronyms for topic map entities used in this chapter are listed in Table B5 in the Appendix. Henceforth this generic TM of algorithms is simply called ***the TM of A***.

Unlike the other concepts, the concept *Author* is a general concept that is not an aspect or attribute of an algorithm. It is, however, deemed essential in any academic TM and therefore specifically for the TM of A.

In TM terminology one would refer to these concepts as *topics* in a TM. Their instantiations are called *occurrences*. In this thesis, however, the terms *classes*, *objects* and *attributes* as used in object oriented programming are often used instead.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

9.2 Topic definitions

9.2.1 Core topics

Listing 9.2.1 is a LTM listing defining the core concepts mentioned in Section 9.1 and shown in Figure 9.1.1 as Topic Types (TTs). The Published Subject Indicators (PSIs) of the TTs are mostly in the Dictionary of Algorithms and Data Structures (DADS) (Discussed in Section 8.2) — i.e. DADS serves as the authoritative source for the definitions of topic types.

Listing 9.2.1: Core topics of the TM of A

```
[Area ="Problem Area" ]
[Problem ="Computational Problem"
    @"http://en.wikipedia.org/wiki/Computational_problem" ]
[Algorithm ="Algorithm"
    @"http://www.nist.gov/dads/HTML/algorithm.html" ]
[PrimAlg : Algorithm ="Primitive Algorithm"
    @"http://www.nist.gov/dads/HTML/primitiveAlgorithm.html" ]
[CompAlg : Algorithm ="Compound Algorithm"
    @"http://www.nist.gov/dads/HTML/compoundAlgorithm.html" ]
[Technique ="Algorithmic Technique"
    @"http://www.nist.gov/dads/HTML/algorithmicTechnique.html" ]
[DataStructure ="Data Structure"
    @"http://www.nist.gov/dads/HTML/dataStructure.html" ]
[PrimData : DataStructure ="Primitive data structure"
    @"http://www.nist.gov/dads/HTML/primitiveDataStructure.html" ]
[CompData : DataStructure ="Compound data structure"
    @"http://www.nist.gov/dads/HTML/compoundDataStructure.html" ]
[Complexity ="Computational Complexity"
    @"http://www.nist.gov/dads/HTML/complexity.html" ]
[ComplexityClass ="Computational Complexity Class"
    @"http://www.nist.gov/dads/HTML/complexityClass.html" ]
[Visualisation ="Visualisation" ]
[Archive ="Archive" ]
[Implementation : Archive ="Implementation" ]
[Support : Archive ="Supporting file" ]
[Executable : Implementation
     ="Executable";;"Compiled implementation" ]
[SourceCode : Implementation
     ="Source Code";;"Raw Implemenation" ]
[Benchmark ="Benchmark" ]
[Author ="Author" ]
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

9.2.2 Occurrence types

As explained in Section 2.2.5 an occurrence of a topic is a link to an instance of the actual *thing* the topic represents. When defining occurrences of topics in the TM of algorithms it is indicated how these occurrence may be incorporated in the TM. Three ways, namely inline (Text), external (Ext) and internal (Intl) are distinguished. Occurrence types **Text** and **Ext** are in compliance with the TM standard while the use of the **Intl** occurrence type is a custom extension of the TM standard that is applied in this thesis as justified and explained in Section 2.2.6. These ways of specifying occurrences are explained in Table 9.2.2.

Table 9.2.2: Occurrence types of attributes in the TM of algorithms

Data Type	Description
Text	The occurrence of the aspect should be a word or a short sentence. The occurrence of the aspect is defined in-line by having this text enclosed in double square brackets as its occurrence locator. This information is stored in the TM.
Intl	The occurrence of the aspect is internal to the TM. It is another object in the TM containing the information about the aspect. This object has to exist as a topic in the TM. The occurrence of the aspect is defined by having the topic identifier (TI) of that topic as its occurrence locator. When applying this occurrence type, it constitutes a **HAS-A** relation between the topic and the occurrence.
Ext	The occurrence of the aspect is external to the TM. It is a resource containing the information about the aspect. The occurrence of the aspect is defined by having the URI of that resource as its occurrence locator.

9.2.3 Association types

Association type concepts identify the type of associations that can be defined between topics in the TM of A. Listing 9.2.3 is a LTM listing defining some Association Types (ACs)

Listing 9.2.3: Association type topics of the TM of A

```
[solves ="solves" ] /* Roles: Algorithm, Problem */
[uses ="uses" ] /* Roles: Algorithm, DataStructure */
[applies ="applies" ] /* Roles: Algorithm, Technique */
[measures ="measures" ] /* Roles: Benchmark, Implementation */
[belongsTo ="belongs to" ] /* Roles: Problem, Area */
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The remainder of this chapter defines the data items of the topic types in Listing 9.2.1 and describes how they are structured and related to one another in the TM of A.

9.3 Attributes of core topics

9.3.1 Algorithm

In computer programming, an algorithm is a method expressed as a finite list of well-defined computer executable instructions to achieve a desired result.

Formulating a formal definition to describe an algorithm, corresponding to the intuitive notion, remains a challenging problem [175]. Scriptol[1] refers to a number of definitions of an algorithm given by famous authors such as Knuth [141], Markov and Nagorny [161], Minsky [172] and Stone [230]. The aspects of this concept that are common among many definitions are requiring that an algorithm be expressed as a finite list of instructions; that the instructions of an algorithm being specified rigorously and unambiguously; and that the algorithm reach a well-defined goal within reasonable time.

Some definitions are more specialised than others, for example; the following definition of the noun *algorithm* from the Cambridge Advanced Learner's Dictionary [250] limits the concept *algorithm* to the mathematical domain:

> *A set of mathematical instructions that must be followed in a fixed order, and that, especially if given to a computer, will help to calculate an answer to a mathematical problem.*

The above definition requires determinism. This attribute for an algorithm seems to be widely assumed. In this thesis, however, non-deterministic specifications are also seen as algorithms. Note that non-determinism does not imply ambiguity. Non-determinism in an algorithm's specification allows for more than one possible next step to be taken at certain points in the algorithm's execution. The range of possible next steps is unambiguously specified, and the correct algorithmic outcome will result, irrespective of which possible execution path is followed.

The following definition of an algorithm by Schneider *et al.* [216] suffices for the purpose of this thesis:

> *An algorithm is a well-ordered collection of unambiguous and effectively computable operations that, when executed, produces a result and halts in a finite amount of time.*

Table 9.3.1 specifies a field name, an occurrence type[2], a multiplicity symbol[3], and a short description for each of the attributes of an algorithm in the TM of A.

[1] `http://www.scriptol.com/programming/algorithm-definition.php`

[2] See Table 9.2.2 for the meaning of these types

[3] See Table A6 for the interpretation of these symbols

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 9.3.1: Algorithm attributes

Attribute Name	**Occurrence Type**	**Multiplicity**	**Description**
Author	Intl or Text	1	The author of this algorithm entry in the TM.
Date	Text	1	The date this entry was made or updated.
Description	Text	1	A concise textual description of the algorithm.
Precondition	Text	0..1	Formal expression of the precondition for this algorithm
Specification	Text or Ext	1	A specification of the algorithm using GCL.
Verification	Text or Ext	$1 \cdots \star$	An argument in support of, or proof of the correctness of the algorithm.
Implementation	Intl	$\star$	An implementation of this algorithm.
Publication	Ext	$\star$	A publication where the algorithm was first introduced.
Complexity	Intl	$\star$	A description of a complexity of the algorithm.
Discussion	Text or Ext	$\star$	A discussion about the algorithm.
DataRestriction	Text	$\star$	A description of restrictions related to the input data for this algorithm, for example requiring the data to be sorted in some order.
Visualisation	Ext or Text	$\star$	A visualisation of this algorithm.
SubAlgorithm	Intl	$1 \cdots \star$	A sub-algorithm of this algorithm. *Only allowable if this algorithm is a compound algorithm.*

The *SubAlgorithm* attribute is grey to indicate that it is an attribute that does not apply to all algorithms. With the exception of `SubAlgorithm` attribute, occurrences of all the attributes shown in Table 9.3.1 may be defined for any algorithm. If the algorithm is a compound algorithm, at least one *SubAlgorithm* has to be specified.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following define topics related to algorithms in the TM of A. Occurrence types identify the topic names that should be used when specifying the attributes of an algorithm. Association types identify the topic names that may appear in associations with algorithms. Required items are shown in blue.

```
/* algorithm occurrence types */
[Author] [Complexity] [DataRestriction] [Date] [Description]
[Discussion] [Implementation] [Publication] [Specification]
[Verification] [SubAlgorithm] [Visualisation]
/* types associated with algorithms */
[DataStructure] [Problem] [Technique]
```

As shown in Figure 9.1.1, *Algorithm* is an abstract concept. Both *Compound Algorithm* and *Primitive Algorithm* are derived from *Algorithm*. This structure allows treating primitive algorithms and compound algorithms uniformly. Compound algorithms may be specified to contain any number of sub-algorithms, each of which could be compound. This recursive structure supports the definition of compound algorithms with arbitrarily complex structure. The following is the specification of this hierarchical relationship between *Algorithm* and its two derivatives:

```
[PrimAlg : Algorithm] [CompAlg : Algorithm]
```

When an algorithm is included in the TM of A, it has to be associated with the problem it solves. For this reason it is compulsory for each algorithm that is added to specify the problem it addresses. It is assumed that the precondition and postcondition specified for the problem apply to the algorithm.

A concise textual description is a compulsory item for each algorithm in the TM of A. It is like a dictionary entry and serves as the entry point to richer information. The brevity increases the navigability to the information it represents.

The author of each entry in the TM and the date of its entry is compulsory to support easy citation. Because it is likely that a specific author contributes multiple entries, authors are encouraged to create a topic in the TM of A to represent the themselves. An author topic may contain more information about the author. This topic can then be specified as the author occurrence of the contributions made by the author.

The precondition for the algorithm may be specified. If not specified, the precondition specified for the problem it solves is assumed. If the precondition for the algorithm is weaker that the precondition implied by the algorithm's position in the TM, it should be specified.

The specification and verification of an algorithm are required aspects. The specification of the algorithm in terms of GCL[4] is required because it is used in this thesis as a standardised notation that facilitates correctness reasoning. The verification is required owing to the policy that is applied in this thesis to ensure the correctness of algorithms that are added to the TM of TC algorithms that is developed in Part III of this thesis.

[4]Guarded Command Language (Section 2.3)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Bearing in mind that the TM defined in Part III of this thesis is designed to support software construction, algorithms should have implementations. Multiple implementations per algorithm are allowed. Despite the importance of having implementations for algorithms, it is not a compulsory item in the TM of A. This is because any TM of algorithms is likely to contain abstract algorithms that cannot be implemented without specialisation.

Each algorithm can be associated with any number of algorithmic techniques the algorithm applies. Algorithmic techniques as understood in the context of the TM of A are discussed in more detail in Section 9.3.3. With the exception of the root algorithm solving a problem (which applies no algorithmic techniques), each algorithm applies at least one algorithmic technique. These are used in the construction of a derivation hierarchy of algorithms.

A reference to the definitive first publication of an algorithm is preferred. If it is unknown it may be omitted. Multiple references are also allowed to accommodate a situation where there is controversy about which publication introduced the algorithm. The rest of the aspects are indicated to be optional. These aspects are its complexities, more detailed discussions about the algorithm, the data structures it uses, possible restrictions on the input to the algorithm, and visualisations.

Should other authors wish to extend the TM of A to build a TM of algorithms to serve a different purpose, they may relax some of these requirements and strengthen others. An example of such an adaptation may be to require a visualisation rather than a verification for each algorithm in a TM that is designed to serve an educational purpose.

9.3.2 Primitive and compound algorithms

A primitive algorithm is described in terms of programming steps that do not involve the execution of another algorithm. It is very common for algorithms to require another algorithm to be processed, for example to sort the input values before performing other specified operations.

The distinction between a primitive and a compound algorithm is subjective and may vary from one context to another. For instance the quicksort algorithm may be considered a primitive algorithm by one author while another author may define the phases in quicksort, namely *choose pivot* and *partition* as primitive algorithms, in which case the quicksort algorithm should be defined as a compound algorithm. A compound algorithm is normally described in terms of high level steps representing the execution of its sub-algorithms.

When entering an algorithm in the TM, the definition of the algorithm has to specify explicitly whether it is to be considered as a primitive algorithm (*PrimAlg*) or a compound algorithm (*CompAlg*). It is expected that when an author realises that a phase or procedure in a primitive algorithm that has already been added to the TM is also used in another algorithm, that existing primitive algorithm should be redefined as a compound algorithm. When doing so, the phase or procedure in question should be defined as a primitive algorithm while it should be specified that the refactored, redefined compound algorithm uses the newly defined primitive algorithm.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

When entering an algorithm in the TM, the location of the algorithm will assume its position in the derivation hierarchy of algorithms by specifying its direct parent in the derivation tree as its topic type in its definition. Owing to the transitivity of the **IS-A** relation, and the fact that its parent is an algorithm, the new algorithm inherits the attributes of its parent specified in the TM of TCA with the exception of being a compound algorithm or a primitive algorithm. The attributes as specified in Section 9.3.1 apply uniformly to primitive algorithms and compound algorithms.

The following is an example of the definition of a primitive algorithm with all the required attributes as it appears in the TM defined in Part III of this thesis:

```
[TCRoot : PrimAlg ="TC Root Algorithm"]
solves (TCRoot : Algorithm, TCProblem : Problem)
{TCRoot, Author, VP}
{TCRoot, Date, [[ 2013-02-14 ]]}
{TCRoot, Description,
    [[ The root algorithm solving the TC problem ]]}
{TCRoot, Specification,
    [[ THIS-Reference: Algorithm 11.2.2 (Root): Page 163 ]]}
{TCRoot, Verification,
    [[ The algorithm is trivially correct because it is the definition of TC. ]]}
```

If a compound algorithm is defined, the sub-algorithms that form part of the compound should be specified as occurrences of the compound algorithm. Each of these occurrences may be compound or primitive. Each compound algorithm should specify at least one algorithm that occurs as a sub-algorithm of the algorithm. The specification of a compound algorithm is a template method that calls each of the identified sub-algorithms appropriately.

The following is a partial definition of a compound algorithm and its sub-algorithms as it may appear in a topic map of algorithms for constructing Minimal Acyclic Deterministic Finite Automata (MADFA)[5]:

```
[Create : PrimAlg]
{Create, Description, [[ Choose a structural invariant ]]}
[AddWord : PrimAlg]
{AddWord, Description, [[ Add a word to the chosen structural invariant ]]}
[CleanUp : PrimAlg]
{CleanUp, Description, [[ Process the chosen structural invariant ]]}
[Skeleton : CompAlg ="MADFA Construction Skeleton"]
{Skeleton, SubAlgorithm, Create}
{Skeleton, SubAlgorithm, AddWord}
{Skeleton, SubAlgorithm, CleanUp}
{Skeleton, Specification,
     [[ Watson10-Reference: Algorithm 3.1, Page 24 ]]}
```

[5] See Section 2.2.5 for the definition of Watson10

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

9.3.3 Algorithmic technique

Algorithms differ from one another in their different approaches or strategies they apply to arrive at the intended solution. In the TM of A the concept of *algorithmic technique* is defined in a broad sense. The term is used to refer to any approach, strategy or method applied by an algorithm. Algorithmic techniques may range from high level strategies that can be applied to solve a variety of problems, to low level operations that are applied as steps in an algorithm. In DADS (Section 8.2) some well known techniques such as *Brute-force*, *Divide and conquer*, *Dynamic programming*, and *The greedy method* are defined. These form the high level classes of algorithmic techniques.

Sometimes algorithmic techniques specific to a given class of algorithms emerge when the algorithms in the class are analysed and classified, for example the techniques called *easysplit/hardjoin* and *hardsplit/easyjoin* were invented by Merrit [167, 168] when creating a taxonomy of sorting algorithms.

Often differences between algorithms that solve the same problem are described in terms of different algorithmic techniques they apply. This can be seen in the handbook of exact string matching algorithms (Section 8.3) where, for example, techniques such as *uses bitwise operations* and *compares from left to right* are used to highlight similarities and differences between algorithms on a more concrete level.

Any number of algorithmic techniques may be associated with each algorithm that is included in the TM of A. The algorithmic techniques included in the TM of A play an important role in constructing the derivation tree of algorithms and also in highlighting similarities and differences between algorithms. These techniques should be named and explained. If a technique is already explained in a source like Wikipedia or DADS, an Algorithmic Technique topic should be defined in the TM of A with a PSI pointing to this source. The following is an example of an algorithmic technique defined in DADS:

```
[EasySplit : Technique ="Easy Split, Hard Merge"
    @"http://www.nist.gov/dads/HTML/easySplitHardMerge.html"]
```

If an algorithmic technique is not general enough to be included in an authoritative resource, it has to be defined in the TM of A using an in-line definition. The following is an example that appears in the TM defined in Part III of this thesis:

```
[Sprout : Technique ="Sprout"]
{Sprout, Definition,
    [[ Construct the TC of R by growing a transitive relation T⊆R with
    edges of R while maintaining its transitivity until it contains all of R ]]}
```

9.3.4 Data structure

Data structures are needed as topics in the TM of A owing to the close relation between algorithms and their underlying data structures. Data structures are the fundamental constructs used to store the data on which algorithms operate.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Gonnet [103] defines an algorithm to be *a function that operates on data structures*. Often algorithms are designed around the appropriate data structure [226]. Many data structures were invented to serve an algorithm design. For example the data structure called PQ-trees was designed by Booth and Lueker [34] to serve their algorithm. Their algorithm uses PQ-trees to test the consecutive ones property[6]. This property is applied to recognise interval graphs.

Another example of a data structure that was created to serve an algorithmic purpose is the one invented by Basoglu and Morrison [22]. It is a geo-spatial data structure with temporal information which enabled them to write an efficient algorithm to retrieve the county boundaries of a given state for any date since that state achieved statehood.

Data structures, similar to algorithms, are required to be specified as primitive or compound in the TM of A. It is expected that they are mostly defined without any attributes other than its PSI. Other attributes such as a reference to its first introduction, as well as references to discussions and visualisations may be defined. Table 9.3.4 shows the specification of data structure topics in the TM of A.

Table 9.3.4: Data structure attributes

Attribute	**Occurrence**		**Description**
Name	**Type**	**Multiplicity**	
Author	Intl or Text	$0 \ldots 1$	The author of this entry in the TM.
Date	Text	$0 \ldots 1$	The date this entry was made or updated.
Definition	Text	$0 \ldots 1$	Only needed if the data structure is not in DADS.
Publication	Ext	$\star$	A publication where the data structure was first introduced.
Implementation	Intl	$\star$	An implementation of this data structure.
Discussion	Text or Ext	$\star$	A discussion about the data structure.
Visualisation	Ext or Text	$\star$	A visualisation of this data structure.
SubDataStructure	Intl	$1 \cdots \star$	A sub-data structure of this data structure. *Only allowable if this data structure is a compound data structure.*

[6]See `http://www.ic.unicamp.br/~meidanis/research/pqr/` for an explanation of the consecutive ones property

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Any number of data structures used in the implementation of an algorithm may be specified for an algorithm in the TM of A. The following define the topic names that should be used when specifying the attributes of a data structure object in the TM of A:

```
/* data structure occurrence types */
[Author] [Date] [Definition] [Publication] [Implementation]
[Discussion] [Visualisation] [SubDataStructure]
/* types associated with data structures */
[Algorithm]
```

Commonly known data structures are usually defined in DADS. These are included in the TM of A by defining them with a PSI pointing to their definition in DADS. The following is an example of the definition of a data structure that is defined in DADS:

```
[BTree: DataStructure ="B—Tree" ="Balanced multiway tree"
    @"http://www.nist.gov/dads/HTML/btree.html"]
```

If a data structure is not general enough to be included in an authoritative resource such as DADS, it has to be defined in the TM of A using an in-line definition. In this case the author who enters it and the date of entering this data structure to the TM should also be specified. The following is an example of an in-line definition of a data structure called *CharVector* that is based on the definition of a *Vector*.

```
[Vector : DataStructure ="Vector" ="Dynamic array"
    @"http://www.nist.gov/dads/HTML/dynamicarray.html" ]
[CharVector : Vector ="Vector of characters" ]
{CharVector, Author, VP}
{CharVector, Date, [[ 2013—07—30 ]]}
{CharVector, Definition,
    [[ A dynamic array. Each element in the array is a character ]]}
```

9.4 Topics specifying the context of an algorithm

9.4.1 Problem area

On a high level, algorithms may be classified according to the areas to which the problems they address belong. In the TM of A the topics of type *Problem area* are included to provide a high level classification of the algorithms in the TM of A. The intention is not to define the problem areas or provide more information about the problem areas. They are needed to provide a top level layer for future integration of different TMs of algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

When examining the classifications of algorithms and their problem areas offered by DADS (Section 8.2) and the Algorithm Design Manual (Section 8.1) as well as many other textbooks covering the topic of algorithms, it is evident that these problems are usually classified in areas such as Graph Problems, Numerical Problems, Combinatorial Problems, etc. An example of such a classification is the one in Wikipedia[7]. Each problem area is defined to be a set of related problems. There is no restriction preventing a specific problem to be associated with more than one problem area.

Instead of creating yet another classification of problems into problem areas, the TM of A uses the classification of problems as maintained in Wikipedia. The TM of A defines problem areas and some sub-areas as topics in the TM of A without any attributes. Wikipedia serves as an authoritative resource for the specification of PSI's. The following defines the areas related to the TC problem as it is specified in Wikipedia:

```
[Combinatorial : Area ="Combinatorial problem"
   @"http://en.wikipedia.org/wiki/Category:Combinatorial_algorithms" ]
[GraphProblem : Combinatorial ="Graph problem"
   @"http://en.wikipedia.org/wiki/Category:Graph_algorithms"]
```

If someone would like to create a TM of algorithms solving a problem that is not yet assigned to an area in Wikipedia, the problem needs to be added to this Wikipedia page and assigned to an appropriate category. This is what I did on 21 January 2013 with the problem called transitive closure[8].

9.4.2 Computational problem

There is a many-to-one relationship between algorithms and problems[9]. Each algorithm solves, or provide an acceptable near-solution, to a specific problem while several algorithms might address the same problem. Some problems have been identified as classic problems, such as travelling salesman, 8-queens, dining philosophers and knapsack problem. Others are general problems that often occur such as, scheduling, sorting, string matching, random number generation, polygon triangulation, etc. DADS contains a comprehensive list of named problems[10].

The definitions of the problems in DADS are used for problem PSIs in the TM of A. It is required that the PSI for the problem is specified upon entering the problem in the TM. If someone would like to create a TM of algorithms solving a problem that is not yet included in DADS, the definition of the problem needs to be submitted to DADS. The same applies to the complexity class of the problem.

[7]http://en.wikipedia.org/wiki/List_of_algorithms

[8]See `http://en.wikipedia.org/w/index.php?title=List_of_algorithms&action=history`

[9]In this context problem means *computational problem*

[10]`http://xlinux.nist.gov/dads/termsType.html#P`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Currently DADS is maintained by Paul E. Black[11]. The content of the website is stored as semi-structured data in plain text files. The HTML source for the website is generated using a script. The source code of the scripts used in the generation of a website as well as the data is available in a public git repository[12].

Table 9.4.2: Computational problem attributes

Attribute Name	Occurrence Type	Multiplicity	Description
Author	Intl or Text	0...1	The author of this entry in the TM.
Date	Text	0...1	The date this entry was made or updated.
Complexity Class	Intl	0...1	The complexity class to which the problem belongs.
Precondition	Text	0...1	Formal expression of the precondition for algorithms solving the problem
Postcondition	Text	0...1	Formal expression of the required postcondition of an algorithm solving the problem, given the precondition.
Publication	Ext	⋆	A publication where the problem was introduced.
UpperBound	Intl	⋆	The upper bound of the complexity of the problem.
LowerBound	Intl	⋆	The lower bound of the complexity of the problem.
Discussion	Text or Ext	⋆	A discussion about the problem.
Visualisation	Ext or Text	⋆	A visualisation of this problem.

Table 9.4.2 shows the attributes of problems in the TM of A. All of these items are deemed optional. If the problem contains in-line occurrences, the author is trusted to specify author and date occurrences. The location of a problem that is defined in the TM of A will assume its position in the hierarchy of problems according to the Wikipedia classification of problems simply by specifying its problem area classification as its topic type in its definition.

[11] People who want to add or edit entries should get in touch with Paul E. Black <paul.black@nist.gov>.

[12] `https://github.com/vpieterse/dads`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following define topics related to problems in the TM of A. Occurrence types identify the topic names that should be used when specifying the attributes of a problem. Association types identify the topic names that may appear in associations with a problem.

```
/* problem occurrence types */
[Author] [ComplexityClass] [Date] [Discussion] [LowerBound]
[Postcondition] [Precondition] [Publication] [UpperBound]
[Visualisation]
/* types associated with problems */
[Algorithm]
```

The following is a specification of the TC problem. It is a `GraphProblem` that was defined in Section 9.4.1 within its complexity class as defined in Section 9.4.3. It also has several occurrences of discussions and visualisations:

Listing 9.4.2: The TC problem

```
[TCProblem : GraphProblem Problem ="Transitive Closure"
    @"http://www.nist.gov/dads/HTML/transitiveClosure.html"]
{TCProblem, Author, VP}
{TCProblem, Date, [[ 2013-03-21 ]]}
{TCProblem, ComplexityClass, P}
{TCProblem, ComplexityClass, NC}
{TCProblem, Precondition, [[ R ⊆ U × U ]]}
{TCProblem, Postcondition, [[ THIS-Reference: Section 6.1.1: Page 79 ]]}
{TCProblem, Discussion,
    [[ THIS-Reference: Section 1.7: Page 8 ]]}
{TCProblem, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html"}
{TCProblem, Discussion,
    "http://en.wikipedia.org/w/index.php?title=Transitive_closure"}
{TCProblem, Discussion,
    "http://mathworld.wolfram.com/TransitiveClosure.html"}
{TCProblem, Visualisation,
    [[ THIS-Reference: Figure 8.1.1: Page 108 ]]}
{TCProblem, Visualisation,
    "http://www.cs.sunysb.edu/~algorith/files/transitive-closure.shtml"}
{TCProblem, Visualisation,
    "http://anh.cs.luc.edu/363/notes/09dynProg.html"}
{TCProblem, Visualisation,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html"}
belongsTo (TCProblem : Problem, GraphProblem : Area)
```

The formal expression of the precondition as well as the required postcondition, given the specified precondition for algorithms solving the problem should be specified in-line. The problem that is solved by the algorithms discussed in Part III is discussed in Section 6.1.1.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The computational complexity class of a problem should be defined as an occurrence of a complexity class as specified in Section 9.4.3. If known, the upper and lower bounds of the computational complexity of the problem may also be specified by defining complexity objects to specify each of these, and use their TI's as occurrences of type `UpperBound` and `LowerBound`. The description of complexities is discussed in more detail in Section 9.4.4.

Other attributes that may be associated with a problem are the publications where the problem was first introduced, as well as additional discussions and visualisations of the problem. These are treated the same as similar occurrences for algorithms.

9.4.3 Computational complexity class

In computability theory it has been established that certain problems cannot be solved by algorithms at all. These problems are called undecidable. The halting problem [71] is a frequently used example of an undecidable problem.

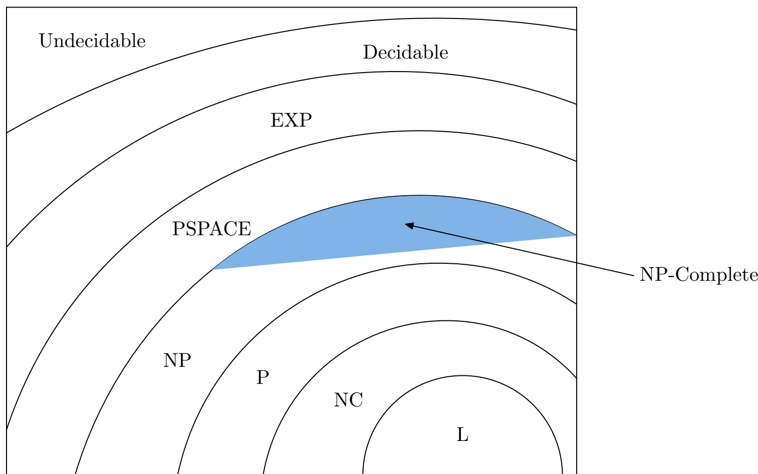


Figure 9.4.3: Computational complexity classes adapted from [189]

Problems are often classified in terms of their computational complexity. The names and definitions of some well known computational complexity classes are listed in Table B6 in the Appendix.

Figure 9.4.3 shows the relation between these classes in a diagram adapted from Papadimitriou [189]. If one region in the diagram contains another region, then the corresponding complexity classes contain one another in the same way. Some of these containments are proper. For example; by a straightforward quantitative extension of the diagonalisation proof which establishes that the halting problem is undecidable, it can be shown that there are problems in EXP that are not in P

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[110]. According to Papadimitriou [189], whether the containments in this figure are proper is for most of this diagram a subject of conjecture.

In the TM of A complexity classes are defined as topics without any attributes. The following are selected examples of their definitions using DADS to specify their PSI's. Similar entries for other complexity class can be found in the TM of A.

```
[Undecidable : ComplexityClass
    @"http://www.nist.gov/dads/HTML/undecidableProblem.html" ]
[EXP : ComplexityClass
    @"http://www.nist.gov/dads/HTML/exponential.html" ]
[NP-Hard : ComplexityClass
    @"http://www.nist.gov/dads/HTML/nphard.html" ]
```

9.4.4 Computational complexity

The complexity of problems and algorithms are typically expressed in terms of the shape of the relation between the size of its input data to the size of the resources required by the algorithm to solve the problem. Usually resources are specified in terms of a limiting factor. For one problem or machine, the number of floating point multiplications may be the limiting factor, while for another, it may be the number of messages passed across a network. Other measures that may be important are compares, item moves, disk accesses, memory used, CPU cycles, or elapsed time. The limiting factors may be influenced by the available resources such as number of processors, CPU speed, connections, logical gates, etc.

Often the complexity of an algorithm that solves (or produce an acceptable near-solution of) a problem is different from the complexity of the problem itself. For this reason the TM of A provides for the stipulation of the complexity of a problem as well as the complexity of the individual algorithm separately.

It might be possible to prove in theory that the solution to a problem has a certain complexity, while all the known practical solutions to the problem are of higher complexity. For example the minimal spanning tree problem has linear complexity although all known algorithmic solutions are polynomial or worse [197]. The best algorithms that have been devised are almost linear in practice but are mostly more complex in theory [184].

The converse may also occur. On occasion it is possible to devise solutions with lower complexity, albeit lower accuracy, for problems with theoretical unacceptably high complexity. An example is the well known travelling salesman problem with NP-hard complexity [268] for which a number of feasible, possibly sub-optimal, solutions are in use.

A widely accepted notation for expressing these complexities is known as Big O Notation that is used to describe the limiting behaviour of a function when the argument tends towards a particular value or infinity [170, 226]. Big O notation is used in the TM of A to formulate complexities.

As indicated in Table 9.4.2, a problem may be characterised in terms of an upper bound and/or a lower bound complexity object. The upper bound is determined

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

by the complexities of known algorithms that solve the problem. It is typically the worst case complexity of a known algorithm with complexity worse or equal to the complexities of the known algorithms that solves the problem. Lower bounds are usually derived by means of a mathematical argument. A lower bound argument makes a statement about all possible algorithms, including algorithms that solve the problem yet to be discovered in the future.

When specifying a complexity object in the TM of A, it is compulsory to indicate what resource is considered. The calculated complexity should be specified using Big O Notation. The appropriate symbols (O, Ω, or Θ)[13] should be used, to indicate which complexity is specified. A different complexity object should be created to specify the complexity for each resource and for each kind of complexity.

Usually complexities are justified through mathematical reasoning. This reasoning may be provided in-line or as an external occurrence that contains this reasoning. If the justification is given in-line the author and date of this entry needs to be specified. Table 9.4.4 shows how these are included in the TM of A.

Table 9.4.4: Computational complexity attributes

Attribute Name	**Occurrence Type**	**Multiplicity**	**Description**
Author	Intl or Text	1	The author of this complexity entry in the TM.
Date	Text	1	The date this entry was made or updated.
Resource	Text	1	The name of the resource analysed i.e. memory, CPU time, connections, etc.
Specification	Text	1	The complexity of the algorithm/problem in consuming the mentioned resource using Big O notation.
Justification	Text or Ext	$0 \cdots \star$	Mathematical justification for the specified complexity.

The following define the topic names that should be used when specifying the attributes of a complexity object in the TM of A:

```
/* computational complexity occurrence types */
[Author] [Date] [Justification] [Resource] [Specification]
```

[13] See Table A3 for the definition of the complexity functions

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following in an example of the definition of a complexity object. It is the definition of the worst case time complexity of Prosser's algorithm to calculate the TC of a given binary relation as it appears in the TM defined in Part III of this thesis:

```
[ProsserTimeComp : Complexity
    ="ProsserTimeComp";;"Complexity of Prosser's algorithm"
{ProsserTimeComp, Author, VP}
{ProsserTimeComp, Date, [[ 2013−07−30 ]]}
{ProsserTimeComp, Resource, [[ Time ]]}
{ProsserTimeComp, Specification, [[ Θ(n^4) ]]}
{ProsserTimeComp, Justification,
     [[ THIS−Reference: Section 12.4.4: Page 178 ]]}
```

9.5 Supporting Information

9.5.1 Archive

Archives are a means to support the inclusion of the implementations of algorithms and data structures in the TM. It is assumed that implementations are published as archives which may contain source code, compiled code, data files, and other supporting files. Here an abstract class called *Archive* is specified. It is generalised to allow the inclusion of archives that need not contain only implementations, but may also contain supporting files.

Providing access to implementations of algorithms seems to be the norm for algorithm collections. Most of the collections that were investigated, in particular all those that were chosen as representative of such collections (Section 8), either include code or point to code that implements all or some of the algorithms in the collection. The TM of A is no exception in this regard. Availability of implementations is required to support software construction in order to adhere to this important aim of TABASCO. Although the required GCL specification on an algorithm is deemed sufficient, many users of a TM of A may benefit by implementations in popular high level programming languages.

Implementations also have the potential to aid learning. They can provide opportunities for students to experiment with the code. Students can use the implementation to explore how the algorithm works or study the source code to gain better understanding of how the operations of the algorithm are implemented. They can perform test runs of an algorithm with different data sets and perform benchmarks of their own.

As can be seen in the class diagram in Figure 9.1.1, the concepts *'Archive'*, *'Supporting file'* (Support), *'Implementation'*, *'Raw implementation'* (SourceCode) and *'Compiled implementation'* (Executable) are related in a hierarchy. Archives can either be implementations or supporting files, while implementations in turn is either source code in a specified programming language or an executable program that was compiled for a specified operating system.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following defines the relations between the topics Archive, Support, Implementation, SourceCode and Executable:

```
[Support : Archive]
[Implementation: Archive]
[SourceCode : Implementation]
[Executable : Implementation]
```

If an implementation contains source code, it is called a *raw implementation*. If only compiled code is provided, it is called a *compiled implementation*. An archive may contain both source code and compiled code. In such case both programming language and the operating system need to be specified.

Table 9.5.1 shows aspects of archive items that can be specified in the TM of A. It may seem strange that the prescribed structure does not specify that the **File** attribute is a compulsory item as an archive is per definition a downloadable fileIt is specified in this way to allow the specification of an abstract archive object with children objects that points to the required files.

The following define the topics to use when specifying the attributes of an archive object in the TM of A. Required items are shown in blue.

```
/* archive occurrence types */
[Author] [Date] [File] [Discussion] [Size]
/* implementation occurrence types */
[Archive] [Visualisation]
/* raw implementation occurrence types */
[Language]
/* compiled implementation occurrence types */
[OperatingSystem]
/* types associated with implementations */
[Benchmark]
```

When an archive object is created in the TM, it is required that the author of the content of the archive, the date of publication, as well as the size of the archive are specified. Additional information about the archive can be specified in-line as a discussion. The following is an example of the detail about multiple implementations as an abstract implementation as it might appear in a TM of tree acceptance algorithms:

```
[ForestFIRE : Implementation
    ="ForestFIRE" ; "forestfire" ]
{ForestFIRE, Author, [[ R.G.W. Strolenberg ]]}
{ForestFIRE, Date, [[ 2008-01-08 ]]}
{ForestFIRE, Discussion,
    "http://www.loekcleophas.com/wp-content/uploads/2013/05
    /forestfire.pdf"}
{ForestFIRE, Support, FIREWood}
{ForestFIRE, Support, FIREData}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 9.5.1: Archive attributes

Attribute Name	Occurrence Type	Multiplicity	Description
All			
Author	Intl or Text	1	The author of the files in the archive
Date	Text	1	The date of publication of the archive.
Discussion	Text or Ext	$\star$	A discussion about the archive, such as a description of the content of the archive, its quality or its use.
Size	Text	1	The size of the archive.
File	Ext	$0\ldots1$	Reference to the saved archive
Only implementations			
Support	Intl	$\star$	Reference an artefact such as data files, test data, a compiler, a test harness or a software library that is associated with the archive.
Visualisation	Ext or Text	$\star$	Reference to a visualisation of this implementation.
Only raw implementations			
Language	Text	1	The programming language that was used for the implementation.
Only compiled implementations			
OperatingSystem	Text	1	The name and version of the operating system for which the code is compiled.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

If the archive is defined as an implementation, related archives and visualisations may be added. In this example two related archives and no visualisations are specified. When an implementation requires additional supporting artefacts such as test data, these may be included in the same archive as the implementation mitigating the need to specify related archives. If they are, however, included in separate archives, as in this example, more archive objects are needed. The occurrences called *FIREWood* and *FIREData* in this definition of *ForestFIRE* are topics that should be defined similarly in the TM. They are respectively a GUI and test data to support the use of *ForestFIRE*.

In this example *ForestFIRE* is a toolkit containing implementations of a number of algorithms. It may be specified as the implementation of each of the algorithms in the toolkit, or it may be associated with an algorithm appearing as a parent of these algorithms in the TM.

The following is the definition of a raw implementation of the above mentioned implementation:

```
[ForestFIRE-J : SourceCode ForestFIRE
    ="ForestFIRE−Java" ; "forestfirejava" ; "ForsetFIRE (Java source)"]
{ForestFIRE-J, Size, [[ 316 KB ]]}
{ForestFIRE-J, Language, [[ Java ]]}
{ForestFIRE-J, File,
    "http://www.loekcleophas.com/wp−content/uploads/2013/05
    /FFW−src−200801081409.zip" }
```

The following is the definition of a compiled implementation of the above mentioned raw implementation:

```
[ForestFIRE-L : Executable ForestFIRE
    ="ForestFIRE−Java";"forestfirejava";"ForsetFIRE (compiled on Linux)"]
{ForestFIRE-L, Discussion,
    [[Archive containing JARs compiled with GTK 2.2.1]]}
{ForestFIRE-L, Size, [[ 1.33 MB ]]}
{ForestFIRE-L, OperatingSystem, [[ Linux ]]}
{ForestFIRE-L, File,
     "http://www.loekcleophas.com/wp−content/uploads/2013/05
     /FFW−linux−200801081417.zip" }
```

9.5.2 Benchmark

The efficiency of an implementation of the algorithm depends upon consumption of resources. Software benchmarking is the process of measuring an algoritm's resource usage while it executes. Most often memory consumption and execution time is measured. However, other attributes such as energy consumption, bandwidth usage, number of connections, etc. may also be measured. This is done in order to compare the performance of different algorithms in terms of a measured resource.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

An important application of benchmarking results is their contribution to software quality assurance. It is widely accepted that the quality of a software product is determined by its conformance with its non-functional requirements [25]. In many contexts, constraints with respect to execution time and memory consumption are important requirements and therefore quality determinants. In these systems benchmarking is an important part of software quality assurance.

In the TM of A the benchmark serves one of the goals of TABASCO, namely to support software construction. It is known that some characteristics of algorithms, such as their performance, may vary given different constraints or different kinds of data. If a programmer has access to information about the practical performance of various algorithms under the conditions of interest, a more informed decision can be made about which algorithm to use when constructing software.

Another use of benchmarking is its educational value. Chen at al [51] pointed out that algorithm benchmarks can be applied as a very useful tutoring tool for students to review the notions of space and time complexity.

Table 9.5.2 shows aspects of benchmark specifications that can be specified in the TM of A.

The following define the topic names that should be used when specifying the attributes of a benchmark object in the TM of A. Required items are shown in blue.

```
[Author] [Description] [Date] [Discussion] [Hardware]
[InputData] [Resource] [Result] [Time]
/* types associated with benchmarks */
[Implementation]
```

9.5.3 Visualisation

It is possible to visualise how an algorithm works without the need to associate the visualisation with a specific implementation. Thus a visualisation can be an attribute of an algorithm as well as an attribute of an implementation of an algorithm. Visualisations may also appear as attributes of other items in the TM of A. The TM is designed to allow the addition of visualisations of data structures, implementations of data structures and also of problems.

An extension of the TM of A may serve as an information source about algorithms for educational purposes. Although it has been shown that *how* students use algorithm visualisation technology has a greater impact on effective learning than what is actually visualised [117],There are many documented pedagogical improvements that are supported by algorithm visualisation [93].

A visualisation may be an animation or series of images or diagrams. Animations can be implemented using supporting technologies such as JavaScript while diagrams can be used to illustrate the requirements of a problem, the characteristic workings of the algorithm or the execution of a specific implementation for a well chosen data example demonstrating the algorithm's ability to cope both with special and common cases. In the TM of A visualisations may appear as occurrence types of problems, data structures, algorithms as well as implementations.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table 9.5.2: Benchmark attributes

Field Name	Data Type	Multiplicity	Description
Author	Intl or Text	$0 \ldots 1$	The author who designed the benchmark and interpreted its results.
Date	Text	1	The date this benchmark was published.
Time	Text	1	Time-stamp of the starting time of the benchmark.
Resource	Text	1	The name of the measured resource i.e. memory, CPU time, connections, etc.
Hardware	Text or Ext	1	Description of the hardware used to execute the benchmark.
Description	Text or Ext	1	Description of the input data that was used in the benchmark.
InputData	Intl	$\star$	An archive containing the input data that was used in the benchmark.
Result	Intl	$1 \cdots \star$	An Archive containing the raw data that was produced by the benchmark. It is assumed that the result sets correspond with the input data sets.
Discussion	Text or Ext	$\star$	An interpretation and/or visualisation of the benchmark results.

There are many websites hosting collections of algorithm animations. *Sorting Algorithm Animations*[14], *Algorithms in Action*[15] [229] and *Algoviz*[16] [223] are all examples of such websites. The latter is a portal where the education community is invited to participate in contributing algorithm visualisations and evaluate the quality of the contributions.

Instead of providing yet another metadata specification for visualisations, the TM of A supports the description and classification of visualisations as maintained by AlgoViz. Therefore, authors who extend the TM of A are encouraged to submit visualisations to AlgoViz and define occurrences of visualisations in their TM's to point to catalogue entries in AlgoViz.

[14]http://www.sorting-algorithms.com
[15]http://ww2.cs.mu.oz.au/aia/
[16]http://algoviz.org/

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following example illustrates how a few visualisations of the Quicksort algorithm may appear in a TM of sorting algorithms adhering to this suggestion:

```
{Quicksort, Visualisation, "http://algoviz.org/node/1286"}
{Quicksort, Visualisation, "http://algoviz.org/catalog/entry/530"}
{Quicksort, Visualisation, "http://algoviz.org/catalog/entry/758"}
```

It is admissible to define occurrences of visualisations to point to other URLs. Assume *Boyer-Moore* and *Rabin-Karp* are TI's of string matching algorithms in a TM of string matching algorithms. The following are the specifications of occurrences of visualisations of these algorithms as they may appear in the TM containing these algorithms:

```
{Boyer-Moore, Visualisation,
    "http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/strMatching/
    boyerMoore/bmm.html"}
{Rabin-Karp, Visualisation,
    "http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/strMatching/
    robinKrap/rk.html"}
```

Visualisations may also be specified using the syntax for a custom locator defined in Section 2.2.7.

The following example illustrates this. It is an occurrence of a visualisation of the algorithm associated with the topic *Warshall* discussed in Chapter 4 as it appears in the TM defined in Part III :

```
{Warshall, Visualisation, [[ THIS—Reference: Figure 12.6.2 : Page 182 ]]}
```

9.5.4 Author

It is standard practise for scholars and students engaged in written academic conversations to cite their sources. Citations are used to provide information for the readers to locate the original source in order to learn more about the topic. When citing, a title, the date of publication and the name of the author are important elements. For this reason the author and date fields are required in all objects in the TM of A. An author should be specified whenever an object contains an in-line occurrence. As can be seen in the class diagram in Figure 9.1.1, this may be the case for almost any type of topic in the TM of A. The author may be entered as text, a bibliography item, or an internal occurrence.

Ideally the site hosting a TM based on the TM of A should provide a function to generate a citation for the currently shown object using the detail in these fields along with the display name of the object. Preferably such a function should support different citation formats and allow the user to select a preferred format.

When an author makes multiple contributions, it is preferable that a topic of type `Author` is created for the author. It may have links to any number of profile pages. A site hosting a TM extending the TM of A may provide a default author

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

profile page and allow authors to set up profiles on the site. Authors should be encouraged to set up their personal profiles in appropriate networks such as Google Scholar, the ACM digital library and LinkedIn. The following defines me as an author:

```
[VP : Author]
{VP, FullName, "Vreda Pieterse"}
{VP, Profile, "http://www.cs.up.ac.za/cs/vpieterse/"}
{VP, Profile, "http://www.linkedin.com/pub/vreda-pieterse/38/62/58b"}
{VP, Profile, "http://dl.acm.org/author_page.cfm?id=81100481696"}
{VP, Profile, "http://www.researchgate.net/profile/Vreda_Pieterse"}
{VP, Profile, "http://scholar.google.co.za
     /citations?user=CeJSo8oAAAAJ&hl=en"}
```

Once such a topic exists in the TM, it can be used as a reference to the author. The following defines `Coat` as an algorithmic technique authored by me as it appears in the TM defined in Part III of this thesis:

```
[Coat : Technique ="Coat"]
{Coat, Definition,
    [[Construct the TC of a relation by coating the relation with additional
      edges until it is transitive]]}
{Coat, Author, VP}
{Coat, Date, [[ 2013-05-14 ]]}
```

If an author is not defined, the full author name may be defined in-line. The following is an example of a definition as it might appear in a TM of tree acceptance algorithms. The definition as well as its author is specified in-line:

```
[T-ACCEPTOR : Technique ="T-ACCEPTOR"]
{T-ACCEPTOR, Definition,
    [[Use a tree automaton accepting the language of an RTG]]}
{T-ACCEPTOR, Author, [[Loek Cleophas]]}
```

If an in-line occurrence is taken from a publication, the publication can be specified as an author occurrence. The following is an example of a definition taken from a publication:

```
{Algorithm, Definition,
    [[A well-ordered collection of unambiguous and effectively computable
     operations that, when executed, produces a result and halts in a
     finite amount of time.]]}
{Algorithm, Author, [[ THIS-Citation: Schneider et al. [216]]]}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

9.6 Summary

This chapter is the culmination of the extensive reading and deliberation about attributes of algorithms that was reported in the previous two chapters. The definitions and descriptions in this chapter represent the core of the design science research presented in this thesis. Concepts for characterising algorithms are identified and described. Attributes that are essential and those that should be supported are specified for each of the identified concepts. Relationships between these concepts are determined and the multiplicity of each relationship is specified. The identified concepts are listed in Section 9.1 while the relationships between them are specified in UML notation in Figure 9.1.1 and specified in the TM of A. The LTM listing of their specification is shown in Listing 9.2.1. The completion of some of these definitions prompted initiatives to ensure that the required information is included in Wikipedia and in DADS, since these resources were identified as authoritative resources for the TM of A specified in this chapter.

The remainder of the chapter deals with each of the concepts separately. In each subsection devoted to a specific concept, the inclusion of the concept in the TM of A is briefly justified and the topics of its specification in the TM of A is described and specified. The LTM Listings of the specification of some of the attributes are shown as representative examples, while the rest of the definitions are available for perusal in the Appendix of this thesis. Practical examples are given to illustrate how topics of the type under consideration in the section could be included in a TM of algorithms that extends the core TM of A as defined in this chapter. While some of these examples are taken from the TM of TCA described in Part III of this thesis, other examples use different content. One such example is the specification of a compound algorithm and its sub-algorithms as it may appear in a topic map of algorithms for constructing Minimal Acyclic Deterministic Finite Automata (MADFA). These algorithms were classified by Watson [254]. Another example is the specification of algorithms that were classified by Cleophas [60] and implemented by Strolenberg [233].

The next chapter is a description and illustration of the process one should apply to add an algorithm to a repository of algorithms to extend the TM of A specified in this chapter.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 10

Process model

This chapter describes a process model for specifying TMs of algorithms. It forms part of a design science research methodology as described by Peffers *et al.* [190]. It is one of the research deliverables recommended to support the demonstration of an artefact produced using this research methodology. A core TM described in Chapter 9 is the primary artefact delivered in this research. This chapter describes a process model for populating a TM of algorithms as an extension of this core TM. It is thus a description of *how* to create and maintain a TM of algorithms.

In Section 10.1 the activities are defined and the order in which they should be performed here is specified. These activities are described in more detail in the remainder of this chapter. The activities in the process model should be executed by an actor. The actor executing the process described in the model proposed in this chapter is a person who extends the TM of A that is defined in Chapter 9. In the remainder of this chapter this person is called *the author*. This should not be interpreted to mean that the TM is authored by one person. Different people may act as the author at different times in different sections of the TM of A.

10.1 Activities

Figure 10.1.1 is a UML activity diagram showing the process to add and maintain information about an algorithm in this TM. The activities for identifying an algorithm, finding its position in the derivation tree, and verifying it correctness are shown as actions that may be performed in parallel. This, however, does not mean that they have to be completed concurrently. It merely means that the order in which they need to be achieved is not specified. They may be completed in any order. The only requirement is that all three of these activities need to be completed before proceeding with the steps to add the algorithm to the TM.

The activities to add information to the TM requires that there should be an object in the TM to represent the algorithm. This objects has to be created. Thereafter information about the algorithm can be specified through the specification of associations between the algorithm and other objects in the TM and by specifying

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

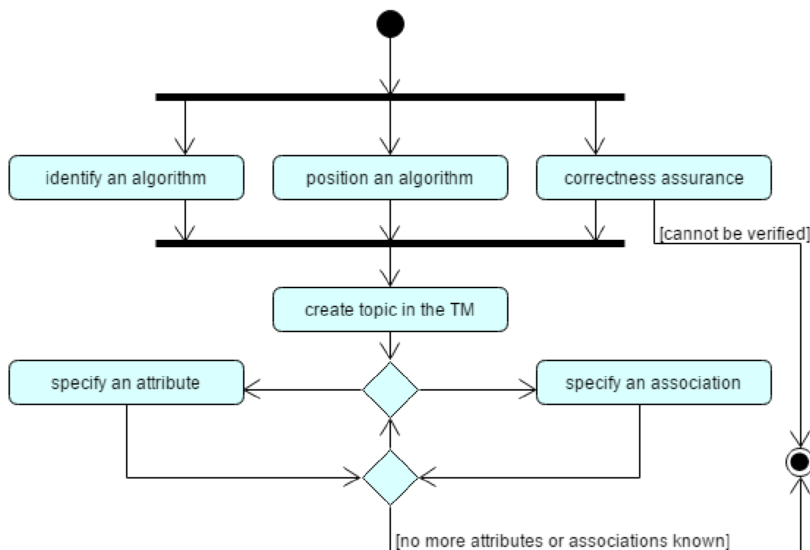


Figure 10.1.1: The process to add an algorithm to the topic map

attributes of the algorithm. The activities of specifying associations and attributes may be performed in any order and as many times as needed.

Both the transitions to the final node shown in the diagram should be interpreted as temporary termination. Ideally the process should not continue if the correctness cannot be verified. As soon as a satisfactory verification is established, the author may resume the process. Likewise if no more attributes or associations are known the author may terminate the process. It is, however, assumed that the author may resume at any time later to update more attributes or add more associations.

10.2 Identify an algorithm

The first step in the process, shown in Figure 10.1.1, is to identify an algorithm that has to be added to the TM. The discovery of an algorithm can be the consequence of one of the following events:

- An algorithm that is possibly not yet included in this TM is found in the literature.
- Someone designed a new algorithm and wishes to add it to the TM.
- A lack of symmetry in the derivation tree of algorithms is observed, indicating a gap in the derivation tree.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

In the first two cases the algorithm needs to be analysed as described in Section 10.3 in order to decide where to include it in a derivation tree.

The position of the algorithm in the derivation tree should depict a derivation path showing how the algorithm can be constructed through correctness preserving transformations of an existing algorithm.

In the third case the position of the discovered algorithm is known, rendering the activity to find the position of the algorithm in the derivation hierarchy trivial. Examples of algorithms that were discovered in this way in the case study in Part III, can be seen in Sections 14.2 and 15.2.

10.3 Position an algorithm

This section explains the reasoning required to determine an optimal position of a given algorithm in a derivation hierarchy of algorithms. First the reason for requiring this step is highlighted and the correlation between the application of algorithmic techniques and a derivation hierarchy is shown. Thereafter the positioning of algorithms in the context of various scenarios is discussed.

10.3.1 Rationale

Algorithms can be classified according to multiple facets. These facets may involve any of the attributes of algorithms shown in Table 9.3.1. The facets may also involve other objects that are related to algorithms in this TM, such as the data structures used by the algorithms, the problems solved by the algorithms and the algorithmic techniques applied by the algorithms. The positioning of an algorithm in a derivation hierarchy is a classification facet that involves only the algorithmic techniques applied by the algorithm.

This classification facet is a crucial aspect of the TM. It is required in order to support the verification of the correctness of the algorithms in the TM. The verification of the correctness of algorithms depends on how the algorithm relates to the other algorithms in the derivation hierarchy that is formed by this classification facet. Correctness verification is discussed in more detail in Section 10.4.

The derivation hierarchy describes the similarities and differences of algorithms. Each derivation step corresponds with the application of an algorithmic technique. Techniques involve adding enriching properties to the algorithm and/or refining the algorithm. The derivation hierarchy so formed is similar to how algorithms are ordered in the taxonomies created by Broy [43], Cleophas [60], Watson [253] and others. These taxonomies are discussed in Section 7.3.1.

When positioning a new algorithm in an existing derivation hierarchy in the TM, the goal is to maximise re-use of existing derivation paths in the hierarchy. One would start with a careful analysis of the new algorithm to identify the algorithmic techniques applied by the algorithm. Once the algorithmic techniques have been identified, the algorithms in the TM that apply the same algorithmic techniques may be considered as possible parents of the new algorithm in the derivation tree.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

An example of a derivation path of an algorithm that applies algorithmic techniques $T0, T1, T2$ and $T3$ is shown in Figure 10.3.1. The top node is the root algorithm. The node at the bottom represents the algorithm that can be derived from the root algorithm by applying algorithmic techniques $T0, T1, T2$ and $T3$ in this order.

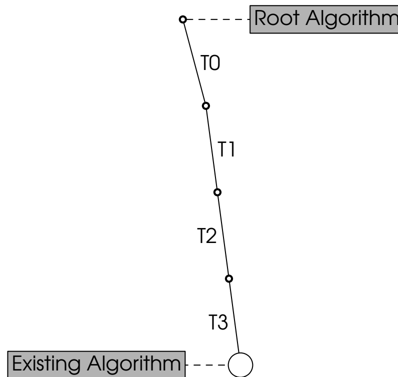


Figure 10.3.1: A derivation path

10.3.2 Starting a derivation tree

A derivation tree is started by specifying a root algorithm. It is an abstract naïve algorithm that theoretically solves the problem without the application of any particular algorithmic techniques.

For a root algorithm the activities of identifying the algorithm, finding its position in the derivation tree, and verifying its correctness shown in Figure 10.1.1 are all trivial. The root algorithm is defined to be an algorithm that solves a problem through the application of the definition of the problem without specifying detail of *how* to solve the problem. Its position is the top node of the derivation tree. It is trivially correct.

The actions to add a root algorithm thus starts with the creation of its algorithmic object in the TM as discussed in Section 10.5.

The specification and creation of the root algorithm of the case study in Part III is discussed in Section 11.2.

10.3.3 Starting a branch

If a new algorithm applies one or more algorithmic techniques, yet none of these algorithmic techniques are shared with other algorithms in the derivation tree, the author has to create a new branch in the derivation tree and show how the new algorithm can be derived from the root algorithm by applying these techniques in an appropriate order.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 11.3.1 (Coat) and 11.4.1 (Grow) are examples of algorithms that were added by starting new branches extending from a root algorithm in the case study in Part III. These algorithms are discussed in Chapter 11.

10.3.4 Extending a branch

If there are algorithms in the TM that apply some of the algorithmic techniques applied by an algorithm one wishes to add to the TM, one would consider these algorithms one by one as possible parents of the new algorithm in the derivation tree. Selecting a suitable candidate is often a case of intuition. If the new algorithm can be derived from a selected candidate, a suitable parent for the new algorithm is found in the derivation tree.

A candidate is chosen by following a path in the existing derivation tree where each step represents the application of one of the techniques applied by the new algorithm. If an algorithm is reached where no derivation that applies one of the remaining techniques applied by the new algorithm exists, the algorithm at this point in the path may be selected as a candidate to serve as the parent of the new algorithm in the derivation tree.

If the chosen candidate appears as a leaf node in the derivation tree, it has to be shown that the new algorithm can be derived from that algorithm by applying the remaining techniques in an appropriate order. The new algorithm can be positioned in the derivation tree on a branch extending from this chosen candidate.

Figure 10.3.4 shows an example where a new algorithm applies $\{T0, T1, T2, T3\}$ and the chosen candidate appears as a leaf node that is derived from the root algorithm by applying $T0, T1$, and $T2$. In this case the new algorithm can be positioned as a child of the chosen candidate if it can be shown that correctness of the solution is preserved if technique $T3$ is applied to the chosen candidate.

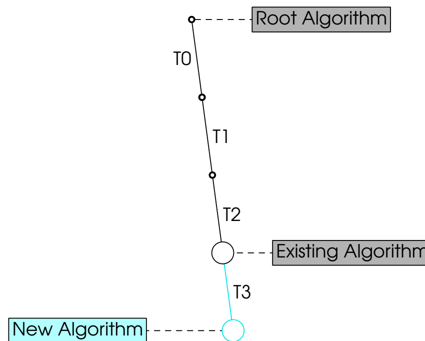


Figure 10.3.4: An algorithm derived from an existing algorithm

An example of extending a branch in the case study in Part III is the Algorithm 12.4.1 (Prosser) discussed in Section 12.4 that is derived from Algorithm 11.3.1 (Coat) through the application of two algorithmic techniques namely *Save upper bound* and *Natural*.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

10.3.5 Starting a sub-branch

If the chosen candidate is not a leaf node, sub-branches have to be formed. The new algorithm can be positioned in the derivation tree on a sub-branch extending from the chosen algorithm. Figure 10.3.5 illustrates an example where a new algorithm is derived from an abstraction that appears on the derivation path of an existing algorithm. Here the new algorithm applies the algorithmic techniques $\{T0, T1, T2, T4\}$, while an existing algorithm applies the algorithmic techniques $\{T0, T1, T2, T3\}$. The candidate that is chosen as the possible parent for the new algorithm is the algorithm that is derived from the root algorithm by applying the algorithmic techniques $T0, T1$ and $T2$. In this case the new algorithm can be positioned as a child of the chosen candidate if it can be shown that correctness of the solution is preserved if technique $T4$ is applied to the candidate instead of $T3$, as is the case with the existing algorithm.

An example of creating a fork in the case study in Part III is where Algorithm 14.1.1 (Baker) is added to the TM in Chapter 14. It is derived from an abstraction that appears as the parent of Algorithm 13.2.1 (Martynyuk) in the derivation tree. It applies the *Change monitor* technique instead of *Save upper bound* technique.

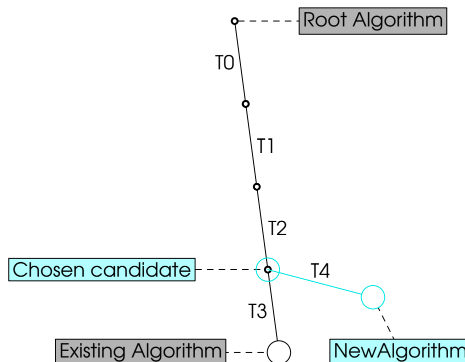


Figure 10.3.5: An algorithm derived from an abstraction on the derivation path

10.3.6 Alternative derivation paths

It is possible that a new algorithm can be seen as an alternative abstraction of an existing algorithm. Figure 10.3.6 shows an example of such a case. Here the new algorithm applies $\{T0, T1, T4\}$. The candidate that is chosen as a possible parent for this new algorithm is the algorithm that is derived from the root algorithm by applying $T0$ and $T1$. The new algorithm may be positioned as a child of the chosen candidate if it can be shown that correctness of the solution is preserved if technique $T4$ is applied to the candidate instead of $T2$.

The existing algorithm that is derived from the chosen candidate applies $\{T0, T1, T2, T3\}$. The red dotted line in the derivation tree in Figure 10.3.6 indicates

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the possibility of the existence of an alternate derivation of this existing algorithm.

If it can be shown that the existing algorithm can be derived from the new algorithm by applying say *T*6, the existing algorithm can be derived from the root algorithm in two different ways.

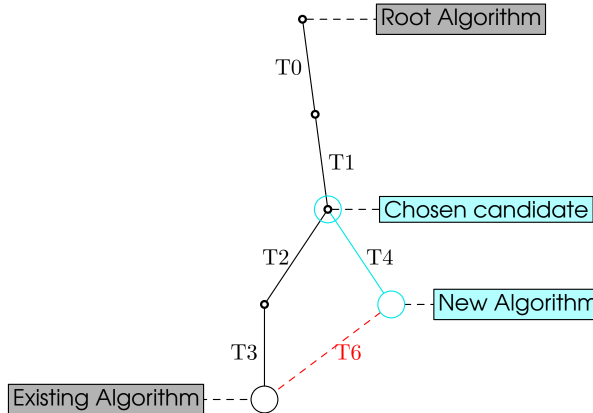


Figure 10.3.6: A new algorithm that is an alternative abstraction of a new algorithm

An example of an algorithm with alternative derivation paths in the case study in Part III is where Algorithm 16.4.1 (TileSkeleton) is added to the derivation hierarchy. Algorithm 12.6.1 (Warshall) is derived from Algorithm 12.5.1 (MatrixGrow) by applying *Natural* and Algorithm 16.4.1 (TileSkeleton) can be derived from Algorithm 12.5.1 (MatrixGrow) by applying *Loop tiling*. It was, however, realised that Algorithm 16.4.1 (TileSkeleton) can be an alternative abstraction of Algorithm 12.6.1 (Warshall). Algorithm 12.6.1 (Warshall) can be derived from Algorithm 16.4.1 (TileSkeleton) by applying the *Trivial* tiling strategy as algorithmic technique.

10.4 Correctness assurance

The creation of the derivation hierarchy prescribed here is similar to the process described by Jonkers [130] which was mentioned in Section 7.2.1 and used by others such as Broy [43], van de Rijdt [243] and Cleophas *et al.* [59]. A succulent description of this method can be found in Chapter 3 of Watson's [253] thesis.

When a new algorithm is added to the TM of A, the new algorithm is described in terms of a derivation of an algorithm which is already in the TM being developed, which has therefore already been validated. When doing so, it is only necessary to provide a convincing argument that the derivation preserves the correctness of the validated algorithm. The correctness of the newly added algorithm then follows from the correctness of its parent and a correctness preserving derivation.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

As shown in Figure 10.1.1 only algorithms that are believed to be correct should be added to a TM. The purpose of the TM that is created will determine the level of rigour required for the verification.

10.5 Create a topic in the TM

After completing the steps described in the above sections, most of the information about a new algorithm is known. The author should now be able to add this information to the TM. This is done by creating an algorithm object that adheres to the structure described in Section 9.3.1.

When creating an algorithm object, the author chooses an appropriate topic identifier (TI) and specifies the name as well as name variants if needed for the algorithm object.

The algorithm is specified in a way to honour its position in the derivation hierarchy. This means that the definition of the algorithm should include the fact that it inherits from the chosen algorithm that serves as its parent in the derivation tree. As prescribed in Table 9.3.1 each algorithm has to have a topic type (TT) specifying whether the algorithm is a primitive or compound algorithm.

The following is the declaration of Warshall's [252] algorithm to calculate the TC of a binary relation as it appears in the case study in Part III. The algorithm is discussed in Section 12.6:

```
[Warshall : GrowM-Op] /* Parent in the derivation tree */
[Warshall : PrimAlg ="Warshall";"warshall";"Warshall's TC algorithm"
     ("Warshall−Floyed" /alsoKnownAs)
     ("Algorithm 96: Ancestor" /alsoKnownAs)]
```

After creating an algorithm object, the attributes of the algorithm should be identified and added as described in Section 10.6. Associations that it may have with other objects should be specified as described in Section 10.7.

10.6 Specify algorithm attributes

The attributes of the algorithm describe the algorithm. They form the main content of the TM of algorithms. Users of the TM will rely on these facts when using the TM.

As indicated in Table 9.3.1, some of the attributes are required, while others are optional. When adding an algorithm to the TM the required attributes have to be added while the optional attributes may be omitted. Additional attributes may be added at any time.

Each attribute is included in the TM using the syntax described in Expression 2.3 (Section 2.2.5, Page 28). Attributes should be specified as prescribed in Table 9.3.1. The permissible occurrence types for each of the attributes are indicated in the table. The steps to update an attribute are determined by its type. The occurrence types of attributes may be *textual, external* or *internal* as discussed in the following subsections.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

10.6.1 Textual attributes

Textual attributes are those that are required to be defined in-line in the TM. An attribute of which the occurrence type is indicated as *Text* in Table 9.3.1 are called textual attributes. The occurrence of the attribute should be a word or a short sentence. The text must be enclosed in double square brackets. This information is stored in the TM.

The following is an example of a textual attribute as it appears in the case study in Part III. It is the specification of the *Description* attribute of the example topic that was specified in Section 10.5.

```
{Warshall, Description,
  [[ Derivation of the grow algorithm using matrix operations. It iterates
    through the entries in the matrix in column order ]]}
```

10.6.2 External attributes

External attributes are attributes those for which the occurrence is external to the TM. Such occurrence is a resource containing the information about the attribute. The occurrence is defined by using the URI of that resource. Each such occurrence links the topic (in this case the algorithm object) in the TM to resources that can exist anywhere in the world. The use of external attributes is a powerful feature of TMs which simplifies the incorporation of information in the TM while the information is created and maintained independent of the TM.

Ideally existing information should be linked and used rather than recreated. In many cases, however, the required information has to be created by the author and published on the Internet in a way that it can be accessed by the TM.

The following are examples of external attributes specified in the TM. Both are declarations of discussions of the example topic that was specified in Section 10.5. The first of these attributes points to a web page containing lecture notes about Warshall's algorithm while the second one points to the discussion of this algorithm in this thesis.

```
{Warshall, Discussion,
    "http://faculty.simpson.edu/lydia.sinapova/www/cmsc250
    /LN250_Tremblay/L17−Warshall.htm" }
{Warshall, Discussion, [[ THIS−Reference: Section 12.6 : Page 180 ]]}
```

10.6.3 Internal attributes

An internal attribute is an attribute for which there is a need to define attributes of its own and possibly make assertions about it in the TM. In the TM of A occurrences represent the attributes of topics. The TM standard restricts occurrences to be either textual or external. To accommodate the definition of attributes that are topics themselves, the concept of an internal occurrence is introduced. This extension

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

is justified and explained in Section 2.2.6. The extension allows the use of a topic identifier as an occurrence as a shorthand for specifying a **HAS-A** relation with the topic that correlates with the specific occurrence type.

The TM designed in Chapter 9 includes specifications for many topics that are suited occurrences for identified occurrence types.

The following are examples of attributes that have topics as occurrence types. As they are topics, it is easy to add information about these attributes in the TM. These examples are declarations of implementations of the example topic that was specified in Section 10.5.

```
{Warshall, Implementation, WarshallImpStd }
{Warshall, Implementation, WarshallImpBoost}
```

If the object containing the required information about the attribute is not yet included in the TM, it has to be created in the same manner that an algorithm object is created. The same as for algorithms, the required and optional attributes for that object should adhere to the specifications for objects of its kind in the metadata specification in Chapter 9. The specification of the topics that appear as internal occurrences in the above example are shown in Listing C44.

10.7 Specify associations

Associations between concepts provide a basic level of formal semantics. The topic map standard provides a construct called the topic association to support this [193].

The ability to describe relationships between topics elevates a topic map to being a semantic network. The knowledge that is represented in terms of associations in a TM can be applied to infer new knowledge. For this reason authors are encouraged to specify as many relations as possible between topics in a TM.

Each algorithm is either a root algorithm or derived from a root algorithm. As explained in Section 10.3.2, the root algorithm is closely related to the problem solved by the algorithm. This association should therefore be defined between the root algorithm and the problem it solves. By doing this each algorithm that is derived from the root algorithm stands by implication in the same relation with the specified problem.

The following term is the declaration of the root algorithm in the case study in Part III followed by the declaration of the *solves* relation between this algorithm and the TC Problem.

```
[TCRoot : PrimAlg ="TC Root Algorithm"]
solves (TCRoot : Algorithm, TCProblem : Problem)
```

The parent of the algorithm in a derivation tree of algorithms as well as the algorithmic techniques applied by the algorithm are determined in Section 10.3. When an algorithm is added to the TM, its parent is specified.

A new algorithm inherits the associations of its parent in hierarchical structure that is created to maintain the derivation tree.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Owing to the close relation between algorithms and their underlying data structures it is likely that the author gained complete cognisance of the data structures used by the algorithm while analysing the algorithm as described in Section 10.3. It is also likely that the author is aware of other items in the TM that can be associated with the algorithm. Additional relations may be added at any time later.

When adding an association of which an object that plays a role in the association is not yet included in the TM, the object has to be created in the same manner an algorithm is created as described in Section 10.5

10.8 Summary

This chapter explains and justifies the process steps an author should follow to extend the TM that is described in Chapter 9 in order to create a knowledge repository of algorithms. The steps are illustrated with generic examples as well as specific examples from the TM that is constructed in Part III of this thesis.

The actions required to add information to the TM of A requires that the particular facts have to be specified. The work required to acquire these facts is not always straight forward. In particular, when one wishes to add an algorithm to the TM, the process of finding the position of this algorithm in the derivation tree often has to rely on intuition. Section 10.3 gives guidelines that can be followed by an algorithm taxonomist to perform this challenging and creative task. Many taxonomists have done this in practice before and the theoretical foundation of this process is described in the literature (see Sections 7.2 and 10.4), but to my knowledge practical guidelines of this nature have not been outlined before. These guidelines suggest that careful analysis of the algorithmic techniques performed by an algorithm provide a key to guide the taxonomist when performing this task. All possible configurations of the set of algorithmic techniques performed by an algorithm that is to be positioned in the derivation tree in relation to the set of algorithmic techniques performed by algorithms that are already part of the derivation tree are considered. For each configuration, a process is suggested to find the ideal position of the algorithm in the derivation tree.

The process is defined and described in abstract terms. These abstract steps are explained by pointing to practical examples of their execution in next part of this thesis. In the next part of the thesis, these steps are applied to the TM that is described in Chapter 9 to develop a TM of transitive closure algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part III

Transitive Closure Algorithms

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Overview of Transitive Closure Algorithms

This part of the thesis describes the creation of a Topic Map (TM) of algorithms that solve the transitive closure (TC) problem. The TM extends the generic TM of algorithms, called the *TM of A*, that is constructed in Chapter 9. This extension is called the TM of TC Algorithms. The abbreviation *TM of TCA* is used to refer to it.

The main aim is to document existing algorithms. These algorithms are analysed, classified, described and implemented and then publicised as a TM. The academic work involves gathering and describing information about algorithms in a principled manner. As part of this exercise, a derivation tree of proven correct algorithms is constructed. A side-effect of the construction of such a derivation tree is the discovery of new algorithms. New algorithms in this sense are algorithms that have not been described in the literature before. They may have been used in practice before. It is not a requirement that they have to be salient or have significant practical implications.

The creation of this TM illustrates the practical use of the metadata that was designed in Chapter 9, confirms the viability of the process that was specified in Chapter 10 and shows the feasibility of using a TM to publicise algorithmic information.

The information about TC Algorithms described in the TM of TCA developed in this part is limited to the description of algorithms that solve the transitive closure problem for binary relations and that are represented using adjacency matrices. The requirement that the representation model for these algorithms should be adjacency matrices limits the relations that can be processed to finite relations. This is because the transformation to represent a relation using its adjacency matrix assumes that the relation is finite.

Algorithms that use directed graphs or use successor and/or predecessor lists as their data models are excluded. With the exception of two algorithms by Agrawal and Jagadish [2], all the algorithms presented in this part have been designed to use square Boolean matrices as their data model. The algorithms by Agrawal and Jagadish were designed to use successor lists. The discussion of these algorithms in Sections 17.3 and 17.5, however, provides transformed versions of these algorithms. They are described and implemented using square Boolean matrices as their data model.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 11

Root algorithms

To comply with the specifications for a topic map (TM) of algorithms as proposed in Chapter 9, it is required that each algorithm be associated with the problem that it solves, which in turn has to be associated with a problem area. The problem that is solved by the algorithms discussed in this part as well as its area are defined in Chapter 9 as part of the TM of algorithms. Section 11.1 refer the reader to these definitions. They are the starting point for the development of the TM that is the subject of this part, namely the TM of transitive closure algorithms (TM of TCA).

One of the facets for the classification of algorithms in the TM of TCA is the derivation tree. It is a hierarchical structure showing how algorithms relate to one another in terms of abstraction, refinement and enrichment. Section 11.2 defines a root algorithm that serves as the parent of all transitive closure algorithms in this derivation tree. It is the most abstract definition of an algorithm to solve the transitive closure problem.

Two basic techniques that can be applied to refine this root algorithm are discussed in Section 11.2.3. Based on these techniques, the root algorithms for two branches of the derivation tree are defined in Section 11.3 and Section 11.4.

11.1 Problem and problem area

The TC problem, the root TC Algorithm and the problem area of Graph Problems are the base topics of the TM that is developed in this part. The problem area of *Graph Problems* is defined in Section 9.4.1 with its topic specification using LTM notation in Listing C2. The definition of the problem *Transitive Closure* can be found in Section 9.4.2 and an occurrence as well as its association with the area of Graph Problems using LTM notation is given in Listing C5.

As explained in Section 2.2.4, an association is a relationship between two or more topics. The topics that participate in an association are called players in the association. Each player plays a role in the association. The role of a participant in an association is determined by the type of such player. If an association is defined between two topics in a TM, the association has to be bidirectional. Thus the relation that is defined between the TC problem and the TC Root Algorithm is

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

bidirectional. The single relation should be interpreted to have one meaning that can be expressed either in active voice or passive voice:

The TC Root Algorithm solves the Transitive Closure problem

The Transitive Closure problem is solved by the TC Root Algorithm

Natural language usually do not have words to describe bidirectional associations, it is customary in TMs to label relations with both alternatives. Technically this relation between the TC problem and the TC Root Algorithm should be labeled *solved/is solved by*. To reduce clutter in the diagrams in this thesis, the associations are named with one-directional verbs. Despite the use of one-directional verbs to identify the associations, all associations should be interpreted to be bidirectional as required in TMs.

Figure 11.1.1 visualises the three base topics of the TM defined so far as well as the associations between them. Topics are shown as circles. Tags are used to indicate the topic names while the text in the circles indicate the topic types. The topic types are also the roles that these topics play in the associations between them. Lines connecting topics indicate associations. The associations are labelled with the association names. An arrow is used to indicate the direction of the chosen name of the association. The reader should interpret it to be bi-directional by converting the sentence to passive voice for the inverse of the relation.

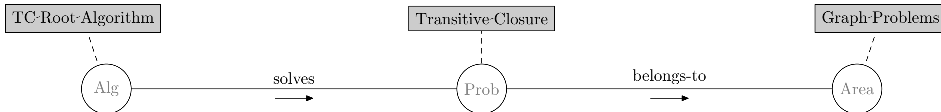


Figure 11.1.1: The TCRoot and TCProblem topics

11.2 Starting the derivation tree of TC algorithms

The base TC algorithm shown in Figure 11.1.1 is associated with the TC problem specified in Listing C5. As such it assumes the precondition and postcondition specified for the problem.

11.2.1 Defining transitive closure

The transitive closure R^+ of a relation R can be defined in several ways. In Section 6.5.1 R^+ is defined as the least fixpoint of $f.X = R \cup X \circ R$. This fixpoint defines R^+ as the least relation that is transitive and contains R. This definition correlates with the intuitive definition of transitive closure that was offered in Section 1.7. The following is the formal expression of this definition of TC:

$$R^+ = R \cup R^+ \circ R.$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The trivial yet unrealistic algorithm for constructing the transitive closure of a relation is specified in Algorithm 11.2.2 (Root). It simply assigns the solution according to the above definition of TC. This algorithm does not have an implementation because the detail of *how* to calculate the transitive closure is not specified in the definition. The specification of this algorithm as a topic in the TM of TCA using LTM notation is shown in Listing C12.

As discussed in Section 2.3.2 the specification of constants is used as a method to specify preconditions for an algorithm. In Algorithm 11.2.2 (Root) the scope of the constant R is delineated in the expression $R \subseteq U \times U$. It specifies that the precondition is that R is a relation of $U \times U$.

11.2.2 The root TC algorithm

Algorithm 11.2.2: TC Root Algorithm

---

const $R \subseteq U \times U$
var $R_1 \subseteq U \times U$

$R_1 := R \cup (R_1 \circ R)$
$\{$**Post:** $R_1 = R^+\}$

11.2.3 Fundamental techniques of TC construction

The algorithms that solve the transitive closure problem for finite relations are classified here according to the fundamental method they use to construct the transitive closure of a given relation $R \subseteq U \times U$. Two such methods have been identified. The mathematical foundations of these methods are discussed in Sections 6.5.1 and 6.5.2.

One approach uses a function $f : (\mathcal{P}(U)) \mapsto (\mathcal{P}(U \times U))$ that is defined such that $f.\varnothing = R$ and $f.U = R^+$. This function should also be defined such that the value of $f.(A \cup \{u\})$ can be calculated if the value of $f.A$ is known. The algorithm starts with $A = \varnothing$ and systematically adds elements of U to A. After adding an element u to A, the value of $f.(A \cup \{u\})$ is calculated using the value of $f.A$. In each iteration the algorithm adds an element to A and to recalculate $f.A$ until the value of $f.U$, and consequently R^+, is found. I call this approach *grow* because the argument of the function is grown by adding additional elements one-by-one until this argument contains all of U. The following is a definition of this technique:

> ***Grow:*** *Construct the TC of R by growing the argument of a function f (that ultimately maps U to the TC of R) from the empty set to U.*

The base algorithm that applies this technique is described in Section 11.4.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Another approach starts with R and systematically adds edges to this relation until a stage is reached when the resulting relation is transitive. I call this approach *coat* because the given relation is coated with the necessary additional edges to turn it into the smallest possible transitive relation containing R. The algorithm that applies this technique is described in Section 11.3. The following is a definition of this technique:

> ***Coat:*** *Construct the TC of a relation by coating the relation with additional edges until it is transitive.*

Note that both these techniques add the same entities one-by-one to a given relation in order to construct the TC of the given relation. The difference in the techniques is the way in which each technique selects the entities it adds as well as how the loop invariant for each technique is specified.

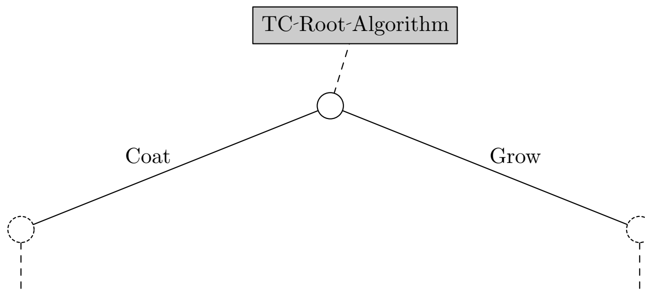


Figure 11.2.3: Basic derivation hierarchy of TC algorithms

The partial derivation tree showing the application of these algorithmic techniques is shown in Figure 11.2.3. This figure shows the location of two different abstract algorithms that are derived from the root algorithm. The one applies the grow technique while the other applies the coat technique. The topic specifications of these two base algorithmic techniques using LTM notation is given in Listing C7.

11.3 The coat technique

11.3.1 Determine the TC by applying the coat technique

The approach to determine the TC of R from R by coating it with additional edges stems from the application of an explicit form of the least fixpoint that was discussed in Section 6.5.1. The following explicit expression for the least fixpoint μf of $\langle \bigcup i : i \in \mathbb{N} : f^i.\varnothing \rangle$ is used:

$$R^+ = \langle \bigcup i : i \in \mathbb{N} : R \circ \langle \bigcup k : 0 \leq k \leq i : R^k \rangle \rangle \qquad (11.1)$$

An algorithm that applies this approach, determines R^+ by computing μf via a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

monotonic sequence that is the same as $\{f^n.\emptyset\}_n$. This approach thus computes R^+ via the calculation of a series Y_j for $j = 0, 1, ..$ where each Y_j is defined by

$$Y_j = R \circ \langle \bigcup k : k \in \mathbb{N}_j : R^k \rangle. \tag{11.2}$$

Figure 11.3.1 visualises the portion of the TM of TCA containing the topics in the TM that are directly related to this algorithm. Due to the nature of "is a", this configuration implies that the Root Coat Algorithm solves the same problem as its parent i.e. the Transitive Closure problem and also has the same preconditions and postconditions as its parent. The algorithm is specified in Algorithm 11.3.1 (Coat), the specification of the topic *Root Coat Algorithm* in the TM of TCA using LTM notation is given in Listing C13. Since precondition, namely that the input relation should be finite, is weaker than the precondition of the parent algorithm of this algorithm, it should be specified. In this case this is implied by having a constant n defined using the expression $n = |R| \in \mathbb{N}^+$.

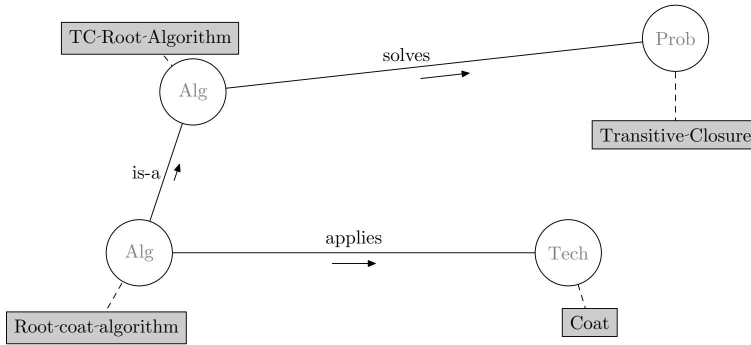


Figure 11.3.1: Topics related to the root coat algorithm

Algorithm 11.3.1: The root coat algorithm

---

const $R \subseteq U \times U$
$n = |R| \in \mathbb{N}^+$
var $R_0 \subseteq U \times U$
$R_1 \subseteq U \times U$

$R_0, R_1 := R, R$
do $R_1 \neq R^+ \quad \rightarrow$
$R_0 := R_0 \circ R;$
$R_1 := R_1 \cup R_0$
od
$\{$**Post:** $R_1 = R^+\}$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

11.3.2 Verification of the root coat algorithm

In iteration t the value of R_0 is the relation containing all edges that have to be part of the calculated relation to ensure that transitivity holds for all the elements of every path of length $m \leq t$ in R. At this point R_1 is coated with these edges by changing the value of R_1 to the union of R_0 and R_1. Lemmata 11.3.2 and 11.3.3 show how the first two values in the series defined in Equation 11.2 are calculated.

The application of Equation 11.2 produces the recurrence relation in Equation 11.3. Lemma 11.3.4 shows that Equation 11.3 holds for all $j \geq 0$ if it is assumed that Equation 11.2 holds. This equation enables the calculation of Y_j for arbitrary $j \geq 0$.

$$Y_{j+1} = Y_j \cup R \circ R^j \tag{11.3}$$

The following chain is now obtained:

$$\varnothing = Y_0 \subseteq Y_1 \subseteq Y_2 \subseteq \ldots \tag{11.4}$$

It is assumed that R is finite. Therefore, there must be an $m \in \mathbb{N}$ such that $Y_m = Y_{m+1} = \ldots$ Thus, at some stage the value of Y_j will coincide with the value of R^+.

$$\begin{aligned}
& Y_0 \\
= & \quad \{\text{Equation 11.2 with } j = 0\} \\
& R \circ \langle \bigcup k : k \in \varnothing : R^k \rangle \\
= & \quad \left\{ \begin{array}{l} \text{The domain of this quantified expression is empty, thus the} \\ \text{resulting value is the unit of } \bigcup \text{ which is } \varnothing \end{array} \right\} \\
& R \circ \varnothing \\
= & \quad \{\varnothing \text{ is the zero of } \circ\} \\
& \varnothing
\end{aligned}$$

Lemma 11.3.2: $Y_0 = \varnothing$

Algorithm 11.3.1 (Coat) implements Equation 11.3. Although the detail of how to calculate the transitive closure is specified, no data model for the representation of the relation is defined. It is also not known at what stage the value of R_1 reaches the required end value of R^+. Thus the condition for the algorithm to stop relies on knowing what the answer must be. This algorithm is deemed an abstraction of many concrete algorithms that may be derived from it. For this reason the TM need not have an implementation occurrence of this algorithm.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{aligned}
& Y_1 \\
= & \quad \{\text{Equation 11.2 with } j = 1\} \\
& R \circ \langle \bigcup k : k \in \{0\} : R^k \rangle \\
= & \quad \{\text{ The domain of this quantified expression is the singleton } k = 0\} \\
& R \circ R^0 \\
= & \quad \{\text{Definition of } R^0\} \\
& R \circ E_U \\
= & \quad \{E_U \text{ is the one of } \circ\} \\
& R
\end{aligned}
$$

Lemma 11.3.3: $Y_1 = R$

$$
\begin{aligned}
& Y_{j+1} \\
= & \quad \{\text{Equation 11.2}\} \\
& R \circ \langle \bigcup k : k \in \mathbb{N}_{j+1} : R^k \rangle \\
= & \quad \{\cup \text{ is associative}\} \\
& R \circ (\langle \bigcup k : k \in \mathbb{N}_j : R^k \rangle \cup R^j) \\
= & \quad \{\circ \text{ distributes over } \cup\} \\
& (R \circ \langle \bigcup k : k \in \mathbb{N}_j : R^k \rangle) \cup (R \circ R^j) \\
= & \quad \{\text{Equation 11.2}\} \\
& Y_j \cup R \circ R^j
\end{aligned}
$$

Lemma 11.3.4: $Y_{j+1} = Y_j \cup R \circ R^j$

11.4 The grow technique

11.4.1 Determine the TC by applying the grow technique

An alternative approach to calculate the transitive closure of a relation applies the following definition of R^+ that was obtained in Lemma 6.4.2:

$$R^+ = R \circ (E_U \circ R)^* \tag{11.5}$$

It relies on the following function:

$$h.A = R \circ (E_A \circ R)^* \tag{11.6}$$

In Section 6.5.2 it is shown that the function in Equation 11.6 can be applied to construct R^+ as it is clear that $h.U = R^+$ as defined in Equation 11.5

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following recursive expression of h provides a method to apply Equation 11.6. The correctness of this expression is shown in Lemma 6.4.3:

$$h.(A \cup \{u\}) = h.A \cup h.A \circ E_{\{u\}} \circ E_{\{u\}} \circ h.A \tag{11.7}$$

Recall that h is defined such that $h.\varnothing = R$. To compute R^+, one starts with $A = \varnothing$. The algorithm then iterates over the elements of U. In each iteration one element of U is added to A and the value of $h.A$ is updated through the application of the formula in Equation 11.7. The loop terminates when $A = U$. At this stage the value of $h.U$ is calculated. Per definition this value is R^+.

The specification of the root grow algorithm as a topic in the TM of TCA using LTM notation is given in Listing C14. Figure 11.4.1 visualises the portion of the TM containing the topics in the TM that are directly related to this algorithm. The algorithm is defined in Algorithm 11.4.1 (Grow).

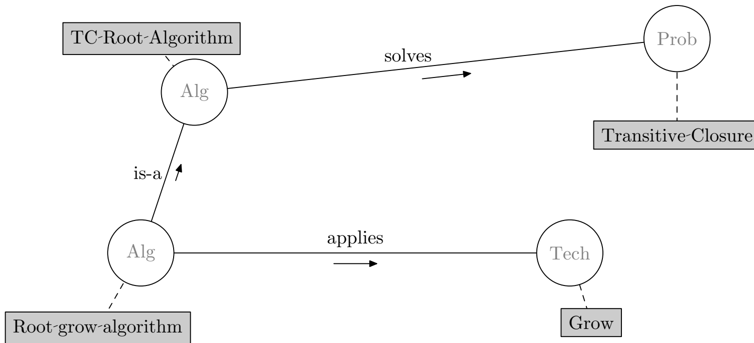


Figure 11.4.1: Topics related to the root grow algorithm

Algorithm 11.4.1: The root grow algorithm

---

const $R \subseteq U \times U$
var $R_0 \subseteq U \times U$

$R_0 := R$
for $u \in U \rightarrow$
  $R_0 := R_0 \cup (R_0 \circ E_{\{u\}} \circ E_{\{u\}} \circ R_0)$
rof
$\{$**Post:** $R_0 = R^+\}$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

11.4.2 Verification of the root grow algorithm

In Section 6.5.2 it was shown that the function in Equation 11.6 that is implemented by this algorithm converges to R^+. Note that the order of selection of the elements of U is non-deterministic. This algorithm is technically not an algorithm because U may be infinite and there is no guarantee that the TC will be reached in finite time.

11.5 Summary

This chapter describes the root algorithm (Algorithm 11.2.2 (Root)) as well as the top nodes in two major branches in a derivation tree of transitive closure algorithms. These algorithms (Algorithm 11.3.1 (Coat) and Algorithm 11.4.1 (Grow)) form the basis of a taxonomy of transitive closure algorithms in terms of a classification facet based on abstraction and refinement.

The further inclusion of these algorithms in the TM of TCA is achieved by meticulously following the process model described in Chapter 10. Each of the algorithms is defined using GCL. The position of each algorithm in the TM of TCA is determined in terms of its relation to other topics in the TM. The relevant topics and the relations between these topics are visualised. The definitions of the topics representing these algorithms in the constructed TM of TCA as well as the definitions of the related topics and the relations between them using LTM notation can be found in Appendix C.

An essential part of the the process that is followed when developing the TM of TCA is to ensure that every algorithm that is added to the TM has all its required attributes and is guaranteed to be correct. To this end the attributes are provided and the correctness of the root algorithm as well as the two algorithms that are derived from it in this chapter have correctness proofs. When failing this step, the correctness of the rest of the algorithms in the taxonomy is compromised.

The novel contributions in this chapter include the identification, naming and concise descriptions of the opposing techniques that are applied by the algorithms described in this chapter as well as the proofs of these algorithms. These proofs are the result of intensive mathematical reasoning based on the mathematical foundations discussed in Chapter 6. To my knowledge these proofs have not been published before.

In the following chapter concrete algorithms are derived from the two algorithms that are discussed in the current chapter, namely Algorithm 11.3.1 (Coat) and Algorithm 11.4.1 (Grow).

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 12

Using matrices

The algorithms discussed in this thesis operate on binary relations. To enable concrete implementations of algorithms, it is required that a representation model for binary relations is selected. Four different representation models for binary relations are commonly used. These are sets, graphs, lists and matrices. They are discussed in more detail in Section 3.5.

While many algorithms exist based on each of these representation models, the scope of this thesis is limited to algorithms whose definition assumes that matrices are used as the underlying data structure for the representation of binary relations. The correspondence between homogeneous binary relations and square Boolean matrices is established in Section 4.7. Section 12.1 recapitulates this correspondence and its consequences that are relevant to the TC algorithms that use matrices as representation model. This chapter describes how the algorithms discussed in Chapter 11 are transformed to basic concrete implementations using matrices as their data model for representing relations.

12.1 Adjacency matrices

The transformation of the algorithms defined in Chapter 11 to their concrete versions is established through the application of two isomorphisms that were developed in Section 4.7. These isomorphisms establish a 1-to-1 correspondence between homogeneous relations and two-dimensional Boolean matrices.

Let R be a binary relation with $R \subseteq U \times U$ and $n = |U|$. The following isomorphisms are used here to define a concrete representation model for relations:

$$\Phi : \mathcal{P}(U \times U) \mapsto \mathbb{B}[n,n] \tag{12.1}$$

$$\Psi : \mathbb{B}[n,n] \mapsto \mathcal{P}(U \times U). \tag{12.2}$$

Let R be a binary relation. $\Phi(R)$ as specified in Equation 12.1 maps to the square Boolean matrix representing R. $\Phi(R)$ is called the adjacency matrix of relation R. Let M be a square Boolean matrix. $\Psi(M)$ as specified in Equation 12.2 maps to the binary relation represented by M. M is called the adjacency matrix of the relation $\Psi(M)$. In Section 4.7.4 it was shown that Φ and Ψ are defined in

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

such a way that the behaviour of operations that are defined on square Boolean matrices correspond to operations on relations. In particular the following hold:

$$\Phi.(R_0 \circ R_1) = \Phi.R_0 \times \Phi.R_1 \tag{12.3}$$
$$\Phi.(R_0 \cup R_1) = \Phi.R_0 + \Phi.R_1 \tag{12.4}$$
$$\Psi.(M_0 \times M_1) = \Psi.M_0 \circ \Psi.M_1 \tag{12.5}$$
$$\Psi.(M_0 + M_1) = \Psi.M_0 \cup \Psi.M_1 \tag{12.6}$$

12.2 Transformation of the root coat algorithm

The transformation $\Phi : \mathcal{P}(U \times U) \mapsto \mathbb{B}[n,n]$ is applied to the data to be manipulated by Algorithm 11.3.1 (Coat). This transformed data serve as input to Algorithm 12.2.2 (MatrixCoat) discussed here. The output of this algorithm is a matrix representing the TC of its input relation. The transformation $\Psi : \mathbb{B}[n,n] \mapsto \mathcal{P}(U \times U)$ is applied to transform this result to the relation calculated by Algorithm 11.3.1 (Coat)

The matrix coat algorithm is a simple transformation of Algorithm 11.3.1 (Coat) where the relation on which it operates is represented using a square boolean matrix. As it has been established that the multiplication and addition of matrices respectively correspond with the composition and union of the binary relations, the matrix view of the relation Y_j, as defined in Equation 11.2, can be written as the series Z_j for $j = 0, 1, \ldots$ where each $Z_j = \Phi(Y_j)$ for $j \geq 1$ is defined by the following equation.

$$Z_j = M \times \langle \sum k : k \in \mathbb{N}_j : M^k \rangle. \tag{12.7}$$

By doing this, the following matrix view of the recurrence relation given in Equation 11.3 is obtained:

$$Z_{j+1} = Z_j + M \times M^j. \tag{12.8}$$

The relation expressed in Equation 12.8 can be applied to calculate Z_j for $j \geq 1$. The corresponding chain can be expressed as follows:

$$\mathbb{O}_n = Z_0 \sqsubseteq Z_1 \sqsubseteq \ldots \tag{12.9}$$

12.2.1 Description of the transformed the root coat algorithm

Algorithm 12.2.2 (MatrixCoat) defines the following three matrices.

$M_0 \in \mathbb{B}[n,n]$
: This matrix is initialised to M and maintains this value throughout the execution of the algorithm.

$M_1 \in \mathbb{B}[n,n]$
: This matrix is initialised to M and is multiplied with M_0 in each iteration. Thus $M_1 = M^t$ for progressive values of t.

$M_2 \in \mathbb{B}[n,n]$
: This matrix is also initialised to M. In each iteration its value is updated to $M_2 + M_1$. Therefore $M_2 = \langle \sum i : i \in \mathbb{N}_t : M^i \rangle$, thus assuming the value of the above defined Z_t for progressive values of t. Consequently, M_2 converges to M^+

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

These matrices are exactly the components of the recurrence relation expressed in Equation 12.8. The algorithm uses Equation 12.8 to calculate the consecutive values of Z_j in Equation 12.9 until the complete closure is calculated.

These topics and the relations between them are shown in Figure 12.2.1. Here *Matrix multiplication* and *Matrix addition* are defined as algorithms. They, however, play the role of techniques in the *applies* TM relations shown in this figure.

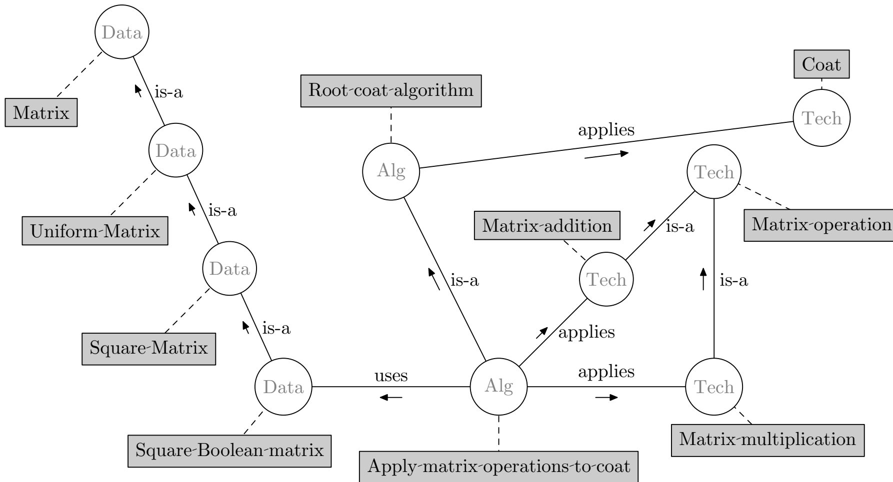


Figure 12.2.1: Topics related to a coat algorithm that applies matrix operations

Listing C15 is the specification of this algorithm as a topic in the TM using LTM notation. The data structure used by this algorithm is a square boolean matrix. The algorithmic technique used to determine the edges that have to be added to the relation in order to converge to transitivity is matrix multiplication. The algorithm also applies addition of square boolean matrices to add the identified edges. Listing C6 is the specification of this data structure and its generalisations as topics in the TM using LTM notation. The specification of the techniques used by the algorithm as topics in the TM using LTM notation are given in Listing C11. They are specified as types of a generic technique named *Matrix operations*. This algorithm is derived form the root coat algorithm. This algorithm has the same precondition and postcondition as its parent in this structure, namely that of the root coat algorithm. In addition, to be able to transform the relation to a Boolean matrix, the domain of the relation has to be finite. For this reason this weaker precondition is specified by specifying a constant n using the expression $n = |U| \in \mathbb{N}^+$.

12.2.2 Verification

Every step in the transformation of Algorithm 11.3.1 (Coat) to this algorithm preserves the correctness of each operation because of the properties of the func-

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 12.2.2: Coat algorithm that applies matrix operations

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^+$
var $M_0 \in \mathbb{B}[n, n]$
$M_1 \in \mathbb{B}[n, n]$
$M_2 \in \mathbb{B}[n, n]$

$M_0, M_1, M_2 := \Phi(R), \Phi(R), \Phi(R)$
do $M_2 \neq M^+ \rightarrow$
$M_1 := M_1 \times M_0$;
$M_2 := M_2 + M_1$
od
$\{$**Post:** $\Psi(M_2) = R^+\}$

tion Φ that was recapitulated in Section 12.1 and the 1-to-1 correspondence that it establishes between relations and Boolean matrices as shown in Section 4.7 assuming that the domain of the relation is finite. In particular, the chain in Equation 12.9 converges to $\Phi(R^+)$. The functions Φ and Ψ were defined in Section 4.7.4. Their duality follows from their definitions. In other words $\Psi.\Phi(R^+) = R^+$ and $\Phi.\Psi(M^+) = M^+$. This duality guarantees that when transforming the resulting matrix back to a relation using Ψ, the answer is R^+.

12.3 Matrix multiplication

The efficiency of different implementations of Algorithm 12.2.2 (MatrixCoat) is determined by the complexity an efficiency of the two operations that it performs, namely matrix addition and matrix multiplication. The naïve implementations of these operations are respectively $\Theta(n^2)$ and $\Theta(n^3)$. As multiplication has a higher order of complexity, the performance of Algorithm 12.2.2 (MatrixCoat) is thus determined by the algorithm that is used to multiply the matrices. When replacing a naïve matrix multiplication algorithm by faster algorithms for matrix multiplication, the overall performance may improve. Investigation of algorithms for matrix multiplication is a topic of its own. Such investigation has to include the analysis of the well known algorithm of Strassen [231]. This is an ingenious recursive algorithm that multiples two $n \times n$ matrices in $\Theta(n^{2.81})$ operations.

Other prominent algorithms for matrix multiplication are improvements of this algorithm by Bini *et al.* [30], Schonhage [220], Strassen [232], Pan [186, 187] that were published in the late 1970's and early 1980's. Widely cited work done by Coppersmith and Winograd [63, 64, 65, 66] contributed largely to this research domain during the 1990's which paved the way for authors like Jiang and Wu [128] and Pan [188] to further refine these ideas and for Eyas [88] to derive a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

parallel algorithm for matrix multiplication that can be executed on a cluster of workstations.

Further development of a branch of the TM to contain the different algorithms for matrix multiplication and alternative versions of Algorithm 12.2.2 (MatrixCoat) using them, is outside the scope of this thesis. Listing C3 defines this problem area as a topic in the TM using LTM notation.

12.4 Prosser's algorithm

This algorithm is historically the first published TC algorithm. Prosser [203] published the proof of its correctness in 1959 in the conference proceedings of the Eastern Joint Computer Conference that was held in Boston, the capital of the US state of Massachusetts.

Algorithm 12.4.1 (Prosser) is derived from Algorithm 12.2.2 (MatrixCoat). It applies a deterministic condition for terminating the loop. I call the algorithmic technique applied here to specify the termination of the loop *safe upper bound*. When this technique is applied, the condition for terminating a loop depends on a calculated value that can be determined before the loop is executed.

The topic specification of this technique is shown using LTM notation in Listing C8. The topic specification of this algorithm as a refinement of Algorithm 12.2.2 (MatrixCoat), is shown in Listing C17. Figure 12.4.1 shows a portion of the TM showing Prosser's algorithm and the topics related to this algorithm.

12.4.1 Description of Prosser's algorithm

Algorithm 12.4.1: Prosser's Algorithm

---

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^+$
var $M_0 \in \mathbb{B}[n, n]$
$M_1 \in \mathbb{B}[n, n]$
$M_2 \in \mathbb{B}[n, n]$

$M_0, M_1, M_2 := \Phi(R), \Phi(R), \Phi(R)$
for $i : i \in \mathbb{N}_{n-2} \quad \rightarrow$
$M_1 := M_1 \times M_0;$
$M_2 := M_2 + M_1$
rof
$\{$**Post:** $\Psi(M_2) = R^+\}$

Prosser [203] showed that the number of iterations needed to reach R^+ when executing this algorithm is determined by the length of the longest acyclic path in R. The value of the safe upper bound applied by this algorithm corresponds with the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

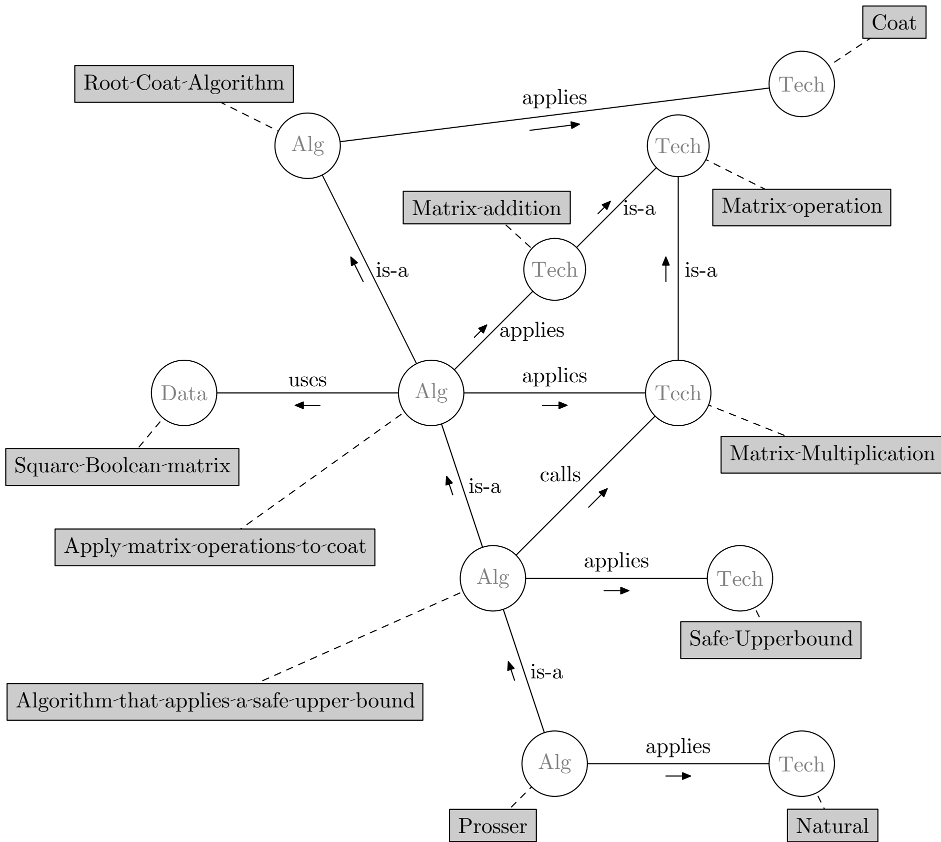


Figure 12.4.1: Topics related to Prosser's algorithm

length of the longest acyclic path in R. The order in which the elements of $\mathbb{N}_{n-2}$ is chosen for the execution of the loop in the definition of Algorithm 12.4.1 (Prosser) is non-deterministic. The operations in the body of this loop is, however, independent of the element that is chosen. Therefore different algorithms that rely on different deterministic ordering of elements yield equivalent results. The outer loop is merely a means of counting the number of iterations.

The implementation of the algorithm selects the elements of $\mathbb{N}_{n-2}$ for the execution of the loop in sequential order. The algorithmic technique to follow this natural order is called *Natural*. The specification of this technique as a topic in the TM using LTM notation is in Listing C8.

12.4.2 The derivation tree of Prosser's algorithm

Figure 12.4.2 shows that Prosser's algorithm can be derived from the root TC algorithm through the application of the fundamental coat technique; the use of a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

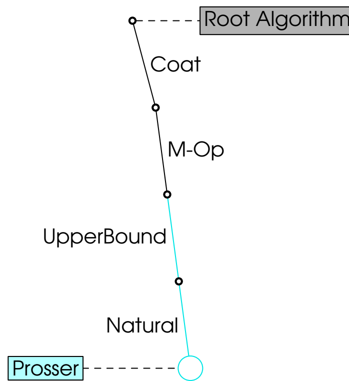


Figure 12.4.2: Derivation tree of Prosser's algorithm

matrix and appropriate matrix operations to manipulate the data; the application of a safe upper bound and performing the outer loop in natural order.

The definition of occurrences of implementations of Prosser's algorithm in the TM of TCA using LTM notation can be found in Listing C42. These point to implementations of this algorithm made available on *Aptī Algoritmi*[1], a website I have created for the purpose of hosting implementations of the algorithms in the TM of TCA constructed in this thesis.

12.4.3 Verification

Since the relation whose TC is to be calculated is finite and the entries of M_2 monotonically increase with respect to $\sqsubseteq$ in each iteration, it is evident that M_2 will, after a finite number of iterations, reach the value of M^+. Lemma 5.5.1 shows that for consecutive values of $t > 0$, each path of length t is represented by an entry in R^t and that each such entry is the direct connection of the head and the tail of this path.

The algorithm initiates $M_1 = R^1$. Thus, before iteration t, $M_1[i, j] \iff$ there is a path of length t from u_i to u_j in R. After iteration $t-1$, $M_2[i, j] \iff$ there is a path P from u_i to u_j in R with $\ell(P) \leq t$. In Section 5.4.2, we observed that if there is a path Q from u_i to u_j in R, there is a non-cyclic path P from u_i, to u_j in R with $\ell(P) \leq \ell(Q)$. When applying this observation it is clear that the number of iterations needed to reach M^+ is at most one less than the length of the longest non-cyclic path in R. Lemma 5.5.2 shows that the longest non-cyclic path in R is smaller than $|U|$, therefore, the number of iterations needed is maximally $|U| - 1 - 1 = n - 2$.

[1] `www.cs.up.ac.za/cs/vpieterse/AptiAlgo`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

12.4.4 Complexity

The complexity of Prosser's algorithm can be determined by examining the complexity of the outer loop of this algorithm and the number of iterations needed to calcualte the result. This loop contains two steps, one is the conjunction of Boolean matrices and the aothher the addition of Boolean matrices.

The complexity of conjunction of Boolean matrices, if carried out naïvely, is $\Theta(n^3)$. This is because there are $n \times n$ elements to calculate, and the value of each of these $n \times n$ elements is calculated in $\Theta(n)$ time. The complexity of the addition of two matrices is $\Theta(n^2)$. This is because, again there are $n \times n$ values to calculate, but these are each calculated in constant time.

The complexity of the body of this loop is thus $\Theta(n^3 + n^2) = \Theta(n^3)$. The loop is repeated sufficiently many times to ensure that the transitive closure is reached. It was established that the number of iterations needed is the length of the longest acyclic path in the relation, which is $n - 2$. Thus the complexity of executing this loop is $\Theta((n-2) \times n^3) = \Theta(n^4 - 2 \times n^3) = \Theta(n^4)$.

Listing C31 specifies this complexity of Prosser's algorithm in the TM using LTM notation. It points to the above paragraph as an occurrence of a justification of the complexity of this algorithm.

12.5 Transformation of the root grow algorithm

In Section 4.7 a transformation $\Phi : \mathcal{P}(U \times U) \mapsto \mathbb{B}[n,n]$ is constructed. It maps any $R \subseteq U \times U$ onto a square Boolean matrix $M \in B[n,n]$ where U is finite and $n = |U|$. It can be used to transform a given relation to a square Boolean matrix. Here this transformation is applied to transform the data of Algorithm 11.4.1 (Grow) to derive this algorithm. It uses the matrix views of the relations it manipulates. The transformation entails replacing the relations with their matrix views and then applying those matrix operations on these matrices that correspond with the operations on relations in Algorithm 11.4.1 (Grow).

The function in Equation 11.6 forms the essence of Algorithm 11.4.1 (Grow). Similarly this algorithm can be designed by implementing the following function:

$$f.A = M \times (\mathbb{I}_A \times M)^* \tag{12.10}$$

Similar to how the function in Equation 11.6 can be applied to construct R^+, this function is applied to compute M^+ using the following recursive expression that is equivalent to Equation 11.7:

$$f.(A \cup \{j\}) = f.A + f.A \times \mathbb{I}_{\{j\}} \times \mathbb{I}_{\{j\}} \times f.A \tag{12.11}$$

12.5.1 Description of the transformed the root grow algorithm

Listing C16 is the specification of this algorithm as a topic in the TM of TCA using LTM notation. The topics and relations that are defined in this listing are shown in Figure 12.5.1. Algorithm 12.5.1 (MatrixGrow) is essentially the same algorithm as Algorithm 11.4.1 (Grow). This algorithm has the same precondition

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

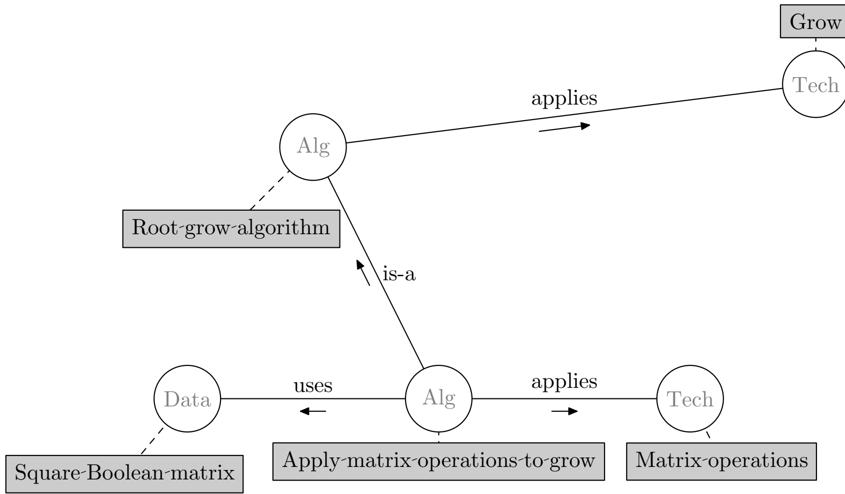


Figure 12.5.1: Topics related to a grow algorithm that applies matrix operations

---

Algorithm 12.5.1: Grow algorithm that applies matrix operations

---

const $R \subseteq U \times U$
$\quad\quad n = |U| \in \mathbb{N}^+$
var $M \in \mathbb{B}[n, n]$

$M := \Phi(R)$
for $j : j \in \mathbb{N}_n \quad \rightarrow$
$\quad M := M + (M \times \mathbb{I}_{\{j\}} \times \mathbb{I}_{\{j\}} \times M)$
rof
$\{$**Post:** $\Psi(M) = R^+\}$

---

and postcondition as its parent in this structure, namely that of the root grow algorithm. In addition, to be able to transform the relation to a Boolean matrix, the domain of the relation has to be finite. For this reason this weaker precondition is specified by specifying a constant n using the expression $n = |U| \in \mathbb{N}^+$.

The order of selection of the elements i and j of $\mathbb{N}_n$ in the loop is non-deterministic. It is, however, required that the body of the loop is processed for each value of $j \in \mathbb{N}_n$ before a new value for $i \in \mathbb{N}_n$ is selected. This requirement limits the possibilities for parallel processing that could be applied as a result of this non-determinism. In the next section a concrete implementation of this algorithm, assuming a natural order of selection is presented.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

12.5.2 Verification

In Section 4.7.4, function Φ was defined in such a way that the following holds.

$$\begin{aligned} \Phi.(R \circ S) &= \Phi.R \times \Phi.S \\ \Phi.(R \cup S) &= \Phi.R + \Phi.S \\ \Phi.E_{\{a\}} &= \mathbb{I}_{\{k\}} \qquad \text{where } \varepsilon.a = k \end{aligned}$$

Owing to these properties of Φ and the 1-to-1 correspondence that Φ establishes between relations and Boolean matrices as shown in Section 4.7, every step in the transformation of Algorithm 11.4.1 (Grow) to Algorithm 12.5.1 (MatrixGrow) preserves the correctness of each operation.

12.6 Warshall's algorithm

Warshall's algorithm is probably the best known algorithm for calculating the TC of a binary relation. It often appears in lecture notes and textbooks as the *de facto* standard algorithm for this operation. It is surprisingly simple and efficient. It was first published by Warshall [252] in January 1962. This publication contains a proof of its correctness. In June of the same year the algorithm was added to *The collected algorithms of the ACM (CALGO)* [115] as **Algorithm 96: Ancestor** authored by Floyd [92]. Floyd's publication only describes the algorithm. It has a single reference namely Warshall's publication of the proof of the algorithm.

12.6.1 Description of Warshall's algorithm

Algorithm 12.6.1 (Warshall) implements Algorithm 12.5.1 (MatrixGrow) using a deterministic order of selecting the elements in $\mathbb{N}_n$. It systematically calculates the value of each entry in M using a formula that is the equivalent of Equation 12.11. The algorithm sequentially iterates through the entries of M. The algorithmic technique to follow this natural order is called *Natural*.

Listing C19 is the specification of this algorithm as a topic in the TM of TCA. The specification of this technique is in Listing C8. In this case operations are performed in column major order. Figure 12.6.1 shows the topics related to this algorithm in the TM of TCA. The specification that points to occurrences of implementations of Warshall's algorithm in the TM of TCA can be found in Listing C44.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

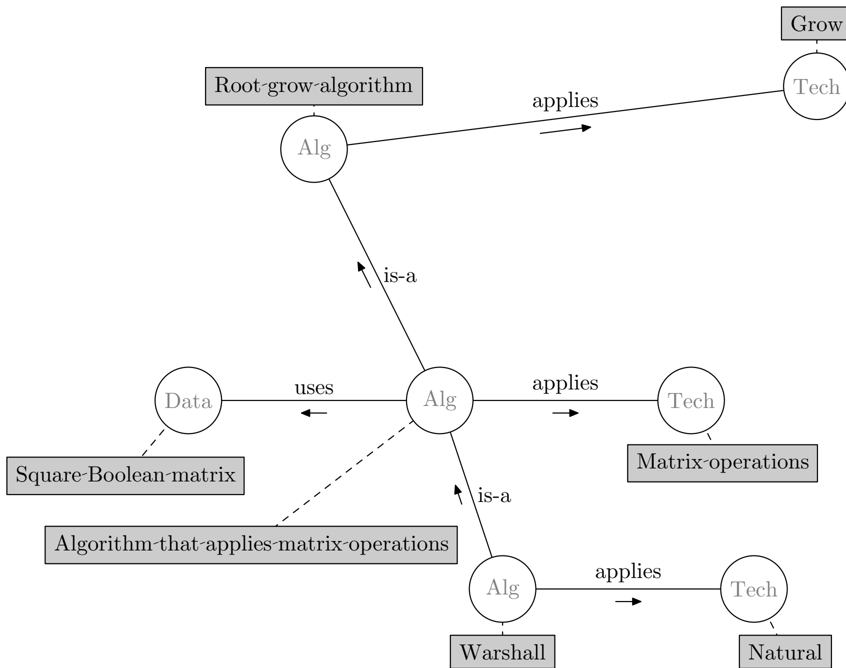


Figure 12.6.1: Topics related to Warshall's algorithm

Algorithm 12.6.1: Warshall's Algorithm

---

const $R \subseteq U \times U$
$\quad n = |U| \in \mathbb{N}^+$
var $M \in \mathbb{B}[n, n]$

$M := \Phi(R)$
for $j : j \in \mathbb{N}_n \quad \rightarrow$
  for $i : i \in \mathbb{N}_n \quad \rightarrow$
    if $\neg M[i, j] \rightarrow$ **skip**
    $[\!]$ $M[i, j] \quad \rightarrow$
      for $k \in \mathbb{N}_n \quad \rightarrow$
        $M[i, k] := M[i, k] \vee M[j, k]$
      rof
    fi
  rof
rof
{**Post:** $\Psi(M) = R^+$}

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

12.6.2 Visualisation of Warshall's algorithm

Figure 12.6.2 shows how this algorithm determines the TC of an example relation. The example relation is the relation that was used as the example in Section 3.5 where different representation models for relations were explained. When a node (shown on the arrow indicating a processing step), is processed, its successors are added as successors to each of its predecessors.

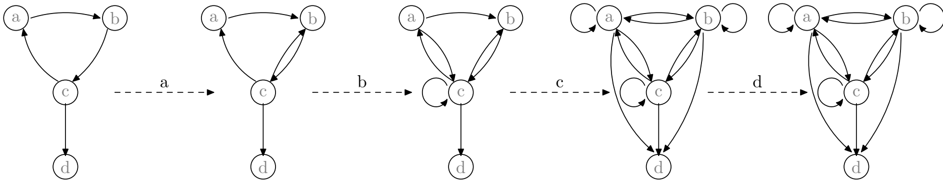


Figure 12.6.2: Processing using Warshall's algorithm

In the leftmost diagram the given relation is shown. In the first step node **a** of this relation is processed. The successors of **a** should thus be added as successors of each of its predecessors. **c** its only predecessor of **a**, and **b** is the only successor of **a**. Thus only one arc, namely the arc from **c** to **b** is added. When adding this arc, **b** becomes a successor of **c**.

In the next step node **b** is processed. First the successors of **b** are found. **b**, at this point, has only one successor namely **c**. Now **c** has to be added as a successor to each of **b**'s predecessors. At this point, **b**'s predecessors are nodes **a** and **c**. Thus two arcs are added. These are **a** to **c** and **c** to **c**.

Then node **c** is processed, the current successors of **c** are found. They are nodes **a**, **b**, **c** and **d**. These should be added as successors of each of the predecessors of **c**. The predecessors of **c** are nodes **a**, **b**, and **c**. Thus the following arcs should be added to expand the successor list of node **a**: **a** to **a**, **a** to **b**, **a** to **c** and **a** to **d**; similarly **b** to **a**, **b** to **b**, **b** to **c** and **b** to **d**; as well as **c** to **a**, **c** to **b**, **c** to **c** and **c** to **d** should all be added. Obviously arcs that are already in the relation to date need not be added again. At this point the following five new arcs are added: **a** to **a**, **a** to **d**, **b** to **a**, **b** to **b**, **b** to **d**.

Finally node **d** is processed. Since node **d** does not have any successors, no arcs need to be added.

12.6.3 The derivation tree of Warshall's algorithm

Figure 12.6.3 shows that Warshall's algorithm is derived from the root TC algorithm in three steps. The steps are the application of the fundamental grow technique, the transformation of the implementation to use square Boolean matrices, and performing the required matrix operations in sequential (natural) order.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

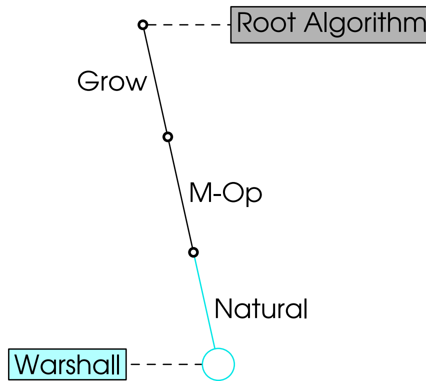


Figure 12.6.3: Derivation tree of Warshall's algorithm

12.6.4 Verification

Consider the following expression that is the right hand side of the equation calculated in the loop body of Algorithm 12.5.1 (MatrixGrow):

$$M + (M \times \mathbb{I}_{\{j\}} \times \mathbb{I}_{\{j\}} \times M) \tag{12.12}$$

Let $M' = M \times \mathbb{I}_{\{j\}} \times \mathbb{I}_{\{j\}} \times M$. The value of the matrix defined by Expression 12.12 can be determined by calculating $M[i,k] + M'[i,k]$ for each $i,k \in \mathbb{N}_n$. Lemma 12.6.4 shows that $M'[i,k] = M[i,j] \wedge M[j,k]$. Thus

$$M'[i,k] = \begin{cases} 0 & \text{if } \neg M[i,j]; \\ M[j,k] & \text{if } M[i,j]. \end{cases} \tag{12.13}$$

Let M'' be the value of Expression 12.12. It is expressed as $M + M'$. If the value of M' as expressed in Equation 12.13 is substituted, it follows that:

$$M''[i,k] = \begin{cases} M[i,k] & \text{if } \neg M[i,j]; \\ M[i,k] \vee M[j,k] & \text{if } M[i,j]. \end{cases} \tag{12.14}$$

This expression is implemented in the definition of the algorithm using an if-statement to substitute the value of $M[i,k]$ by the value of $M[i,k] \vee M[j,k]$ for each value of k in all cases where $M[i,j]$ holds. Thus, Algorithm 12.6.1 (Warshall) is a correct implementation of Algorithm 12.5.1 (MatrixGrow).

12.6.5 Complexity

To determine the complexity Warhall's algorithm the complexity of the loops of this algorithm and the number of iterations needed in each loop to calculate the result is examined.

The algorithm body consists of three levels of nested loops. Each loop requires n iterations. Hence, the resulting complexity of the nested loop structure is $\Theta(n^3)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & (M \times \mathbb{I}_{\{j\}} \times \mathbb{I}_{\{j\}} \times M)[i,k] \\
= & \{\text{Matrix multiplication is associative}\} \\
 & ((M \times \mathbb{I}_{\{j\}}) \times (\mathbb{I}_{\{j\}} \times M))[i,k] \\
= & \{\text{Definition of matrix multiplication of Boolean matrices}\} \\
 & \langle \exists t :: (M \times \mathbb{I}_{\{j\}})[i,t] \wedge (\mathbb{I}_{\{j\}} \times M)[t,k] \rangle \\
= & \left\{ \begin{array}{l} \text{All entries in } M \times \mathbb{I}_{\{j\}} \text{ are 0 except possibly for} \\ \text{entries in its } j^{th} \text{ column, which corresponds} \\ \text{with the } j^{th} \text{ column of } M. \end{array} \right\} \\
 & \langle \exists t :: M[i,j] \wedge j = t \wedge (\mathbb{I}_{\{j\}} \times M)[t,k] \rangle \\
= & \left\{ \begin{array}{l} \text{All entries in } \mathbb{I}_{\{j\}} \times M \text{ are 0 except possibly for} \\ \text{entries in its } j^{th} \text{ row, which corresponds} \\ \text{with the } j^{th} \text{ row of } M. \end{array} \right\} \\
 & \langle \exists t :: M[i,j] \wedge j = t \wedge M[j,k] \wedge j = t \rangle \\
= & \{\text{t=j}\} \\
 & M[i,j] \wedge M[j,k]
\end{array}
$$

Lemma 12.6.4: $(M \times \mathbb{I}_{\{j\}} \times \mathbb{I}_{\{j\}} \times M)[i,k] = M[i,j] \wedge M[j,k]$

Listing C32 defines this complexity of Warshall's algorithm in the TM using LTM notation. It points to the above paragraph as an occurrence of a justification of the complexity of this algorithm.

This algorithm has the same complexity as Warren's algorithm discussed in Section 17.1. When compared to Warren's algorithm it uses unnecessary processing time when it redundantly processes the entries on the diagonal of the matrix. These operations are avoided by Warren's algorithm. Warshall's algorithm, however, saves some processing time by not having to take the position of an entry in the matrix into account. Warshall's algorithm is often preferred above Warren's algorithm for its simplicity.

12.7 Summary

This chapter describes the derivation steps of the first two concrete algorithms in the TM of TCA. These are the algorithms by Prosser [203] and by Warshall [252] — Algorithm 12.4.1 (Prosser) and Algorithm 12.6.1 (Warshall). These algorithms are derived from the respective abstract algorithms that are defined in Chapter 11 as the root algorithms of the two major branches of the taxonomy namely Algorithm 11.3.1 (Coat) and Algorithm 11.4.1 (Grow).

In both cases the first derivation step was achieved by transforming the relevant abstract algorithm in Chapter 11 to an equivalent algorithm that uses Boolean matrices to represent the data on which the algorithm operates. The arguments to show that these transformations preserve the correctness of the algorithms

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

are based on the work presented in Chapter 6. The resulting algorithms are Algorithm 12.2.2 (MatrixCoat) and Algorithm 12.5.1 (MatrixGrow).

Algorithm 12.4.1 (Prosser) is derived from Algorithm 12.2.2 (MatrixCoat) by applying two steps. The first step is the application of a safe upper bound as an algorithmic technique and the second step is the use of a sequential order to select the elements of $\mathbb{N}_{n-2}$ for the execution of the loop.

Algorithm 12.6.1 (Warshall) is derived from Algorithm 12.5.1 (MatrixGrow) by applying one step, namely by using a sequential order to iterate through the entries of M.

Attributes of the algorithms and artifacts associated with the algorithms are specified. These include the following:

- The definition of the algorithms using GCL.
- The positioning of each algorithm in the TM of TCA in terms of its relation to other topics in the TM.
- A visualisation of Warshall's algorithm.
- The complexity of each of the algorithms and arguments to justify these complexities.
- Proofs that all the derivation steps preserve the correctness of the algorithms.

These have been newly described in a consistent format in this chapter for the purpose of providing this information to be included in the TM of TCA.

To my knowledge the identification of the relation between these algorithms in a derivation hierarchy of this form has not been done before. The correctness proofs of these algorithms based on correctness preserving transformations, presented here differs from the proofs that were given by the authors of the algorithms.

The next chapter investigates the optimisation of Algorithm 12.6.1 (Warshall) through the application of loop interchange.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 13

Loop Interchange

The basic concrete implementation of the grow algorithm is the algorithm attributed to Warshall. It is discussed in Section 12.6. Warshall's algorithm processes the elements of the adjacency matrix in column order while storing data in row order. This may cause memory thrashing.

A technique that is often used to avoid this kind of memory thrashing is loop interchange. It is the algorithmic technique of exchanging the order of two iteration variables used by nested loops. The variable used in the inner loop switches to the outer loop, and vice versa. This technique is often applied to ensure that the elements of a matrix are accessed in the order in which they are stored in memory. By doing this locality of reference is improved and consequently memory thrashing is reduced. Warren [251] estimates that the application of loop interchange to Algorithm 12.6.1 (Warshall) may cause fewer page faults by a factor of approximately $\frac{n}{2}$.

This chapter investigates the consequences of applying loop interchange to Algorithm 12.6.1 (Warshall). Loop interchange, has the unfortunate effect that the algorithm is no longer correct. All is, however, not lost. It is shown that after loop interchange the algorithm can be amended to define an abstract algorithm that correctly calculates the TC. A concrete implementation of this newly formed algorithm is also presented.

13.1 Naïve implementation

13.1.1 Investigating loop interchange

Algorithm 13.1.3 (GrowRow) is derived from of Algorithm 12.6.1 (Warshall) by applying the algorithmic technique called *interchange loops*. The effect of this change is altering the processing order from column major order to row major order. The specification of this technique in the TM of TCA is included in Listing C11.

The action in every step when processing Algorithm 12.5.1 (MatrixGrow) in column order is to add the successors of the current node as successors to each of its predecessors. When processing in row order without changing anything else in

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the algorithm, this action is changed. The action becomes adding the successors of each of the current node's successors, as successors to the current node. The difference in terms of the order in which arcs are added and the resulting graph can be observed when comparing Figure 12.6.2 and Figure 13.1.1 that shows step-by-step traces of these operations for one test case.

Section 12.6 verifies that the actions of Algorithm 12.6.1 (Warshall) reach the TC of the given algorithm having processed each node once. On the contrary, the example execution of this algorithm that does the same processing in a different order, shown in Figure 13.1.1, fails to reach the TC of the given relation having processed each node once.

Figure 13.1.1 shows one pass of the outer loop when executing this algorithm to determine the TC of an example relation. In each step the name of the node that is processed is shown on the arrow indicating the processing step.

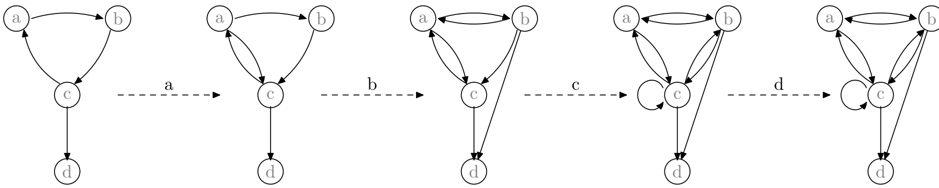


Figure 13.1.1: One iteration of the naïve grow algorithm

When a node is processed, the successors of the successors of this node are added as successors to the node being processed. In the first step node **a** is processed. Node **b** is the only successor of node **a**, and node **c** is the only successor of node **b**. Thus, the arc (**a**, **c**) is added. The second step is to process node **b**. Node **c** is the only successor of node **b**. The successors of node **c**, namely nodes **a** and **d**, are added. Thus, the arcs (**b**, **a**) and (**b**, **d**) are added. In the step where node **c** is processed the successors of **c** are considered. These are nodes **a** and **d**. Node **d** has no successors, therefore no changes are made when considering the successors of **d**. The successors of node **a** are nodes **b** and **c**. Thus the two arcs (**c**, **b**) and (**c**, **c**) are added. In the step where **d** is processed, no changes are made as node **d** has no successors.

Warren [251] gives a similar example that shows that loop interchange may cause the algorithm not to work in one pass anymore.

13.1.2 Fixing the problem of failure to reach the TC in one pass

In Section 13.1.3 it is shown that the value of M converges to M^+ when the actions performed on row order would be repeated a sufficient number of times. To derive a correct algorithm, an outer loop that iterates until the TC is reached is added.

Algorithm 13.1.3 (GrowRow) defines this algorithm. Figure 13.1.2 shows the topics related to this algorithm. The topic specification of this algorithm in the TM of TCA using LTM notation is given in Listing C22.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Figure 13.1.3 shows the derivation steps of Algorithm 13.1.3 (GrowRow). It is derived from Algorithm 12.6.1 (Warshall). The derivation from root TC algorithm is four steps. The steps are the application of the fundamental grow technique, the transformation of the implementation to use square Boolean matrices, processing in natural (sequential) order, and lastly interchanging the two outer loops of the algorithm.

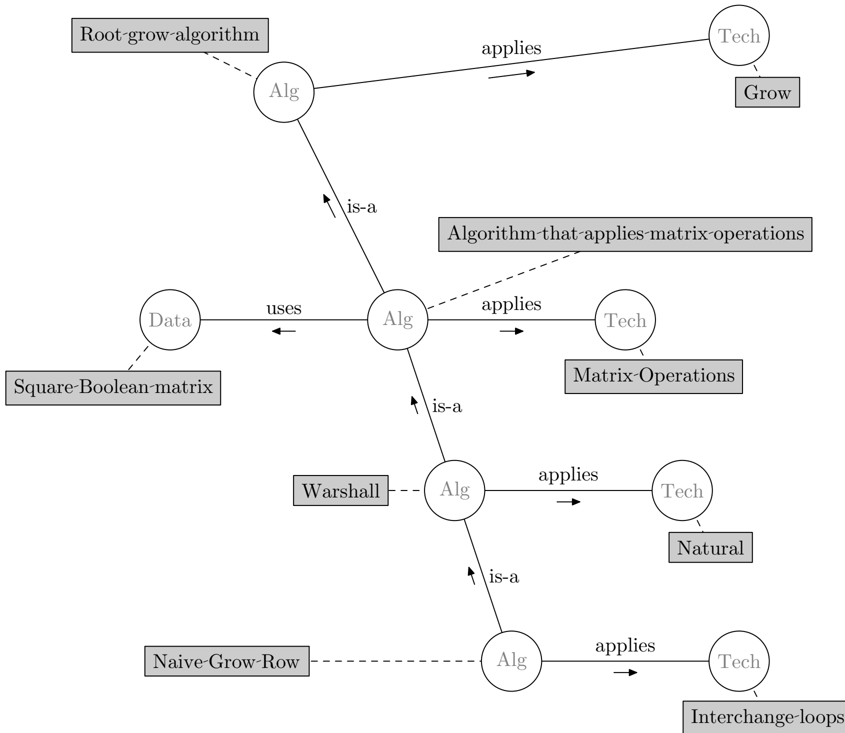


Figure 13.1.2: Topics related to the naïve grow row algorithm

13.1.3 Verification

This algorithm is essentially the same as Algorithm 12.6.1 (Warshall), which is shown to be correct in Section 12.6.4. During the processing of each iteration of this algorithm, edges are never removed, while edges may be added. Edges that are added in any given pass are edges that represent a direct connection between two elements that were connected via one other element at the end of the previous iteration, thus, belonging to the TC of the original relation. Figure 13.1.1, however, illustrates that the algorithm might not add all the required arcs in any given pass.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 13.1.3: Naive row-order grow

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^+$
var $M \in \mathbb{B}[n,n]$

$M = \Phi(R)$
do $M \neq M^+ \rightarrow$
  for $i : i \in \mathbb{N}_n \rightarrow$
    for $j : j \in \mathbb{N}_n \rightarrow$
      if $\neg M[i,j] \rightarrow$ **skip**
      $[\!]$ $M[i,j] \rightarrow$
        for $k \in \mathbb{N}_n \rightarrow$
          $M[i,k] := M[i,k] \vee M[j,k]$
        rof
      fi
    rof
  rof
od
$\{$**Post:** $\Psi(M) = R^+\}$

Let M_t represent the value of M after the tth iteration of the outer loop of Algorithm 13.1.3 (GrowRow). The following is a formal expression of the above explanation. Predicate 13.1 states that edges are never removed, while Predicate 13.2 states that when an edge is added, the added edge connects two elements that were connected via one element before the operation:

$$M_t[i,k] \implies M_{t+1}[i,k] \quad (13.1)$$
$$\neg M_t[i,k] \wedge M_{t+1}[i,k] \implies \langle \exists j :: M_t[i,j] \wedge M_t[j,k] \rangle \quad (13.2)$$

Thus the following monotonically increasing chain that converges to M^+ is formed:

$$M = M_0 \sqsubseteq M_1 \sqsubseteq M_2 \sqsubseteq \cdots \sqsubseteq M^+ \quad (13.3)$$

As the relation (of which the TC is calculated) is finite and M_t are monotonicly increasing with each value of t, it is evident that M_t will, after a finite number of iterations, reach the value of M^+. Thus the loop repeating the process until the TC is reached, terminates. A concrete implementation is presented in the next section.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

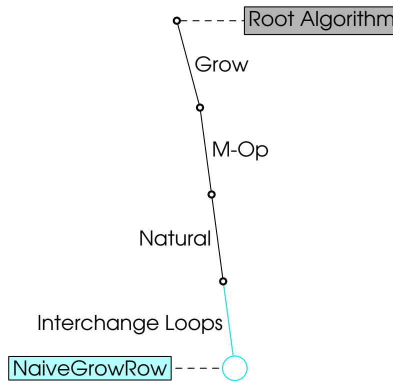


Figure 13.1.3: Derivation tree of the naïve grow algorithm

13.2 Martynyuk's algorithm

Martynyuk's algorithm [162] was first published in March 1962 in a Russian journal. This publication appeared a few months after Warshall's algorithm was first published, but before Warshall's algorithm was added to the collected algorithms of the ACM (CALGO) [115]. I obtained an English translation of Martynyuk's article that appeared in 1963. Martynyuk [163] refers to Prosser (Section 12.4) and to Baker (Section 14.1), but does not mention Warshall.

Martynyuk [163] showed that the outer loop in Algorithm 13.1.3 (GrowRow), needs at most $log_2 n$ iterations to reach the desired post-condition. Thus the value of the safe upper bound applied by this algorithm is $log_2 n$. The definition of this algorithm is shown in Algorithm 13.2.1 (Martynyuk).

13.2.1 Description of Martynyuk's algorithm

Listing C24 specifies this algorithm as a topic in the TM of TCA using LTM notation. Figure 13.2.1 shows the topics related to this algorithm. It is a concrete version of Algorithm 13.1.3 (GrowRow). It uses a safe upper bound. This is the same technique that was applied to derive Prosser's algorithm from Algorithm 12.2.2 (MatrixCoat). The topic specification of this technique in the TM of TCA using LTM notation is included in Listing C8. The specification of occurrences of implementations of Martynyuk's algorithm using LTM notation can be found in Listing C46.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

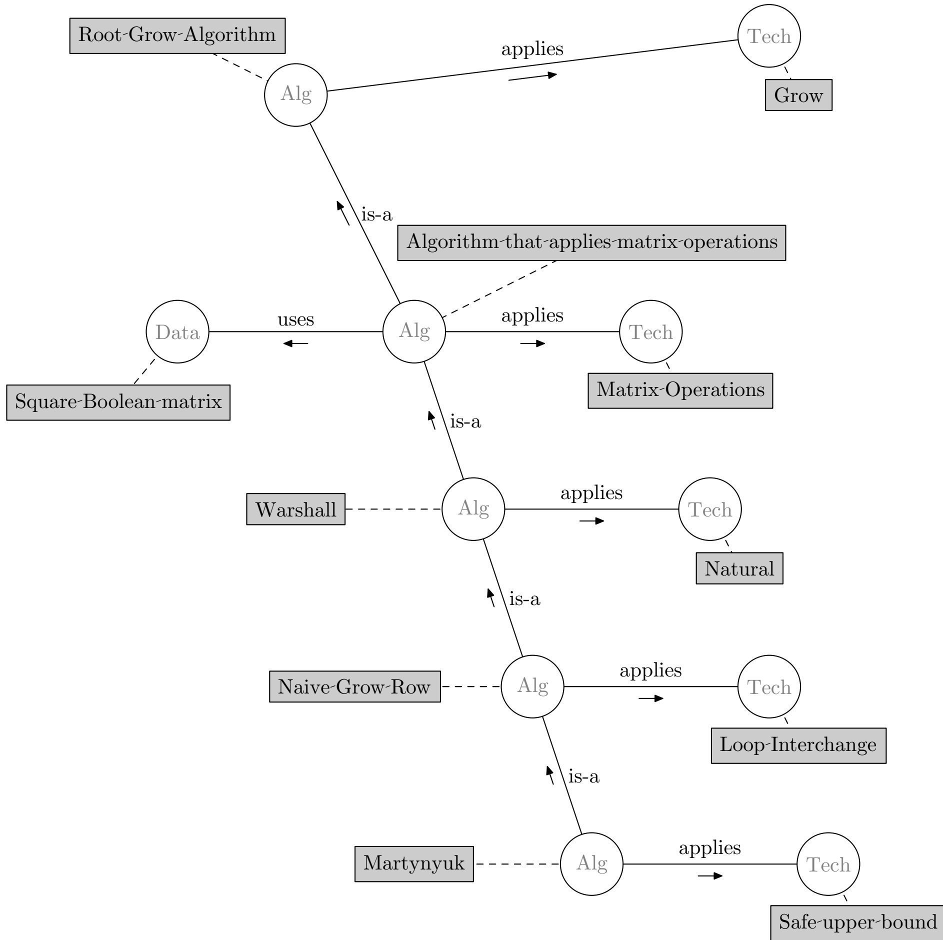


Figure 13.2.1: Topics related to Martynyuk's algorithm

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 13.2.1: Martynyuk's Algorithm

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^{+}$
var $M \in \mathbb{B}[n, n]$

$M = \Phi(R)$
for $r : r \in \mathbb{N}_{log_2 n}$ $\rightarrow$
  for $i : i \in \mathbb{N}_n$ $\rightarrow$
    for $j : j \in \mathbb{N}_n$ $\rightarrow$
      if $\neg M[i, j] \rightarrow$ **skip**
      $[\!]$ $M[i, j]$ $\rightarrow$
        for $k \in \mathbb{N}_n$ $\rightarrow$
          $M[i, k] := M[i, k] \vee M[j, k]$
        rof
      fi
    rof
  rof
rof
$\{$**Post:** $\Psi(M) = R^{+}\}$

13.2.2 Verification

Let M_t denote the value of M in Algorithm 13.1.3 (GrowRow) after t iterations. Here I argue that, for any acyclic path $P \in U[m]$ in relation R, at most $log_2 m$ iterations of the outer loop of Algorithm 13.1.3 (GrowRow) are needed to ensure that the resulting relation has a direct connection from the head of the path to the tail of the path. This is achieved by showing that predicate 13.4 holds for all acyclic paths $P \in U[m]$ in relation R.

$$t > log_2 m \implies M_t[p_0, p_{m-1}] \tag{13.4}$$

The proof is done by induction on the length of the path; i.e. the value of m.

The first step of this induction proof requires verification of the trivial case; i.e. the case when $m = 1$. To this end let P be an acyclic path in R of length 1 in R i.e $P \in U[2]$. It can be said that $P = (p_0, p_1)$ and $\mathcal{R}(P) = \{(p_0, p_1)\}$.

In this trivial case p_0 is the head and p_1 is the tail of P. As $M[p_0, p_1]$ is set during initialisation in Algorithm 13.1.3 (GrowRow) and the algorithm never un-sets any entry, it follows that $\langle\ \forall t \mid t > 0 \mid M_t[p_0, p_1]\ \rangle$, thus, Predicate 13.4 holds for all acyclic paths P of length 1 in relation R.

Continuing the induction proof, Lemma 13.2.2 shows that Predicate 13.4 holds for any acyclic path $P \in U[m]$ in relation R under an induction assumption that the equation holds for all paths with length half the length of P.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & P = \langle\!\langle\, i \mid i \in \mathbb{N}_{m-1} \mid p_i \,\rangle\!\rangle \\
\equiv & \left\{ \begin{array}{l} \text{P can be written as the concatenation of two paths,} \\ \text{each consisting at most } \lceil m/2 \rceil \text{ of the entries in P} \end{array} \right\} \\
 & P = \langle\!\langle\, i \mid i \in \mathbb{N}_{m-1}, i \leq \lfloor m/2 \rfloor \mid p_i \,\rangle\!\rangle || \langle\!\langle\, i \mid i \in \mathbb{N}_{m-1}, i \geq \lfloor m/2 \rfloor \mid p_i \,\rangle\!\rangle \\
\equiv & \left\{ \begin{array}{l} \text{Applying the induction assumption to the sub-paths} \\ \text{each consisting at most } \lceil m/2 \rceil \text{ entries} \end{array} \right\} \\
 & t > log_2(\lceil m/2 \rceil) \implies M_t[p_0, p_{\lfloor m/2 \rfloor}] \wedge M_t[p_{\lfloor m/2 \rfloor}, p_{m-1}] \\
\Rightarrow & \left\{ \begin{array}{l} \lceil m/2 \rceil \geq m/2 \text{ and } a \geq b \Rightarrow log_2 a \geq log_2 b \\ \text{and transitivity of } \geq \end{array} \right\} \\
 & t > log_2(m/2) \implies M_t[p_0, p_{\lfloor m/2 \rfloor}] \wedge M_t[p_{\lfloor m/2 \rfloor}, p_{m-1}] \\
\equiv & \{ \text{Quotient log rule } log_b(x/y) = log_b x - log_b y \} \\
 & t > log_2 m - log_2 2 \implies M_t[p_0, p_{\lfloor m/2 \rfloor}] \wedge M_t[p_{\lfloor m/2 \rfloor}, p_{m-1}] \\
\equiv & \{ log_x x = 1 \} \\
 & t > log_2 m - 1 \implies M_t[p_0, p_{\lfloor m/2 \rfloor}] \wedge M_t[p_{\lfloor m/2 \rfloor}, p_{m-1}] \\
\Rightarrow & \left\{ \begin{array}{l} \text{Executing the next iteration of the outer loop.} \\ \text{Since } M_t[p_0, p_{\lfloor m/2 \rfloor}], \text{ row } p_{\lfloor m/2 \rfloor} \text{ is added to row } p_0. \end{array} \right\} \\
 & t > log_2 m \implies M_t[p_0, p_{m-1}]
\end{array}
$$

Lemma 13.2.2: $t > log_2 m \Rightarrow M_t[p_0, p_{m-1}]$

The notation $\lceil x \rceil$ and $\lfloor x \rfloor$ is used to respectively mean the *ceiling* and *floor* of x with x being a real value. $\lceil x \rceil$ refers to the smallest integer greater or equal to x and $\lfloor x \rfloor$ is the largest integer smaller or equal to x. Using this notation, the induction assumption is formulated more precisely as: Predicate 13.4 holds for any acyclic path $P \in U[\lceil m/2 \rceil]$ in relation R.

The validity of Predicate 13.4 for a path of given length implies that the TC of all the nodes on the path has been calculated after $log_2 m$ iterations since the equation guarantees that the head of every sub-path of such path is connected to the tail of that sub-path. Furthermore, the equation is valid for all paths of a given length. Thus at the point when the TC of the nodes on the longest path in the relation has been calculated, the TC of all shorter paths and consequently the TC of the entire relation has been reached.

Recall that for every path in R from u_i to u_j, there is an acyclic path P from u_i to u_j with $\ell(P) < |U| = n$ (Section 5.4.2). The number of iterations needed to reach the TC of the relation is therefore exactly the number of iterations needed to find the TC of the longest acyclic path in R.

Lemma 5.5.2 states that the length of the longest acyclic path in any relation is less than $|U|$. The application of Lemma 13.2.2 asserts that the number of iterations needed to find the TC of the longest path is less than $log_2|U| = log_2 n$. Thus $M = \Phi(R^+)$ after $log_2 n$ iterations of the algorithm. This proves that when Algorithm 13.2.1 (Martynyuk) is derived from Algorithm 13.1.3 (GrowRow) by applying a safe upper bound of $log_2 m$ preserves correctness.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

13.2.3 Complexity

This algorithm has four levels of nested loops. As shown in Section 13.2.2, the outer loop requires $log_2 n$ operations. Each of the remaining loops nested in the body of this algorithm requires n iterations. Hence, the resulting complexity of the nested loop structure is $\Theta((log\ n) \times n \times n \times n) = \Theta(n^3 log\ n)$. This complexity of Martynykuk's algorithm is specified in Listing C33.

13.2.4 Derivation tree of Martynyuk's algorithm

The first three steps to derive this algorithm from the root TC algorithm are the same as those of Warshall's algorithm. These steps are the application of the fundamental *grow* technique, and the transformation of the implementation to use square Boolean matrices to be able to apply matrix operations, and the application of these operations in a natural order. This algorithm further applies loop interchange and a safe upper bound. Figure 13.2.4 shows this derivation path.

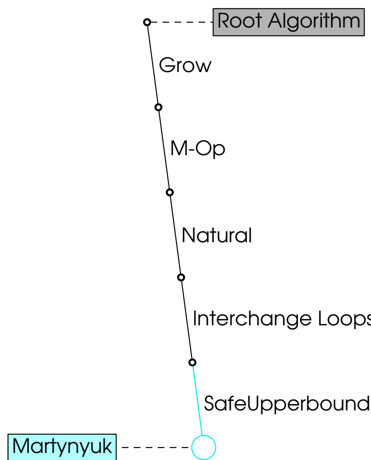


Figure 13.2.4: Derivation tree of Martynyuk's algorithm

13.3 Summary

This chapter investigates the optimisation of Algorithm 12.6.1 (Warshall) through the application of loop interchange. It is discovered that when the loops on Warshall's algorithm are interchanged, the process no longer yields the TC after a single pass as is the case with Warshall's algorithm. Next, an algorithm is derived from Algorithm 12.6.1 (Warshall) by adding an outer loop that requires enough iterations to reach the TC. The correctness of this algorithm is verified. The lemma that is used in this correctness argument is a new lemma, stated and proved specifically for this purpose.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

A concrete implementation of this derived algorithm is then introduced, where a safe upper bound is used to control the added outer loop. The algorithm that is defined in this manner turns out to be the unfamiliar algorithm published by Martynyuk [162].

The artifacts required to add Martynyuk's algorithm to the TM of TCA were created in this chapter, namely ones corresponding to correctness verification and determining the complexity of the algorithm.

An alternative solution to address the issue that one needs potentially a large number of passes to complete the calculation of the TC of a relation after the loop interchange technique is applied to Algorithm 12.6.1 (Warshall), is to split the iteration space in such a way that fewer passes are needed. An algorithm applying this technique is discussed in Section 17.1. The next chapter investigates another alternative to the *safe upper bound* algorithmic technique to terminate the outer loops of Algorithm 12.4.1 (Prosser) and Algorithm 13.2.1 (Martynyuk).

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 14

Monitoring change

Often algorithms are applied to manipulate data in incremental steps until a desired state is reached. In many cases it is difficult to determine whether the desired state is reached. If it is known how many iterations are needed to reach the desired state, this safe upper bound can be applied to avoid the need to determine in other ways if the desired stated has in fact been reached. This is the case with Prosser's algorithm and Martynyuk's algorithm. It is, however, not always possible to determine such a safe upper bound.

Another technique that can be applied is to observe when this stable state is reached. This can be done by monitoring whether changes are made during the execution of the loop. If the execution of the loop body did not change the state of the data, all successive iterations will also not change the state. Therefore the execution may be terminated. When applying this technique one would typically change the type of outer loop structure from a counting loop (`for`-loop) to a conditional loop (`while`-loop).

The application of such monitor can be useful in situations where it can reasonably be expected that a stable state may be reached well in advance of a calculated safe upper bound. Such a monitor could also be used in situations where a safe upper bound is unknown. The application of a change monitor is only practical if the cost of monitoring does not exceed the cost gained through earlier termination.

In this chapter, algorithms are explored that apply a change monitor instead of a safe upper bound to terminate the outer loop of the transitive closure algorithms so far discussed.

14.1 Baker's algorithm

Baker's algorithm is a concrete implementation of Algorithm 13.1.3 (GrowRow). It differs from Algorithm 13.2.1 (Martynyuk) only in the application of a different deterministic mechanism to terminate the loop. The TM specification pointing to implementations of Baker's algorithm in the TM of TCA can be found in Listing C48.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

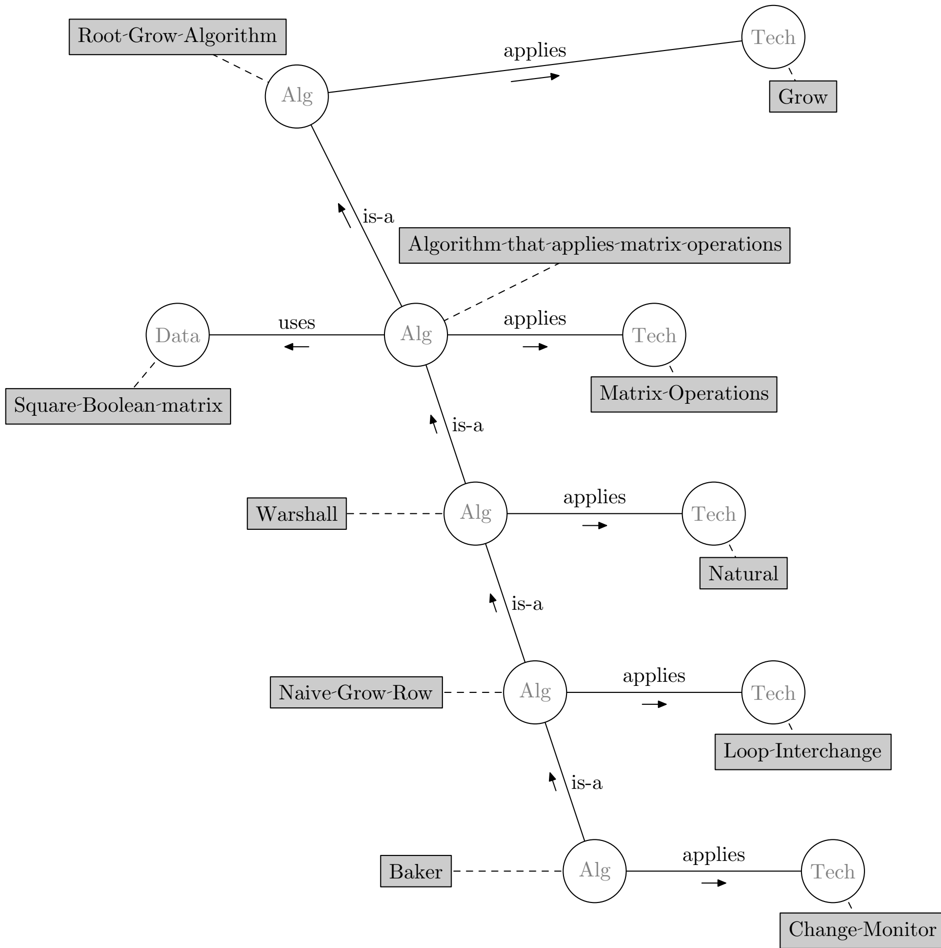


Figure 14.1.1: Topics related to Bakers's algorithm

In February 1962, a month before Martynyuk's algorithm was published, Baker [17] published the following description of his concrete implementation of Algorithm 13.1.3 (GrowRow):

> *Given a Boolean matrix (i.e. a matrix whose entries are only 0's and 1's): scan row 1 of the matrix until a 1 is encountered, say in column j. OR row j into row 1. Continue scanning columns j+1, j+2, etc. of row 1 until another 1 is found. Then OR the row corresponding to its column into row 1. Continue in this manner until the whole row is scanned. Then process rows 2, 3, etc. in the same way. When the last row has been scanned, go to row 1 again. Keep going through the matrix until one complete pass has been made without changing it.*

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

14.1.1 Description of Baker's algorithm

The algorithmic technique that Baker's algorithm applies to derive a concrete version of Algorithm 13.1.3 (GrowRow), is called a *change monitor*. The TM definition of this technique as a topic in the TM of TCA is included in Listing C8. The definition of Baker's algorithm as a topic in the TM of TCA is shown in Listing C25.

Algorithm 14.1.1: Baker's Algorithm

---

const $R \subseteq U \times U$
$\quad\quad n = |U| \in \mathbb{N}^+$
var $M_0 \in \mathbb{B}[n,n]$
$\quad\quad M_1 \in \mathbb{B}[n,n]$

$M_0, M_1 := \Phi(R), \mathbb{O}_n$
do $M_0 \neq M_1 \quad \rightarrow$
$\quad M_1 := M_0$
$\quad$ **for** $i : i \in \mathbb{N}_n \quad \rightarrow$
$\quad\quad$ **for** $j : j \in \mathbb{N}_n \quad \rightarrow$
$\quad\quad\quad$ **if** $\neg M_0[i,j] \rightarrow$ **skip**
$\quad\quad\quad$ $[\!]$ $M_0[i,j] \quad \rightarrow$
$\quad\quad\quad\quad$ **for** $k \in \mathbb{N}_n \quad \rightarrow$
$\quad\quad\quad\quad\quad M_0[i,k] := M_0[i,k] \vee M_0[j,k]$
$\quad\quad\quad\quad$ **rof**
$\quad\quad\quad$ **fi**
$\quad\quad$ **rof**
$\quad$ **rof**
od
$\{$**Post:** $M_0 = M_1, \Psi(M_0) = R^+\}$

14.1.2 Verification

The operations performed by this algorithm are the same as the operations performed by Algorithm 13.2.1 (Martynyuk). Equation 13.3 that was established when Algorithm 13.1.3 (GrowRow) was verified, guarantees that the result grows with each iteration. In Section 13.2.2 it was shown that the value of M_0 converges to M^+ when these operations are performed on M_0 and that $M_0 = M^+$ after a finite number of iterations. As soon as M_0 has assumed the value of M^+, further iterations will not change its value. At this stage the change monitor detects stabilisation and the loop terminates. Consequently the algorithm terminates with M_0 containing the desired value.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

14.1.3 Complexity

Assume a naive implementation where the change monitor is applied by making a copy of the entire matrix before an iteration of the outer loop and comparing the copy with the value of the matrix after the iteration was performed. The guard of the outer loop is a $\Theta(n^2)$ operation since the comparison of two matrices requires each of the $n \times n$ elements to be compared. The body of the loop starts with a step that executes matrix assignment to preserve the current value of the matrix to be able to verify if it had changed later. The complexity of assigning one matrix to another is $\Theta(n^2)$ since the value of each of the $n \times n$ elements has to be assigned. The rest of the body of the outer loop is three levels of nested loops, each requiring n operations, yielding a complexity of $\Theta(n^3)$. Therefore, the complexity of the body of the outer loop is $\Theta(n^2 + n^2 + n^3) = \Theta(n^3)$.

In Section 13.2 it is shown that the algorithm will reach the state where $M_0 = M^+$ in at most $log_2 n$ iterations. Consequently it will take at most $log_2 n + 1$ iterations for Algorithm 14.1.1 (Baker) to terminate as it needs an extra iteration to verify that no changes occurred during its execution. Thus, the complexity of the algorithm is $O((log_2 n + 1) \times n^3) = O(n^3 log_2 n)$. The definition of this complexity in the TM of TCA is given in Listing C34.

A more efficient change monitor can be implemented. It will, however, have no impact on the complexity of the algorithm. The complexity of the body of the algorithm is dominated by the three levels of nested loops each requiring n iterations. Thus, even if the change monitor is implemented in constant time, the complexity of the algorithm would not be effected.

14.1.4 Position in a derivation tree

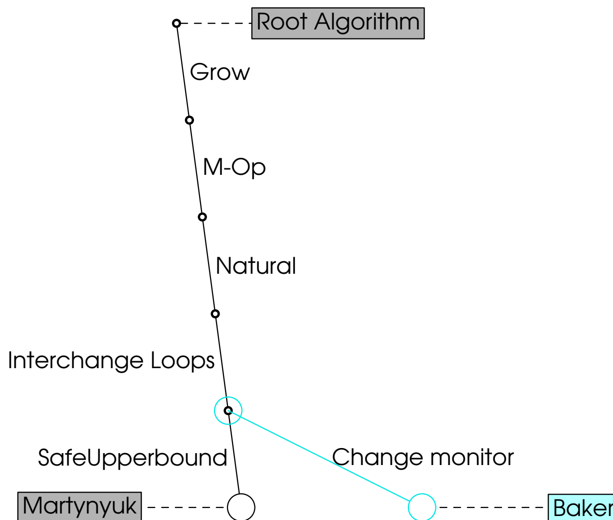


Figure 14.1.4: Derivation path of Baker's algorithm

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The derivation steps of Baker's algorithm is shown in Figure 14.1.4. The first three steps to derive this algorithm from the root TC algorithm are the same as that of Martynyuk's algorithm. These three steps are the application of the fundamental *grow* technique, the transformation of the implementation to use square Boolean matrices to be able to apply matrix operations, and systematically processing the entries in the matrix in *row* order. This algorithm does not apply a safe upper bound to terminate the outer loop. Instead it applies a change monitor and terminates when a complete pass occurs without any changes. When compared with Martynyuk's algorithm, the derivation of this algorithm is achieved by applying the *ChMonitor* technique instead of the *UpperBound* technique.

14.2 A variant of Prosser's algorithm

The derivation tree shown in Figure 14.2.4 in Section 14.2.4 is the combined derivation tree containing a selection of the algorithms that has been added to the TM so far. The two branches of the tree are not the same length. They are, however, drawn to appear the same length to emphasise the symmetry that can be achieved by adding the derivation shown in blue. The observed gap in the derivation tree, when this branch is omitted, leads to the identification of a new algorithm. This new algorithm is the subject of this section.

Baker's algorithm and Martynyuk's algorithm in the left branch are both derived from the same abstract algorithm by applying opposing techniques to determine when to terminate iteration of the outer loop. These techniques are respectively called *change monitor* and *safe upper bound*. The derivation path of Prosser's algorithm in the right branch includes a step where *safe upper bound* is applied. The algorithm proposed here is a new algorithm that applies a strategy in the place of *safe upper bound* to terminate the loop. I call this algorithm the Monitored Coat Algorithm because it is a coat algorithm that applies the *change monitor* technique. It has to my knowledge not been published before. It differs from Prosser's algorithm similarly to how Baker's algorithm differs from Martynyuk's algorithm in that it applies a change monitor instead of a safe upper bound.

14.2.1 Description of the monitored coat algorithm

The derivation path of the proposed algorithm is on the same branch as Prosser's algorithm but branches when it applies *change monitor* where the derivation path to Prosser's algorithm applies *safe upper bound*. The TM definition of the monitored coat algorithm in the TM of TCA is shown in Listing C26 while Figure 14.2.1 visualises the topics related to this algorithm.

The outer loop of Prosser's algorithm requires the selection of an element of a finite set for each iteration. The order in which these elements are chosen is not deterministic. For this reason the implementation of Prosser's algorithm requires an additional derivation step namely to select the elements in a natural order. In contrast, the outer loop of the Monitored Coat algorithm is terminated based on a conditional statement. Therefore there is no need for any additional specifications for its implementation.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The TM specification pointing to implementations of this variation of Prosser's algorithm in the TM of TCA can be found in Listing C50. For efficiency the change monitor that is applied in these implementations registers the change by means of a flag that is raised when an entry is changed in the loop that calculates $M_2 + M_1$.

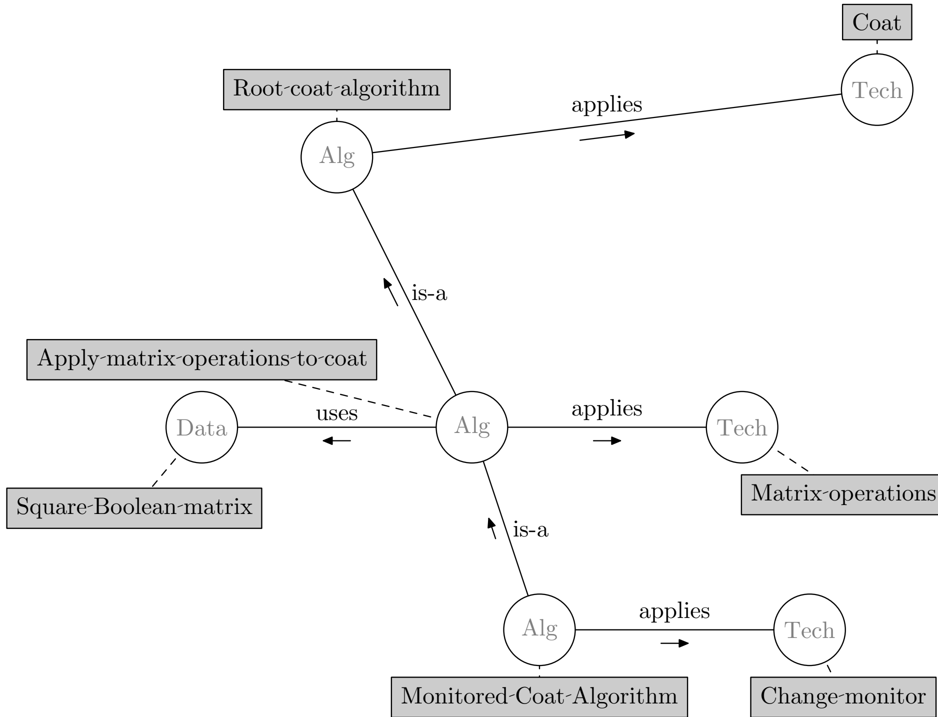


Figure 14.2.1: Topics related to the monitored coat algorithm

14.2.2 Verification

The operations performed by this algorithm are the same as the operations performed by Algorithm 12.4.1 (Prosser). In Section 12.4.3 it was shown that the value of M_2 converges to M^+ when the operations of this algorithm are performed and that $M_2 = M^+$ after a finite number of iterations. As soon as M_2 has assumed the value of M^+, further iterations will not change its value. At this stage the change monitor detects that no changes were made during the last pass and that iteration should stop. Consequently the algorithm terminates with M_2 containing the desired value.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 14.2.1: The monitored coat algorithm

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^+$
var $M_0 \in \mathbb{B}[n,n]$
$M_1 \in \mathbb{B}[n,n]$
$M_2 \in \mathbb{B}[n,n]$
$M_3 \in \mathbb{B}[n,n]$

$M_0, M_1, M_2, M_3 := \Phi(R), \Phi(R), \Phi(R), \mathbb{O}_n$
do $M_3 \neq M_2 \rightarrow$
$M_3 := M_2$;
$M_1 := M_1 \times M_0$;
$M_2 := M_2 + M_1$
od
{**Post:** $M_2 = M_3, \Psi(M_2) = R^+$}

14.2.3 Complexity

As in Baker's algorithm in Section 14.1.3 (Page 200), the guard of the outer loop and the first instruction in the body of this loop are both $O(n^2)$ operations. The rest of the body is the same as the body of Prosser's algorithm. In Section 12.4.4 (Page 178) it was established that the complexity of the body of the loop is $\Theta(n^3)$. Thus each iteration has a complexity of $\Theta(n^2 + n^2 + n^3) = \Theta(n^3)$.

The application of a more efficient change monitor, as is done in the implementations provided, has no impact on this complexity. The application of the more efficient change monitor fuses the first two loops in the implementation with the second loop in the body of Prosser's algorithm. Instead of having two loops with $\Theta(n^2)$ operations each as well as a loop with $\Theta(n^3)$ operations, the implementation has only one loop with $\Theta(n^3)$ operations. The overall complexity is the same.

In Section 12.4.3 it is shown that the algorithm will reach the state where $M_2 = M^+$ in at most $n - 2$ iterations. Consequently it will take at most $n - 1$ iterations for Algorithm 14.2.1 (CoatMonitor) to terminate as it needs an extra iteration to verify that no changes occurred during its execution.

Thus, the complexity of the algorithm is $O((n-1) \times n^3) = O(n^4 - n^3) = O(n^4)$. The TM specification of this complexity in the TM of TCA is given in Listing C36.

14.2.4 Position in a derivation tree

This algorithm was identified by observing a lack of symmetry in the existing derivation tree discussed in the beginning of this section. This is one of the three

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

events that can lead to the discovery of algorithms mentioned in Section 10.2. The discovery of algorithms in this manner is a prominent benefit of the classification of algorithms in terms of derivation hierarchies. As can be seen in Figure 14.2.4 this algorithm is derived from an abstraction of Prosser's algorithm. This derivation applies the *change monitor* technique where Prosser's algorithm applies the *safe upper bound* technique.

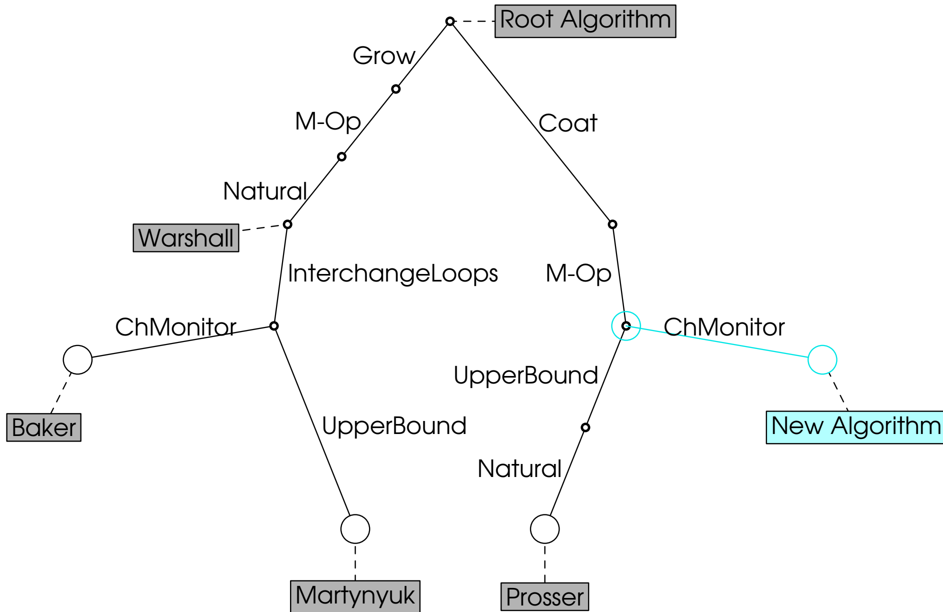


Figure 14.2.4: Derivation tree showing Prosser, Warshall, Martynyuk and Baker

14.3 Summary

This chapter investigates the derivation of algorithms that apply a so-called technique *change monitor* technique, instead of the *safe upper bound* technique used in the algorithms discussed in Chapter 13. One of these, namely Algorithm 14.1.1 (Baker) was published by Baker [17] in 1962 while the other is, to my knowledge, published here for the first time. The discovery of this algorithm serves as an example of one of the events listed in Section 10.2 that may lead to the identification of an algorithm that has not been described before.

The artifacts required to add these algorithms to the TM of TCA were created in this chapter, namely ones corresponding to correctness verification and determining the complexity of the algorithm.

The next chapter investigates the use of *loop fusion* as algorithmic technique. This technique is used to improve the performance of Algorithm 12.4.1 (Prosser) and Algorithm 14.2.1 (CoatMonitor) since these algorithms have nested loops that can be fused.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 15

Loop Fusion

One of the most common loop optimising transformations is called *loop fusion*. When applying this transformation, the bodies of two adjacent loops are combined to share the same loop counters.

It is not always possible to fuse an arbitrary pair of adjacent loops. Loop fusion may only be applied if there are no dependencies between the operations performed in the adjacent loops. Pouchet *et al.* [200] acknowledge that manual optimisations involving loop transformations are generally far more effective than known automatic optimisations of this kind. Similarly Loveman [157] remarks that program improvements are often subject to programmer direction and guidance and need not be limited to the *behind the scenes* activities done by compilers. In this chapter the applicability of loop fusion on the adjacent loops in Prosser's algorithm (Section 12.4) and in the monitored coat algorithm (Section 14.2) are discussed.

15.1 The fused coat algorithm

The implementation of Algorithm 12.4.1 (Prosser) has two adjacent loops inside its outer loop. The first of these loops calculates $M_1 \times M_0$, and the second one adds the result of this calculation to M_2. Because these loops both iterate similarly over all the entries of the matrices on which they operate, the fusion of these loops is an obvious optimisation step to consider.

The fusion of these loops is straightforward. Although it seems as if the second loop needs the result after completing the entire first loop, the completion of the first loop is not actually needed. Closer investigation reveals that the value of $M_1[a, b]$ is not altered during any iteration of the first loop that alters $M_1[j, k]$ where $a \neq j$ and $b \neq k$. This observation leads to conclusion that the final value of $M_2[j, k]$, as it would appear after completion of the first loop, is already determined within the first loop at the point in the iteration when it is needed in the second loop. It is likely that this optimisation will be applied by most optimising compilers when compiling an implementation of Prosser's algorithm as it is specified in Section 12.4.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Figure 15.1.1 shows the topics related to this algorithm in the TM of TCA. The algorithmic technique applied for the optimisation is called *loop fusion*. The TM definition of this technique in the TM of TCA is included in Listing C8. Listing C21 is the TM specification of this algorithm in the TM of TCA. The specification pointing to implementations of the fused coat algorithm in the TM of TCA can be found in Listing C52.

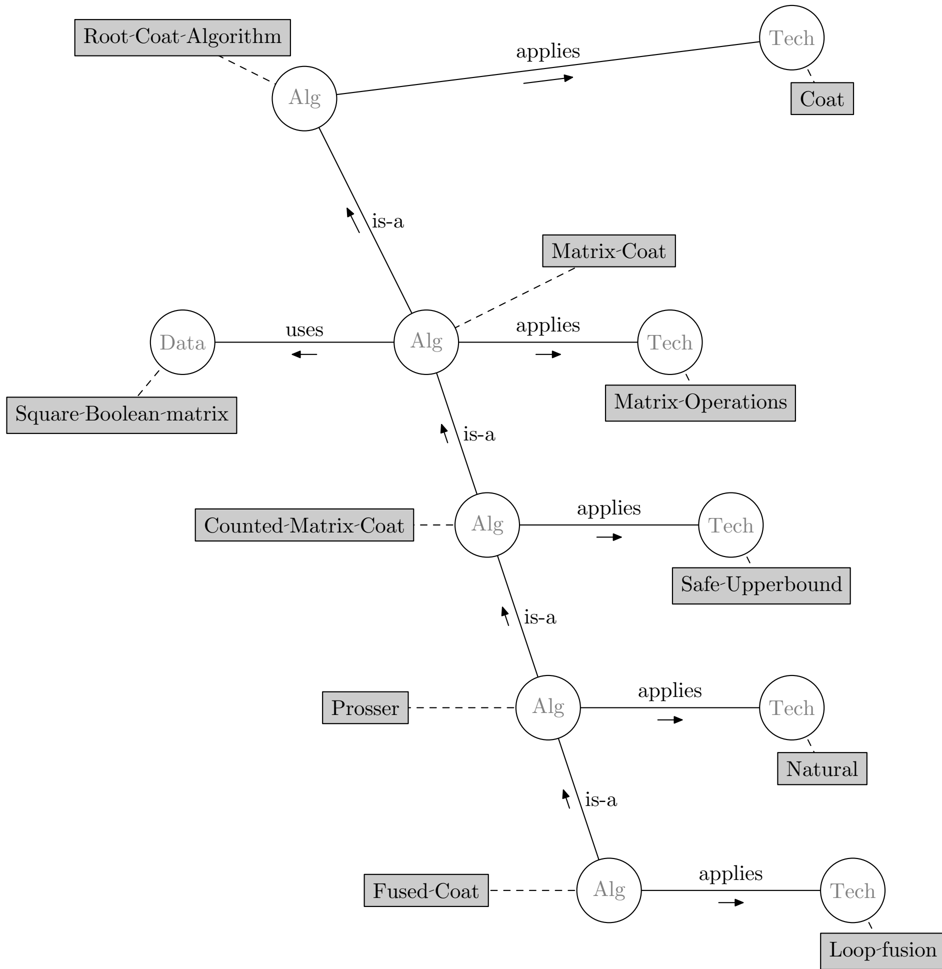


Figure 15.1.1: Topics related to the fused coat algorithm

When taking advantage of this optimisation, the resulting algorithm is an optimised version of Algorithm 12.4.1 (Prosser) that, to my knowledge, has not yet been published.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The parent of this algorithm is a compound algorithm that uses sub-algorithms for matrix multiplication and matrix addition. The derived algorithm does not perform these sub-algorithms separately. Here the outer loops of these sub-algorithms are fused resulting in a new routine. This new routine is in theory a sub-algorithm, but is not considered as such. This is because this routine is not usable to the same general level as the sub-algorithms that were fused here to form the new operation.

Algorithm 15.1.1: Fused Coat Algorithm

---

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^+$
var $M_0 \in \mathbb{B}[n, n]$
$M_1 \in \mathbb{B}[n, n]$
$M_2 \in \mathbb{B}[n, n]$

$M_0, M_1, M_2 := \Phi(R), \Phi(R), \Phi(R)$
for $i : i \in \mathbb{N}_{n-2} \rightarrow$
  for $j, k : j, k \in \mathbb{N}_n \rightarrow$
    $M_1[j,k] := \langle \exists t : t \in \mathbb{N}_n : M_1[j,t] \wedge M_0[t,k] \rangle;$
    $M_2[j,k] := M_2[j,k] \vee M_1[j,k]$
  rof
rof
$\{$**Post:** $\Psi(M_2) = R^+\}$

15.1.1 Verification

$\{M_1[a,b] \wedge a \neq j \wedge b \neq k\}$ is an invariant of the first inner loop of Prosser's algorithm. This is the loop that is applied in this algorithm to calculate the value of each entry in $M_1 \times M_0$. Statement 15.1 shows the calculation applied in this loop. The statement in the body of this loop alters the value of $M_1[j,k]$ and has no impact on the value of $M_1[a,b]$ for all other entries of M_1.

$$\textbf{for } j, k : j, k \in \mathbb{N}_n \rightarrow M_1[j,k] := \langle \exists t : t \in \mathbb{N}_n : M_1[j,t] \wedge M_0[t,k] \rangle \textbf{ rof} \quad (15.1)$$

15.1.2 Complexity

This algorithm is the same as Prosser's algorithm (Algorithm 12.4.1 (Prosser)), except that it fuses the two adjacent inner loops. It was shown in Section 12.4.4 that the complexity of executing the loops one after the other in Prosser's algorithm is $\Theta(n^3 + n^2) = \Theta(n^3)$. In this algorithm these loops are fused resulting in a single loop with complexity $\Theta(n^3)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The loop is repeated $n-2$ times. The complexity of executing this loop can therefore be expressed as $\Theta((n-2)\times n^3) = \Theta(n^4 - 2\times n^3) = \Theta(n^4)$. Listing C35 shows the TM specification of the complexity of the fused coat algorithm.

15.1.3 Position in a derivation tree

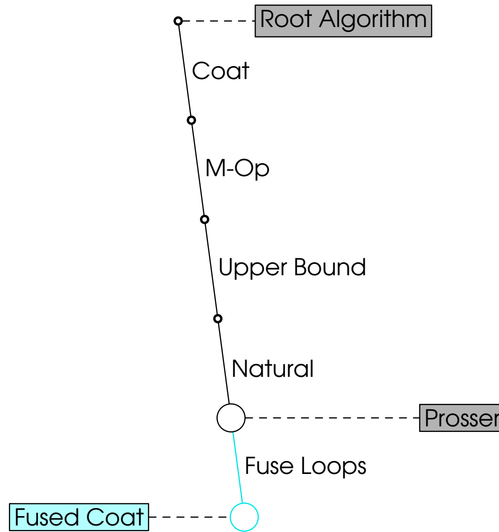


Figure 15.1.3: Derivation tree of the fused coat algorithm

As can be seen in Figure 12.4.2, Prosser's algorithm appears as a leaf node in a derivation tree. Furthermore, the fused coat algorithm applies all the algorithmic techniques applied by Prosser's algorithm and can be derived from Prosser's algorithm by applying loop fusion. The fused coat algorithm can thus be positioned in the derivation tree on the branch extending from Prosser's algorithm. Figure 15.1.3 shows this.

15.2 A neat derivation of the monitored coat algorithm

Figure 15.2.4 in Section 15.2.4 shows a portion of the derivation tree containing the fused coat algorithm along with the monitored coat algorithm. These algorithms are respectively discussed in Sections 15.1 and 14.2. The tree is asymmetric if the derivation shown in blue is omitted. The observed gap in the derivation tree when this branch is omitted, leads to the identification of an elegant new algorithm. This new algorithm is the subject of this section.

The proposed new algorithm is called the neat coat algorithm. The topics related to this algorithm are shown in Figure 15.2.1.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

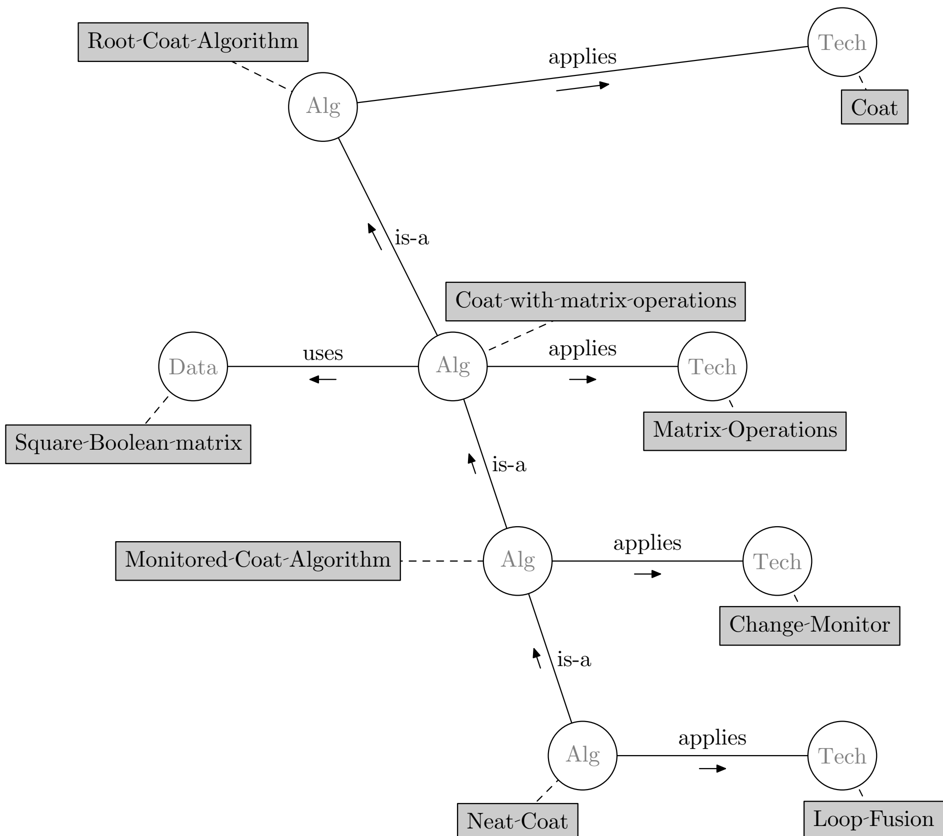


Figure 15.2.1: Topics related to the neat coat algorithm

The neat coat is derived from the monitored coat algorithm, similarly to the way in which the fused coat algorithm was derived from Prosser's algorithm. The transformation of monitored coat algorithm into this new algorithm involves the elimination of two of four adjacent loops of the monitored coat algorithm and the fusion of the remaining two loops.

The four steps in the body of Algorithm 14.2.1 (CoatMonitor) are each performed by iterating over the entries in two $n \times n$ Boolean matrices. These loops are:

1. The loop to calculate the value of $M_3 \neq M_2$ in the guard of the outer loop,
2. the loop to update the value of M_3 by executing the assignment $M_3 := M_2$,
3. the loop that calculates $M_1 \times M_0$ and assigning it to M_1, and
4. the loop that calculates $M_2 + M_1$ and assigning it to M_2.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The fusion of the last two loops is the same loop fusion that was applied to derive the fused coat algorithm from Prosser's algorithm.

Further optimisation is now achieved by the elimination of the first two loops. Instead of monitoring the matrix as a whole for changed values, changes are registered by means of a flag that is raised when an entry in M_2 is changed. Thus the use of M_3 is replaced by a flag called b. This flag is initially set. The outer loop of the algorithm is repeated as long as this flag is set. It is cleared at the beginning of the execution of each iteration and set each time an entry in M_2 is changed. The value of a given entry $M_2[j,k]$ changes only if it was *false* before the iteration starts and the value of $M_1[j,k]$ as calculated in the loop is *true*. In the algorithm the condition for change need to be observed in order to raise the flag if needed. If the test fails there, is no need to do any further calculations or assignments. If the test passes, the value of $M_2[j,k]$ is updated and the flag is raised.

15.2.1 Specification

Algorithm 15.2.1: Neat Coat Algorithm

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^+$
var $M_0 \in \mathbb{B}[n,n]$
$M_1 \in \mathbb{B}[n,n]$
$M_2 \in \mathbb{B}[n,n]$
$b = 1 \in \mathbb{B}$

$M_0, M_1, M_2 := \Phi(R), \Phi(R), \Phi(R)$
do $b \rightarrow$
$b = 0;$
for $j,k : j,k \in \mathbb{N}_n \rightarrow$
$M_1[j,k] := \langle \exists t : t \in \mathbb{N}_n : M_1[j,t] \wedge M_0[t,k] \rangle;$
if $M_2[j,k] \vee \neg M_1[j,k] \rightarrow$ **skip**
$[\!] \; \neg M_2[j,k] \wedge M_1[j,k] \rightarrow M_2[j,k], b := 1, 1$
fi
rof
od
$\{$**Post:** $\Psi(M_2) = R^+\}$

I call this algorithm the neat coat algorithm because it neatly avoids performing unnecessary calculations both by fusing loops and by applying a change monitor.

The TM definition of the neat coat algorithm in the TM of TCA is shown in Listing C27 while Figure 15.2.1 visualises the topics related to this algorithm. The TM specification pointing to implementations of the neat coat algorithm in the TM of TCA can be found in Listing C54.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Pre-condition		Post-condition			
		Statement 15.2	Statement 15.3		
$M_1[j,k]$	$M_2[j,k]$	$M_2[j,k]$	Command Executed	$M_2[j,k]$	b
0	0	0	First	0	unchanged
0	1	1	First	1	unchanged
1	0	1	Second	1	raised
1	1	1	First	1	unchanged

Table 15.2.2: Program trace of a statement in the neat coat algorithm

15.2.2 Verification

There are two differences between Algorithm 15.2.1 (CoatNeat) and the fused coat algorithm specified in Algorithm 15.1.1 (CoatFuse).

The one difference is the implementation of a change monitor to control termination of the loop in the neat coat algorithm where the fused coat algorithm uses a safe upper bound for this purpose.

The other difference is that Statement 15.2 is used in the fused coat algorithm in the place where Statement 15.3 is used in the neat coat algortithm.

$$M_2[j,k] := M_2[j,k] \vee M_1[j,k] \tag{15.2}$$

$$\begin{array}{l} \textbf{if} \\ \quad M_2[j,k] \vee \neg M_1[j,k] \ \rightarrow \ \textbf{skip} \\ \quad [\!] \ \neg M_2[j,k] \wedge M_1[j,k] \ \rightarrow M_2[j,k], b := 1, 1 \\ \textbf{fi} \end{array} \tag{15.3}$$

To verify that the result of Statement 15.3 is equivalent to the result of Statement 15.2, the four possible pre-conditions for the statement and their corresponding post-conditions are determined. This is shown in Table 15.2.2. This program trace shows that the state of $M_2[j,k]$ after executing both statements are the same for all possible starting states. This shows that the replacement of Statement 15.2 with Statement 15.3 preserves the correctness of the algorithm.

In the same table, it is verified that the change monitor produces the expected outcome. The flag b is cleared before executing each iteration of the outer loop. It can be seen that b is only raised in the case where the value of $M_2[i,j]$ was changed as a result of executing the statement. Since this statement is the only statement in the loop that has an effect on b, it can be concluded that b is raised after an iteration only if one or more entries in M_2 has changed during the execution of the body of the loop during the iteration.

In Section 15.1.1 it was shown that in Algorithm 15.1.1 (CoatFuse), the value of M_2 converges to M^+ when performing its operations. It is also observed that $M_2 = M^+$ after a finite number of iterations. Here I have shown that this algorithm

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

performs operations that produce equivalent results. Therefore the same applies to M_2 in this algorithm. As soon as M_2 has assumed the value of M^+ further iterations will not change its value and consequently b will not be raised causing the outer loop to terminate with M_2 containing the desired value.

15.2.3 Complexity

This algorithm is the same as Algorithm 14.2.1 (CoatMonitor), except that it fuses the adjacent inner loops. It was shown in Section 14.2.3 that the complexity of executing the loops one after the other in Algorithm 14.2.1 (CoatMonitor) can be expressed as $\Theta(n^2 + n^2 + n^3) = \Theta(n^3)$. In this algorithm these loops are fused. The resulting loop is a nested $n \times n$ loop with a body that consists of the calculation of $M_1[j,k]$ and a conditional assignment. The calculation of $M_1[j,k]$ is an operation of complexity $\Theta(n)$, while the complexity of the assignment is $\Theta(1)$. Thus, the complexity of the body of the loop is $\Theta(n+1) = \Theta(n)$. The total complexity of the loop is thus $\Theta(n^2 \times n) = \Theta(n^3)$.

The loop is repeated until the transitive closure is reached. In Section 12.4.4 it is established that the number of iterations needed is the length of the longest path in the relation, which is $n - 2$ times. Thus the transitive closure will be reached after at most $n - 2$ iterations. The complexity of executing this loop can therefore be expressed as $O((n-2) \times n^3) = O(n^4 - \ 2 \times n^3) = O(n^4)$. As expected, the fusion of the loops does not effect the general complexity of the algorithm. It may, however, impact positively on the performance of the algorithm as the complexity is no longer tight.

Listing C37 shows the TM specification of the complexity of the neat coat algorithm in the TM of TCA.

15.2.4 Position in a derivation tree

As is the case with Algorithm 14.2.1 (CoatMonitor), this algorithm is also an example of an algorithm that was discovered by observing a lack of symmetry in the existing derivation tree. This discovery strengthens the benefit of maintaining the derivation hierarchy in the TM of TCA. The position of Algorithm 15.2.1 (CoatNeat) is shown in Figure 15.2.4.

As can be seen in the figure, this algorithm is derived from the monitored coat algorithm by applying loop fusion.

15.3 Summary

This chapter investigates the derivation of algorithms that apply a technique called *fuse loops*. The algorithms that are derived in this chapter, namely Algorithm 15.1.1 (CoatFuse) and Algorthm 15.2.1 (CoatNeat) are respectively derived from an algorithm that is discussed in Chapter 12 and from one that is discussed in Chapter 14. Both algorithms derived in this chapter have, to my knowledge, not been mentioned in the literature previously.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

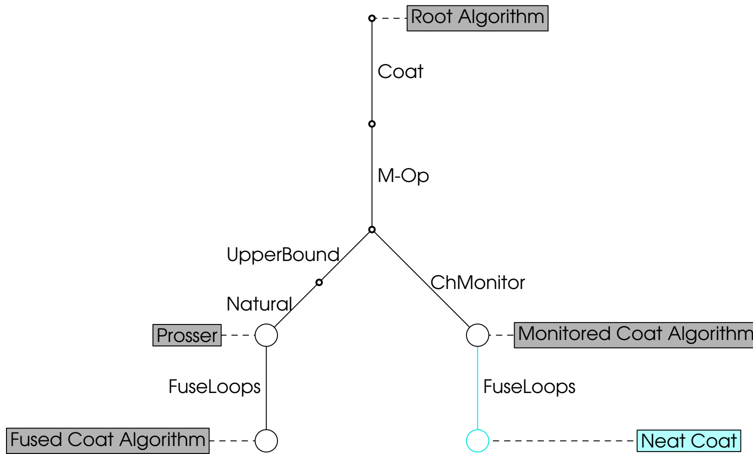


Figure 15.2.4: Derivation tree showing new variants of Prosser's algorithm

The discovery of these algorithms is not ground breaking, as the technique they apply is one that is often automatically applied by optimising compilers. Some programmers may not agree that they should be distinguished from their parents in the derivation tree. The explicit inclusion of these algorithms as distinct variants of Algorinthm 12.4.1 (Prosser) and Algorithm 14.2.1 (CoatMonitor) contributes to the comprehensiveness of the TM of TCA described in this thesis. Their inclusion may lead to deeper understanding of these algorithms and is useful for suggesting possible optimised versions of these algorithms in software construction.

The artifacts required to add these algorithms to the TM of TCA were created in this chapter, namely ones corresponding to correctness verification and determining the complexity of the algorithm.

The next chapter investigates the use of *loop tiling* as an algorithmic technique. This technique results in an alternative to Algorithm 12.5.1 (MatrixGrow). Loop tiling uses a deterministic processing order that is different from the natural processing order used by other algorithms derived from Algorithm 12.5.1 (MatrixGrow) so far discussed. In so doing, loop tiling improves paging behaviour and avoids memory thrashing.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 16

Loop Tiling

A technique called *loop tiling* partitions a loop's iteration space into smaller chunks or blocks, so as to help avoid overhead related to loading data. It manipulates data in such a way that multiple operations that are to be performed on the same elements can be performed while the elements remain accessible. The technique is used when the data needed for manipulation cannot fit into main memory. It can also be applied to manage how portions of the data are loaded into cache memory to avoid cache thrashing. This technique is also known as *strip mining* or *loop blocking*. This technique is widely used especially in context of parallelisation of algorithms as can be seen in Wakatani and Wolfe [249].

The amount of cache-swapping and processor-memory traffic generated by the execution of algorithms has been observed to be a bottleneck for achieving high performance in many applications [222]. Lam *et al.* [153] purports that the application of loop tiling can produce significant performance speedup when processing large matrices.

In this chapter the term *memory* is used generically. It may refer to cache memory in the case where the algorithm is used to boost performance through active management of cache memory usage. It may also refer to main memory in the case where the algorithm is used to enable the calculation of the transitive closure of a relation where the data is stored on a secondary memory device and the relation is too large to fit the entire matrix that represents the relation into main memory.

Penner and Prasanna [192] point out that in certain transitive closure algorithms, the irregularity of operations poses unique problems that can complicate tiling strategies. This applies in particular to algorithms that calculate transitive closure by means of manipulating the adjacency matrix of the relation. Implementations of transitive closure algorithms often have poor locality of reference.

If the matrices are small, both the row and column elements needed for an operation can be contained in the data cache, and the processor can run at full speed. When the size of the matrices increases, the locality of reference of such operations deteriorates to a point where the data can no longer be maintained in the cache. In fact, the natural access pattern of these operations often generates a

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

predictable thrashing pattern in cache memory. For example; when an operation has to be performed involving the elements of a row which is too large to fit into cache, it is likely that during the operation involving elements in parts of the row which are not in close proximity with one another, the initially accessed elements may be forced out of cache so as to gain access to other elements and then repeating the access pattern again for the same data.

Algorithms that are derived from Algorithm 11.4.1 (Grow) — most notably Algorithm 12.6.1 (Warshall) — are particularly vulnerable in this regard. Algorithms that are derived from Algorithm 13.1.3 (GrowRow) avoid this problem at the cost of increased complexity. This warrants research to find algorithms derived from Algorithm 12.6.1 (Warshall) and Algorithm 13.1.3 (GrowRow) that exhibit better paging behaviour. The algorithms that are discussed in Chapters 12, 13, 14 and 15 are likely to suffer from thrashing because they follow a processing order that is related to the numeric progression of row and column numbers rather than seeking ways to minimise cache-swapping and processor-memory traffic. By way of contrast, the algorithms in the new branch created here typically follow a deterministic processing order different from the natural processing order so as to improve paging behaviour.

The TM specification of the algorithmic technique applied by the algorithms in the branch discussed in this chapter, called *loop tiling*, is included in Listing C8. In this chapter the technique is introduced and an abstract algorithm applying the technique is discussed.

Concrete algorithms applying the technique that are part of this branch of the derivation tree are discussed in Chapter 17. These are an algorithm proposed by Warren [251] as well as two algorithms described by Agrawal and Jagadish [2].

The ability to control the usage of different levels of cache memory coupled with interest in exploiting the parallel processing capabilities of modern computers, led to the discovery of a substantial number of multiprocessor solutions that are further extension of this branch in the derivation tree but beyond the scope of this thesis. Examples of algorithms in this category are those by Agrawal and Jagadish [3], Cappello *et al.* [45], Griem and Oliker [105], Kung *et al.* [152], Milovanović *et al.* [171], Pagourtzis *et al.* [185], Penner and Prasanna [192], Scheiman and Cappello [214] and Valduriez and Khoshafian [241].

16.1 Specifying a valid ordering

The iteration space of Algorithm 12.5.1 (MatrixGrow) is $\Phi(R) = M$. The application of loop tiling to this iteration space changes the order in which the entries in M are processed in a way that is likely to reduce overhead related to loading data while maintaining the correctness of the algorithm. In most cases it increases the complexity, however, the performance gain that can be attributed to reduction of cache thrashing seems to cancel out the impact of the increased complexity. Here a way to specify an ordering is described and requirements that ensure that a specified order maintains correctness are reviewed. Finally a relaxation

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

of constraints that is allowed is discussed. It may increase performance without compromising correctness.

A partition P of a matrix $M \in \mathbb{B}[m, n]$, denoted by $P \vdash M$, is a sequence of submatrices of M that are mutually exclusive and exhaustive with respect to M. Partitions of Boolean matrices, as defined for use in this thesis, are discussed in more detail in Section 4.5. Submatrices are per definition sequences, the entries in submatrices are ordered. Likewise, a partition is a sequence. Thus the submatrices comprising the partition are also ordered. The specification of a partition of M thus describes an ordering of the entries in M.

The order in which an algorithm processes the entries in its iteration space M, can uniquely be defined in terms of the specification of a partition of M. A description of the construction of a specific partition of M is called the *tiling strategy* of the algorithm that uses the partition. An algorithm that applies such a tiling strategy iterates over the submatrices in the partition in the specified order. When processing each of the submatrices in the partition, the order of the elements in the submatrix determines the processing order of the entries in M. To show the correctness of a tiling algorithm it has to be verified that its tiling strategy indeed constructs a partition on M and that the processing of the elements as specified by this partition applies a processing order that ensures correctness of the algorithm.

16.2 Processing order constraints

Agrawal and Jagadish [2] derived constraints that should hold for the processing order of the entries in the adjacency matrix when executing algorithms derived from Algorithm 11.4.1 (Grow). If the processing of the entries complies with these restrictions, M^+ is reached after each of the entries in the adjacency matrix has been processed once.

The following are the constraints, exactly as originally formulated by these authors:

1. for all i, j, k, processing of the element (i, k) precedes the processing of $(i, j) \iff k < j$.

2. for all i, j, k, processing of the element (j, k) precedes the processing of $(i, j) \iff k < j$.

They explain these constraints as follows:

> *The first constraint effectively requires that elements of a row must be processed from left to right. Similarly, the if part of the second constraint requires that before processing (i, j), all elements of row j left of the diagonal must have been processed. The only if part of this constraint requires that any element on row j to the right of the diagonal can only be processed after all elements on column j have been processed.*

Although this explanation captures the meaning of the expressions that describe the constraints, the expressions are ambiguous. To address the ambiguity,

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

an alternative set of constraints using a different notation is introduced. It is then shown that the alternate set of constraints are sufficient to guarantee the correctness of an algorithm that complies with the specified constraints. The notation as well as the concepts that are used in this formulation of the constraints are introduced in this thesis in Section 4.5 specifically for this purpose. To be kind to the reader the next paragraph recaps the essence of the relevant content.

As was explained in Section 4.5, a partition $P \vdash M$ is a sequence of submatrices of M. The order of the entries in the submatrices is determined by the ordering of the dummy variables in the specification of the submatrix. For example the sequence $\langle\!\langle\, i,j \mid R(m_{ij}) \mid m_{ij}\,\rangle\!\rangle$ contains a subset of the entries of M in row major order and the sequence $\langle\!\langle\, j,i \mid R(m_{ij}) \mid m_{ij}\,\rangle\!\rangle$ contains the same entries but in column major order. See Section 4.4.1 for more about this convention.

The following symbols are introduced for frequently used phrases in the arguments that follow:

$\lhd$ is used for the phrase *is/are processed before*
$\rhd$ is used for the phrase *is/are processed after*

The description of the entries in M which should be processed before a given entry m_{xy} may be processed, can thus be denoted in terms of a submatrix of M. The predicate defining the submatrix should be specified in terms of how the indices of the entries in the submatrix relate to the indices of the given entry m_{xy}. The order of dummy variables used in the definition of such submatrix then imposes the required ordering on the elements in the defined submatrix of M.

The following three subsections each define one constraint in terms of the above defined concepts and notation. For ease of referring the the constraints, the constrains are respectively called the ***primary constraint***, the ***secondary constraint*** and the ***column constraint***.

16.2.1 Primary constraint

$$\langle\!\langle\, i,j \mid (i = x) \wedge (j < y) \mid m_{ij}\,\rangle\!\rangle \lhd m_{xy}$$

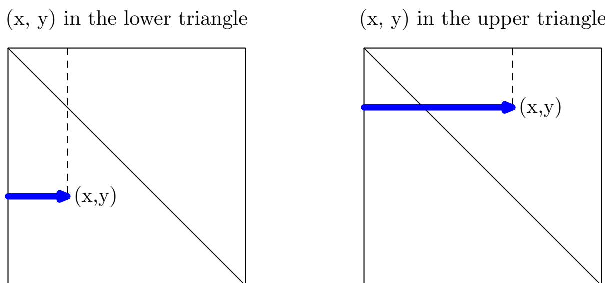


Figure 16.2.1: Primary constraint for processing an entry m_{xy}

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Figure 16.2.1 is a reformulation the first constraint specified by Agrawal and Jagadish [2] in terms of a submatrix containing all entries that have to be processed before m_{xy}. The order of the dummy variables specifies that the order of the elements in the submatrix should be the same as the order of the elements in M when following row major order in M. The constraint limits the entries to one specific row. The order of the entries in this submatrix is therefore the same as the order of the columns in M. The entries in this submatrix should thus be processed in sequential order from left to right.

The submatrix defined by the primary constraint is shown using a bold blue line in the diagrams in the figure. For the sake of completeness, the figure shows two examples. One where m_{xy} is in the lower triangle and one where m_{xy} is in the upper triangle. The specification for both cases is the same.

16.2.2 Secondary constraint

$$\langle\!\langle\, i, j \mid (i = y) \wedge (j < y) \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy}$$

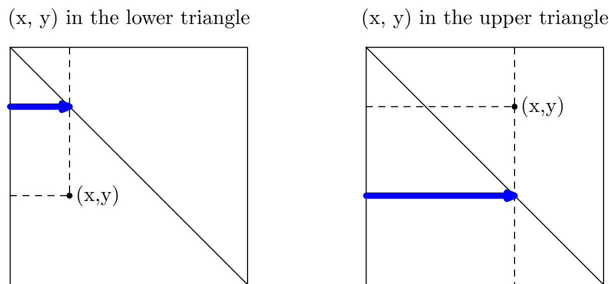


Figure 16.2.2: Secondary constraint for processing an entry m_{xy}

The secondary constraint is specified in Figure 16.2.2. It specifies what Agrawal and Jagadish [2] attempted to specify in the *if*-part of their second constraint in terms of a submatrix. The order of the dummy variables specifies that the order of the elements in the submatrix should be the same as the order of the elements in M when following row major order in M. The constraint limits the entries to one specific row. The order of the entries in this submatrix is therefore the same as the order of the columns in M. The entries in this submatrix should thus be processed in sequential order from left to right. The submatrix contains entries in the row corresponding with the column of m_{xy}. It is limited to entries in this row that are below the main diagonal of M.

The submatrix defined to specify this constraint is shown using a bold blue line in the diagrams in the figure. The figure shows two examples. One where m_{xy} is in the lower triangle and one where m_{xy} is in the upper triangle. Although the specification for both cases are the same, the result in each case is slightly different. In the case where m_{xy} is in the lower triangle, the specified submatrix is above m_{xy} while it is below m_{xy} in the case where m_{xy} is in the upper triangle.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

16.2.3 Column constraint

$$\langle\!\langle\, i, j \mid (j = x) \wedge (x < y) \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy}$$

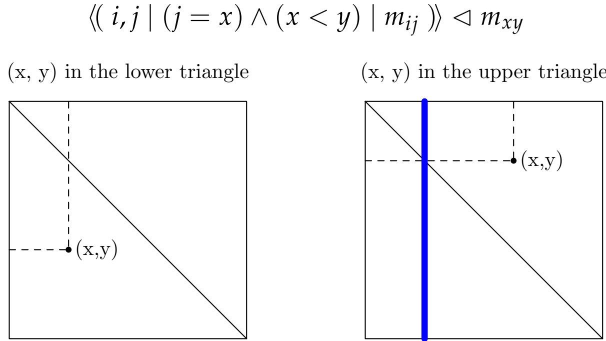


Figure 16.2.3: Column constraint for processing an entry m_{xy}

The column constraint is shown Figure 16.2.3. It is the specification of the *only if*-part of the second constraint that was specified by Agrawal and Jagadish [2]. The submatrix defined to specify the constraint contains all the entries on the column corresponding with the row of the entry. The order of the dummy variables specifies that the order of the elements in the submatrix should be the same as the order of the elements in M when following row major order in M. The constraint limits the entries to one specific column. The order of the entries in this submatrix is therefore in sequential order from top to bottom; the same as the order of the rows in M.

This constraint applies only to entries of M that are above the diagonal. The submatrix defined to specify this constraint is shown using a bold blue line in the diagrams in the figure. The predicate $x < y$ in the specification renders the submatrix empty if m_{xy} is below the diagonal. As before, the figure shows two examples for the sake of completeness. If m_{xy} is below the diagonal, this submatrix of entries that have to be processed before m_{xy}, as specified here, is empty, hence the absence of a blue portion in the left diagram.

16.2.4 A consequence

Lemma 16.2.4 shows that compliance with the secondary constraint with respect to m_{xy} implies that $\langle\!\langle\, i, j \mid i < y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy}$. Thus compliance with the secondary constraint implies that the elements in the triangle below the main diagonal and above the row corresponding with the column of the element in question are processed before the element in question.

The submatrix containing entries of which processing is required before m_{xy} as described by Lemma 16.2.4 is illustrated in Figure 16.2.4. The submatrices in question are shaded in the diagrams in this figure.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & \langle\!\langle\, i,j \mid i = y \wedge j < y \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy} \\
\Rightarrow & \left\{ \begin{array}{l} \text{Compliance with secondary constraint} \\ \text{with respect to } m_{ya} \text{ where} \\ m_{ya} \in \langle\!\langle\, i,j \mid i = y \wedge j < y \mid m_{ij} \,\rangle\!\rangle \end{array} \right\} \\
 & \langle\!\langle\, i,j \mid i = a \wedge j < a \wedge a < y \mid m_{ij} \,\rangle\!\rangle \lhd m_{ya} \lhd m_{xy} \\
\Rightarrow & \{ \lhd \text{ is transitive} \} \\
 & \langle\!\langle\, i,j \mid i = a \wedge j < a \wedge a < y \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy} \\
\Rightarrow & \{ i = a \wedge a < y \Rightarrow i < y, i = a \wedge j < a \Rightarrow j < i \} \\
 & \langle\!\langle\, i,j \mid i < y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy}
\end{array}
$$

Lemma 16.2.4: $\langle\!\langle\, i,j \mid i < y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy}$

The diagram on the left shows the submatrix containing the entries that should be processed before a given entry, if the given entry is below the diagonal. The diagram on the right shows the submatrix containing the entries that should be processed before a given entry if the given entry is above the diagonal.

$$\langle\!\langle\, i,j \mid i < y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy}$$

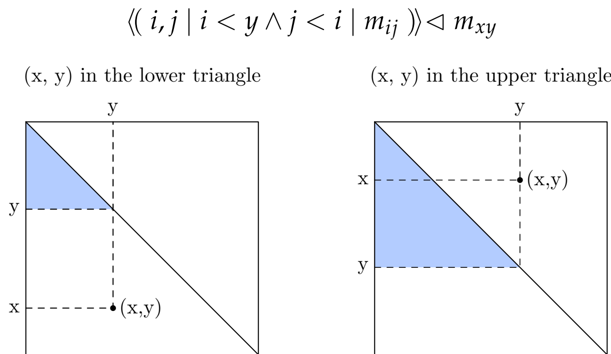


Figure 16.2.4: Submatrices that has to be processed before processing m_{xy}

The order of the dummy variables specifies that the order of the elements in the submatrix used in Lemma 16.2.4 should be the same as the order of the elements in M when following row major order in M. This ordering was also used when specifying the constraints. While the ordering in the constraints are limited to one row only, the ordering would be the same if the order of the elements in the submatrix had been specified as being in column major order. The proof in Lemma 16.2.4 when applied with the definition of the constraints using column major order can be applied to show that $\langle\!\langle\, j,i \mid i < y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd m_{xy}$.

It can therefore be concluded that compliance with the secondary constraint with respect to m_{xy} implies that all entries in the lower triangle of M and above $\langle\!\langle\, i,j \mid i = y \mid m_{ij} \,\rangle\!\rangle$ is processed before m_{xy} either in column major order or in row major order.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

16.3 Lemmata related to processing order constraints

This section prove lemmata related to the processing order constraints that are specified in Section 16.2. These lemmata are needed to show the correctness of algorithms derived from the algorithm that is introduced in Section 16.4.

16.3.1 Processing the elements in $M_{\mathbb{I}}$

When determining the transitive closure of a relation represented by the matrix M using the processing steps specified for grow algorithms, the entries in the main diagonal ($M_{\mathbb{I}}$) need not be processed. The omission of processing of these entries is allowed because processing an element $m_{ii} \in M_{\mathbb{I}}$ in a grow algorithm boils down to deciding whether or not to add the ith row of M to itself, and if deemed necessary to perform this operation. Since both elements of $\mathbb{B}$ are idempotent with respect to $\vee$, performing the operation $M[\star, j] + M[\star, j]$ would leave $M[\star, j]$ unchanged. Consequently the decision whether or not to perform the operation as well as actually performing this operation are redundant in these algorithms.

Despite this fact, this operation is anyway performed by many grow algorithms. This is mainly because often the overhead to avoid performing these operations does not warrant the gain of not performing them. It may also add unnecessary complexity to otherwise elegant solutions.

When performing loop tiling, one can capitalise on this observation when one creates a partition of $M - M_{\mathbb{I}}$ rather than a partition of M.

16.3.2 The trivial case of Warren's lemma

Lemma 16.3.3 is a generalisation of a lemma that Warren [251] proposed and proved in connection with the correctness of his algorithm to calculate the TC of a relation. Warren's algorithm is discussed in Section 17.1. I call this lemma Warren's Lemma. It states that for every acyclic path $A = \langle\!\langle\ i \mid i \in \mathbb{N}_{k+1} \mid a_i\ \rangle\!\rangle$ in R with the head of the path denoted by a_0, it follows that Predicate 16.1 holds after Algorithm 16.4.1 (TileSkeleton) has processed the entries of the sub-matrix $\langle\!\langle\ i, j \mid (i = a_0) \wedge (j < a_0) \mid m_{ij}\ \rangle\!\rangle$.

$$M[a_0, a_k] \vee \langle \exists t \mid 0 < t < k \mid M[a_0, a_t] \wedge a_t > a_0 \rangle \tag{16.1}$$

Warren's Lemma asserts that after processing the entries left of the diagonal in the row that contains the head of a given path, either the tail of the path is already a direct successor of a_0 or an entry in the path that has a value larger than a_0 is a direct successor of a_0; i.e. there is an entry that has the value *true* positioned above the diagonal and in the row that contains the head of the path.

To prove Warren's lemma one has to verify that Predicate 16.1 holds after completing the processing of $\langle\!\langle\ i, j \mid (i = a_0) \wedge (j < a_0) \mid m_{ij}\ \rangle\!\rangle$ for all paths. In the correctness argument, it is assumed that processing complies with the specified processing constraints. i.e. the entries that are required to be processed before a given entry as specified in terms of constraints, are indeed processed before the given entry.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Predicate 16.1 obviously holds for any given path of length 1 because if the length of the path is 1, the tail of the path is a direct successor of head of the path before any operations are performed on M by any algorithm that is designed to calculate the TC of M.

The proof of Warren's Lemma can be done by induction on the value of the head of the path; i.e. the value of a_0. The first step of this induction proof requires verification of the trivial case; i.e. the case when $a_0 = 1$. In this case $\langle\!\langle\, i, j \mid (i = a_0) \wedge (j < a_0) \mid m_{ij} \,\rangle\!\rangle$ is empty. The submatrix of entries that have to be processed before each of this empty submatrix is also empty.

Since the predicate trivially holds for paths of length 1, the proof that Predicate 16.1 holds for this trivial case, thus, requires that it has to be shown that Predicate 16.1 holds before any operations are performed on M for any path of length more than 1. i.e. for the general case where $a_0 = 1$ and $k > 1$. Lemma 16.3.2 shows this.

$$
\begin{array}{ll}
 & (A = \langle\!\langle\, i \mid i \in \mathbb{N}_{k+1} \mid a_i \,\rangle\!\rangle \text{ in } R) \wedge (a_0 = 1) \\
\Rightarrow & \quad \{\text{Replace } a_0 \text{ with 1. Definition of a path}\} \\
 & \{(1, a_1), (a_1, a_2), \ldots (a_{k-1}, a_k)\} \subseteq R \\
\Rightarrow & \quad \{\text{Initialisation of } M \text{ in Algorithm 16.4.1 (TileSkeleton)}\} \\
 & M[1, a_1] \wedge M[a_1, a_2] \wedge \ldots M[a_{k-1}, a_k] \\
\Rightarrow & \quad \{\text{A is acyclic, thus } \langle \forall t \mid \mid a_t > 1 \rangle \text{ in particular } a_1 > 1\} \\
 & M[1, a_1] \wedge (a_1 > 1) \\
\Rightarrow & \quad \{a_0 = 1. \text{ Replace 1 with } a_0\} \\
 & M[a_0, a_1] \wedge a_1 > a_0 \\
\Rightarrow & \quad \{t = 1 \text{ and } k > 1 \text{ ensures the truth of the following expression}\} \\
 & \langle\, \exists t \mid 0 < t < k \mid M[a_0, a_t] \wedge (a_t > a_0) \,\rangle \\
\Rightarrow & \quad \{\mathit{true} \text{ is the annihilator of } \vee\} \\
 & M[a_0, a_k] \vee \langle\, \exists t \mid 0 < t < k \mid M[a_0, a_t] \wedge a_t > a_0 \,\rangle
\end{array}
$$

Lemma 16.3.2: Trivial case of the induction proof of Lemma 16.3.3

16.3.3 Induction step for Warren's Lemma

The next step in the induction proof of Warren's Lemma (Lemma 16.3.3) is to show that Predicate 16.1 holds after processing $\langle\!\langle\, i, j \mid (i = a_0) \wedge (j < a_0) \mid m_{ij} \,\rangle\!\rangle$. The first expression in the proof shown in Lemma 16.3.3 is determined by assuming that a_0 is the head of A, an acyclic path in R with $a_0 = q$ and further assuming that Predicate 16.1 does not hold after processing $\langle\!\langle\, i, j \mid (i = a_0) \wedge (j < a_0) \mid m_{ij} \,\rangle\!\rangle$. The induction assumption is $a_0 < q$. Lemma 16.3.3 uses contradiction that this assumption does not hold as the arguments lead to the conclusion that it is false.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$\neg M[a_0, a_k] \wedge \langle \forall t \mid 0 < t < k \mid M[a_0, a_t] \Rightarrow a_t \leq a_0 \rangle$

$\Rightarrow$ { Pick an entry $M[a_0, a_r]$ so that $M[a_0, a_r]$ and a_r is maximised, i.e. $M[a_0, a_t] \Rightarrow a_t < a_r$ }

$\neg M[a_0, a_k] \wedge M[a_0, a_r] \wedge \langle \forall t \mid |M[a_0, a_t] \Rightarrow a_t < a_r \leq a_0 \rangle$

$\Rightarrow$ { A is acyclic, thus $a_r \neq a_0$ }

$\neg M[a_0, a_k] \wedge M[a_0, a_r] \wedge \langle \forall t \mid |M[a_0, a_t] \Rightarrow a_t < a_r < a_0 \rangle$

$\Rightarrow$ { It is given that processing of $\langle\!\langle\, i, j \mid i = a_0, j < a_0 \mid m_{ij} \,\rangle\!\rangle$ is completed. Since $a_r < a_0$, Lemma 16.2.4 specify that $\langle\!\langle\, i, j \mid i = a_r, j < a_r \mid m_{ij} \,\rangle\!\rangle \lhd \langle\!\langle\, i, j \mid i = a_0, j < a_0 \mid m_{ij} \,\rangle\!\rangle$. Furthermore, $a_r < a_0 = q$ implies that $a_r < q$, thus the induction assumption holds for the path $A' = \langle\!\langle\, i \mid r \leq i \leq k \mid a_i \,\rangle\!\rangle$; i.e. since the processing of $\langle\!\langle\, i, j \mid i = a_r, j < a_r \mid m_{ij} \,\rangle\!\rangle$ is done and $a_r < q$, Predicate 16.1 holds. }

$\neg M[a_0, a_k] \wedge M[a_0, a_r] \wedge \langle \forall t \mid |M[a_0, a_t] \Rightarrow a_t < a_r < a_0 \rangle \;\wedge$
$(\langle \exists s \mid r < s < k \mid M[a_r, a_s] \wedge a_s > a_r \rangle \vee M[a_r, a_k])$

$\Rightarrow$ { It is given that processing of $\langle\!\langle\, i, j \mid i = a_0, j < a_0 \mid m_{ij} \,\rangle\!\rangle$ is completed. Since $a_r < a_0$, it is known that $M[a_0, a_r]$ has been processed. Because $M[a_0, a_r]$ it follows that when $M[a_0, a_r]$ was processed, row a_r was added to row a_0. In particular $M[a_0, a_s] = M[a_0, a_s] + M[a_r, a_s]$ and $M[a_0, a_k] = M[a_0, a_k] + M[a_r, a_k]$ }

$\neg M[a_0, a_k] \wedge M[a_0, a_r] \wedge \langle \forall t \mid |M[a_0, a_t] \Rightarrow a_t < a_r < a_0 \rangle \;\wedge$
$(\langle \exists s \mid r < s < k \mid M[a_0, a_s] = M[a_r, a_s] \wedge a_s > a_r \rangle \vee M[a_0, a_k] = M[a_r, a_k])$

$\Rightarrow$ { The conclusion that $M[a_0, a_k]$ contradicts the fact that $\neg M[a_0, a_k]$. Therefore $\neg M[a_r, a_k]$ }

$\neg M[a_0, a_k] \wedge M[a_0, a_r] \wedge \langle \forall t \mid |M[a_0, a_t] \Rightarrow a_t < a_r < a_0 \rangle \;\wedge$
$\langle \exists s \mid r < s < k \mid M[a_0, a_s] = M[a_r, a_s] \wedge a_s > a_r \rangle$

$\equiv$ { Because $M[a_0, a_t] \Rightarrow a_t < a_r$ and also $M[a_0, a_s]$ it follows that $a_s < a_r$ which contradicts the fact that $a_s > a_r$ }

false

Lemma 16.3.3: The assumption that Predicate 16.1 is untrue leads to a contradiction

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Through the application of induction it has thus been shown that for every acyclic path $A = \langle\!\langle\ i \mid i \in \mathbb{N}_{k+1} \mid a_i\ \rangle\!\rangle$ in R with the head of the path denoted by a_0, it follows that Predicate 16.1 holds after Algorithm 16.4.1 (TileSkeleton) has processed the entries of the sub-matrix $\langle\!\langle\ i, j \mid (i = a_0) \wedge (j < a_0) \mid m_{ij}\ \rangle\!\rangle$.

Warren's lemma assumes compliance with the primary processing order constraint discussed in Section 16.2. It also assumes Lemma 16.2.4. As Lemma 16.2.4 is a consequence of the secondary processing order constraint, it is clear that Lemma 16.3.3 relies only on the primary and secondary processing order constraints discussed in Section 16.2.

16.3.4 Agrawal's lemma

Agrawal's lemma purports that when the row containing the head of a path is fully processed while adhering to the processing order constraints, it is guaranteed that the tail of the path is a direct successor of the head of the path. Warren's lemma (Lemma 16.3.3) identifies possible unprocessed entries in the same row of the head of a given path that — when processed — will lead to the desired condition, namely that the tail of the path is a direct successor of the head of the path. Agrrawal's lemma (Lemma 16.3.4) guarantees that after processing all these unprocessed entries on this row, this required state is reached.

To prove the lemma we start by taking any acyclic path $A = \langle\!\langle\ i \mid i \in \mathbb{N}_{k+1} \mid a_i\ \rangle\!\rangle$ in R. Denote the head of A by a_0. We then pick m, $0 < m < n$, such that $a_m \in A \wedge \langle\ \forall i \mid i \in \mathbb{N}_{k+1} \mid a_i < a_m\ \rangle$. In other words, we pick a_m to be the largest element of A. If this largest element is the tail of the path the desired state is already reached before the algorithm is executed, therefore we assume that a_m is not the tail of the path. We further assume that $\langle\!\langle\ i, j \mid i = a_0 \mid m_{ij}\ \rangle\!\rangle$ is processed, i.e. all the entries in the row containing the head of the path A in $M(R)$ is processed.

In the argument shown in Lemma 16.3.4, the fist expression holds as it is the above mentioned chosen case after applying Warren's lemma to A. In the rest of the argument, Warren's lemma is applied repeatedly. In every step it is argued that the required state is reached as part of the expression that is derived. If it has not been reached, the remainder of the expression has to hold. Continuing in this fashion it is shown that the required state is inevitably reached as the relation is finite.

Lemma 16.3.4 thus states that for every acyclic path $A = \langle\!\langle\ i \mid i \in \mathbb{N}_{k+1} \mid a_i\ \rangle\!\rangle$ in R, it follows that while executing Algorithm 16.4.1 (TileSkeleton), $M[a_0, a_k]$ holds after processing the entries of $\langle\!\langle\ i, j \mid (i = a_0) \mid m_{ij}\ \rangle\!\rangle$ where a_0 is the head of the path.

Lemma 16.3.4 assumes compliance with the primary processing order constraint along with Lemma 16.2.4 and Lemma 16.3.3. As both Lemma 16.2.4 and Lemma 16.3.3 do not assume anything other than the primary and secondary processing order constraints, it is clear that Lemma 16.3.4 also relies only on these two constraints discussed in Section 16.2.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$M[a_0, a_k] \vee \langle \exists t_0 \mid 0 < t_0 < k \mid M[a_0, a_{t_0}] \wedge a_{t_0} > a_0 \rangle$

$\Rightarrow$ { If $M[a_0, a_k]$, we are done. The case if $\neg M[a_0, a_k]$, is taken further. Also, since m was chosen such that $\langle \forall i \mid i \in \mathbb{N}_{k+1} \mid a_i < a_m \rangle$, it holds that $a_m > a_{t_0}$ }

$\langle \exists t_0 \mid 0 < t_0 < k \mid M[a_0, a_{t_0}] \wedge a_m > a_{t_0} > a_0 \rangle$

$\Rightarrow$ { The conditions for the application of Warren's lemma to the sub-path $\langle\!\langle\, i \mid t_0 \leq i \leq k \mid a_i \,\rangle\!\rangle$ holds because Lemma 16.2.4 with respect to $M[a_0, a_{t_0}]$ asserts that $\langle\!\langle\, i, j \mid i < a_{t_0} \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd M[a_0, a_{t_0}]$ }

$\langle \exists t_0 \mid 0 < t_0 < k \mid M[a_0, a_{t_0}] \wedge a_m > a_{t_0} > a_0 \rangle$
$\wedge\ (M[a_{t_0}, a_k] \vee \langle \exists t_1 \mid 0 < t_1 < k \mid M[a_{t_0}, a_{t_1}] \wedge a_m > a_{t_0} > a_{t_1} > a_0 \rangle)$

$\Rightarrow$ { When processing $M[a_0, a_{t_0}]$, the value $M[a_0, a_k] \vee M[a_{t_0}, a_k]$ is assigned to $M[a_0, a_k]$. If $M[a_{t_0}, a_k]$, then $M[a_0, a_k]$ and we are done. The case if $\neg M[a_{t_0}, a_k]$ is taken further. }

$\langle \exists t_0 \mid 0 < t_0 < k \mid M[a_0, a_{t_0}] \wedge a_m > a_{t_0} > a_0 \rangle$
$\wedge\ \langle \exists t_1 \mid 0 < t_1 < k \mid M[a_{t_0}, a_{t_1}] \wedge a_m > a_{t_0} > a_{t_1} > a_0 \rangle$

$\Rightarrow$ { $\wedge$ is associative }

$\langle \exists t_0, t_1 \mid 0 < t_0, t_1 < k \mid M[a_0, a_{t_0}] \wedge M[a_{t_0}, a_{t_1}] \wedge a_m > a_{t_0} > a_{t_1} > a_0 \rangle$

$\Rightarrow$ { $0 < t_0, t_1 < k$ may be omitted as it is obvious that only entries in A are chosen }

$\langle \exists t_0, t_1 \mid\mid M[a_0, a_{t_0}] \wedge M[a_{t_0}, a_{t_1}] \wedge a_m > a_{t_0} > a_{t_1} > a_0 \rangle$

$\Rightarrow$ { Since A is finite, Warren's lemma can be applied a finite number (say s) times. If a q can be identified for which $M[a_{t_q}, a_k]$, the processing of $M[a_0, a_{t_q}]$ will cause $M[a_0, a_k]$ to become *true* ($M[a_0, a_k] \vee M[a_{t_q}, a_k]$) and we are done. Otherwise after s applications of the Lemma we will have the following }

$\langle \exists t_0, t_1, \ldots t_s \mid\mid M[a_0, a_{t_0}] \wedge M[a_{t_0}, a_{t_1}] \wedge \cdots \wedge M[a_{t_{s-1}}, a_{t_s}]$
$\wedge\ a_m > a_{t_0} > a_{t_1} > \cdots > a_{t_s} > a_0 \wedge \langle i \mid a_0 < a_i < a_{t_s} \mid a_i \rangle = \varnothing \rangle$

$\Rightarrow$ { While $\langle i \mid a_0 < a_i < a_{t_s} \mid a_i \rangle = \varnothing$, apply Warren's lemma to the sub-path $\langle\!\langle\, i \mid t_{t_s} \leq i \leq k \mid a_i \,\rangle\!\rangle$ }

$\langle \exists t_0, t_1, \ldots t_s \mid\mid M[a_0, a_{t_0}] \wedge M[a_{t_0}, a_{t_1}] \wedge \cdots \wedge M[a_{t_{s-1}}, a_{t_s}]$
$\wedge\ a_m > a_{t_0} > a_{t_1} > \cdots > a_{t_s} > a_0 \wedge \langle i \mid a_0 < a_i < a_{t_s} \mid a_i \rangle = \varnothing \rangle$
$\wedge\ \ M[a_{t_s}, a_k]$

$\Rightarrow$ { When processing $\langle\!\langle\, i, j \mid i = a_0 \mid m_{ij} \,\rangle\!\rangle$, compliance with the primary constraint requires that $M[a_0, a_{t_s}] \lhd M[a_0, a_{t_{s-1}}] \lhd \cdots \lhd M[a_0, a_{t_0}]$ The processing of $M[a_0, a_{t_s}]$ will cause $M[a_0, a_k]$ to become $M[a_0, a_k] \vee M[a_{t_s}, a_k]$. }

$M[a_0, a_k]$

Lemma 16.3.4: Processing $\langle\!\langle\, i, j \mid i = a_0 \mid m_{ij} \,\rangle\!\rangle \Rightarrow M[a_0, a_k]$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

16.4 Tiling algorithm

Here an abstract skeleton algorithm is presented for calculating the transitive closure of a relation. This skeleton algorithm is an abstract algorithm from which a number of concrete implementations of the algorithm are derived in Chapter 17. Distinct concrete implementations of this algorithm differ from one another only in terms of the application of contrasting tiling strategies. When describing a concrete implementation of the algorithm, it is sufficient to describe the tiling strategy of the algorithm in terms of rules to construct a partition of M. Since it is shown in Section 16.3.1 that the processing of elements in $M_{\mathbb{I}}$ are superfluous, it is acceptable to describe the algorithm in terms of rules that construct a partition of $M - M_{\mathbb{I}}$.

Per definition a partition is a sequence, therefore the rules for constructing the specified partition should include the specification of an ordering of the submatrices comprising the partition. When executing the algorithm, processing will iterate over the submatrices in the order they are listed in the partition. Every submatrix in the partition is also a sequence. Thus an ordering for the entries in each submatrix is specified. The entries in a submatrix are processed by the algorithm in the order they are listed in the submatrix.

The specification of this algorithm is given in Algorithm 16.4.1 (TileSkeleton). It requires a function called **tile** that creates a partition of M or $M - M_{\mathbb{I}}$. To specify an algorithm derived from this algorithm one only has to define an implementation of this function.

Algorithm 16.4.1 (TileSkeleton) implements Algorithm 12.5.1 (MatrixGrow). It systematically calculates the value of each entry in M using a formula that is the equivalent of Equation 12.11. When compared to Algorithm 12.6.1 (Warshall), it uses an alternative deterministic order to traverse the elements in M while maintaining the correctness of the algorithm. Where Warshall's algorithm sequentially iterates through the entries of M in a natural order, this algorithm follows an order that is specified by the **t**ile function.

Figure 16.4.1 shows the topics and relations in the TM of TCA that are related to Algorithm 16.4.1 (TileSkeleton).

The TM specification of this algorithm in the TM of TCA is given in Listing C23. The TM specification of the algorithmic technique applied by this algorithm, called *loop tiling*, is included in Listing C8.

16.4.1 Verification

The operations performed by this algorithm are the same as the operations performed by all algorithms derived from Algorithm 12.5.1 (MatrixGrow). In Section 12.5.2 it was shown that the value of M converges to M^+ when these operations are performed on M and that $M = M^+$ after performing these operations a finite number of times on each of the entries in M. For a concrete implementation of this algorithm to be valid, it has to be shown that M^+ is reached after processing each of the entries in $M - M_{\mathbb{I}}$ at most once, provided that the entries are processed in the specified order.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

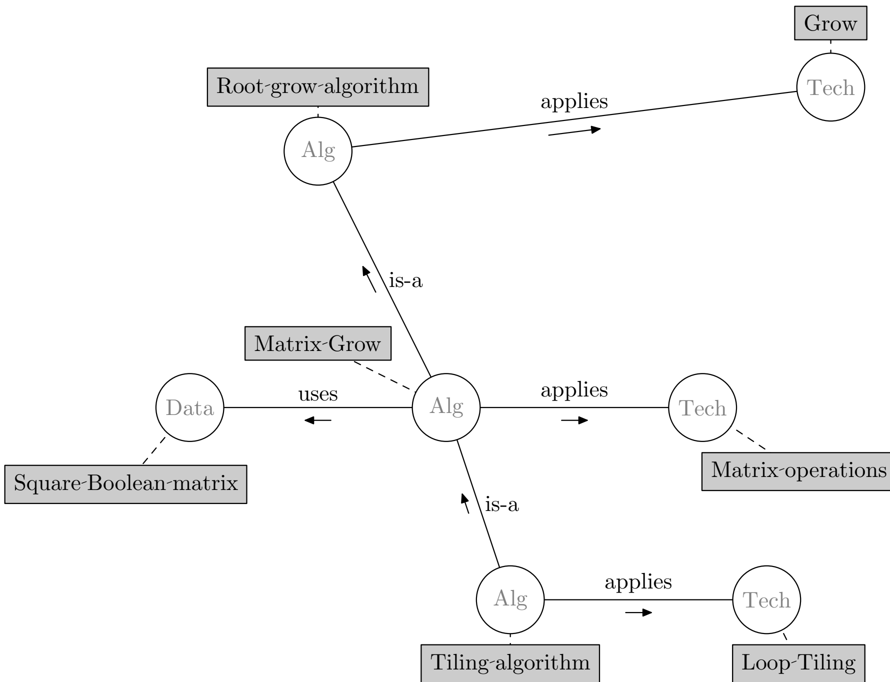


Figure 16.4.1: Topics related to the tiling algorithm

Algorithm 16.4.1 (TileSkeleton) reaches M^+ after processing all the submatrices of any valid partition of M because any valid tiling strategy creates a partition that includes all the rows of M. In particular, for each path contained in the relation of which the TC is to be calculated, the partition contains a set of submatrices that intersects with the row containing the head of the path. Furthermore, the partition has to comply with the primary order constraint. This guarantees that this set of submatrices that intersects with the row containing the head of the path is ordered in a way that the entries on this row are processed as assumed in the proof of Lemma 16.3.4. Application of Lemma 16.3.4 renders the tail of the path to be a direct successor of the head of the path after processing this set of submatrices.

The correctness of Algorithm 16.4.1 (TileSkeleton) follows from the application of Agrawal's lemma. Lemma 16.3.4 holds for every path in the relation of which the TC is to be calculated. This means that for each indirect connection between two points in the original relation, the resulting relation will have a direct connection between these points. It can thus be concluded that M^+ is reached after processing the submatrices of any concrete implementation of Algorithm 16.4.1 (TileSkeleton).

Warren's lemma assumes compliance with the primary processing order constraint discussed in Section 16.2. It also assumes Lemma 16.2.4. As Lemma 16.2.4 is a consequence of the secondary processing order constraint, it is clear that

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 16.4.1: Tiling Algorithm

const $R \subseteq U \times U$
$n = |U| \in \mathbb{N}^+$
var $M \in \mathbb{B}[n, n]$
$\mathcal{M} \subseteq \langle\!\langle\ t \mid t \in \mathbb{N}, P_t \subset M \mid P_t\ \rangle\!\rangle$

$M = \Phi(R)$
$\mathcal{M}$:= **tile**(M);
$\{(\mathcal{M} \vdash M) \vee (\mathcal{M} \vdash (M - M_{\mathbb{I}}))\}$
for $P_t \in \mathcal{M} \quad \rightarrow$
  for $M[i, j] \in P_t$
    if $\neg M[i, j] \rightarrow$ **skip**
    $[\!]\ M[i, j] \quad \rightarrow$
      for $k \in \mathbb{N}_n \quad \rightarrow$
        $M[i, k] := M[i, k] \vee M[j, k]$
      rof
    fi
  rof
rof
$\{$**Post:** $\Psi(M) = R^+\}$

Lemma 16.3.3 relies only on the primary and secondary processing order constraints discussed in Section 16.2.

Lemma 16.3.4 assumes compliance with the primary processing order constraint along with Lemma 16.2.4 and Lemma 16.3.3. As both Lemma 16.2.4 and Lemma 16.3.3 do not assume anything other than the primary and secondary processing order constraints, it is clear that Lemma 16.3.4 also relies only on these two constraints discussed in Section 16.2. It can therefore be concluded that compliance with only the primary and secondary order constraints are sufficient for a tiling strategy to guarantee correctness of a concrete implementation of Algorithm 16.4.1 (TileSkeleton).

To verify the correctness of a concrete implementation of this algorithm it is necessary to show that the tiling strategy of the implementation constructs a partition of M or $M - M_{\mathbb{I}}$. It is also necessary to show that the ordering implied by the constructed partition complies with the processing order constraints that were assumed when the correctness of Algorithm 16.4.1 (TileSkeleton) was explored here, namely the primary constraint specified in Section 16.2.1 and the secondary constraint specified in Section 16.2.2.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

16.5 Position in a derivation tree

There are two ways to position the abstract tiling algorithm in the derivation tree of TC algorithms depending on how one prefers to view Warshall's algorithm in relation to the abstract tiling algorithm.

16.5.1 Warshall's algorithm is a tiling algorithm

It can be argued that Algorithm 12.6.1 (Warshall) can be interpreted to be a degenerate tiling algorithm that uses the trivial tiling strategy of defining a partition that consists of one submatrix namely the entire M. This view requires that the derivation hierarchy that is developed so far, be changed by adding Algorithm 16.4.1 (TileSkeleton) as an abstraction of Algorithm 12.6.1 (Warshall) as illustrated in Figure 16.5.1.

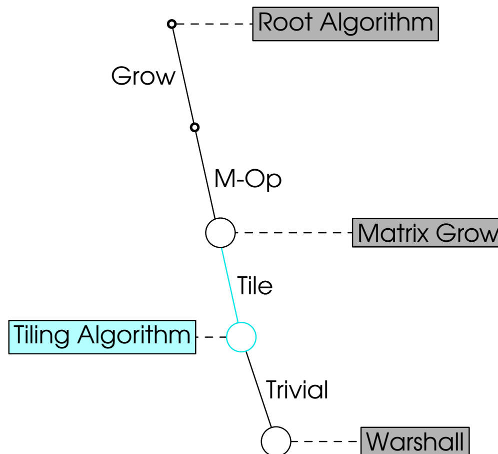


Figure 16.5.1: Tiling algorithm is an abstraction of Warshall's algorithm

The TM specification of this tiling strategy called *Trivial* as an algorithmic technique in the TM of TCA is included in Listing C10.

16.5.2 Derived from the abstraction of Warshall's algorithm

An alternative argument is that Algorithm 16.4.1 (TileSkeleton) is an alternative implementation of Algorithm 12.5.1 (MatrixGrow) similar to how Algorithm 12.6.1 (Warshall) is an implementation of Algorithm 12.5.1 (MatrixGrow) where these two algorithms use different process orderings. To accommodate this view both Algorithm 16.4.1 (TileSkeleton) and Algorithm 12.6.1 (Warshall) are derived from Algorithm 12.5.1 (MatrixGrow) as illustrated in Figure 16.5.2.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

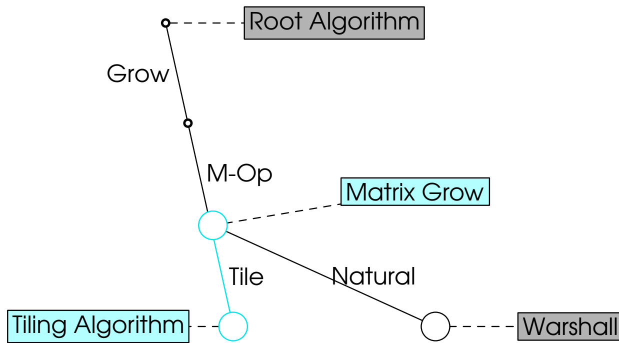


Figure 16.5.2: Tiling algorithm is an alternative to Warshall's algorithm

16.5.3 Accepting both views

Both these views are valid and in both cases the tiling algorithm is derived from Algorithm 12.5.1 (MatrixGrow). The different viewpoints, however, specify different derivations of Warshall's algorithm. Figure 16.5.3 combines both these views showing both derivation paths for Algorithm 12.6.1 (Warshall)

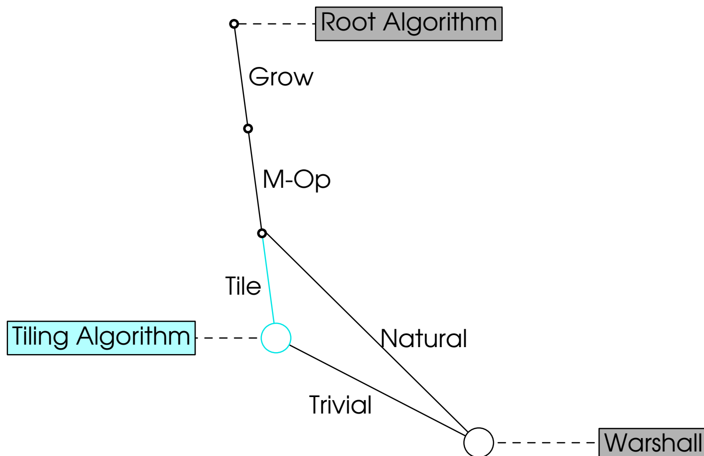


Figure 16.5.3: Alternative derivation paths of Warshall's algorithm

16.6 Summary

Although loop tiling is an optimisation strategy that is commonly applied automatically by optimising compilers, this chapter argues that manual specification of such strategies may be beneficial. It has been observed that implementations

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

of TC algorithms often have poor locality of reference [192]. This poor locality of reference complicates the automatic application of loop tiling.

The discussions in this chapter rely on the concept of a partition of a matrix and on the notation to specify such a partition. These were specifically developed in Section 4.5 for their use here to specify the concept of a tiling strategy.

Section 16.2 discusses the requirements regarding the order in which the operations in grow algorithms should be performed. Agrawal and Jagadish [2] claim that compliance with the processing order constraints they specify are sufficient to ensure that a grow algorithm reaches the TC of a relation in one pass over the iteration space of the algorithm. In Section 16.2 of this chapter the constraints presented by Agrawal and Jagadish [2] are reformulated to remove ambiguities in their specification.

The proof of Agrawal and Jagadish's [2] claim could not be retrieved. Instead, a novel and salient proof of a similar claim is given in this chapter. The proof assumes two lemmas which I call Warren's lemma and Agrawal's lemma. Warren's lemma is a lemma I have formulated and proved. It is a generalisation of a lemma that Warren [251] proposed and proved in connection with the correctness of his algorithm to calculate the TC of a relation. Where Warren's proof of his lemma relies on qualities imposed by the specification of his algorithm, I rely only on compliance with the generic constraints I have formulated in Section 16.2 to prove my version of the Lemma.

Agrawal's lemma is a lemma I formulated in order to prove Agrawal and Jagadish's [2] claim. I succeeded in proving a stronger claim namely that compliance with only the primary and secondary processing order constraints I have specified is sufficient to ensure that a grow algorithm reaches the TC of a relation in one pass over the iteration space of the algorithm. This breakthrough opens possibilities to simplify the proofs of parallelised grow algorithms which calculate the TC of a relation.

The positioning of this algorithm discussed in Section 16.5 posed a challenge. It was at first not obvious how this abstract tiling algorithm relates to the other grow algorithms that have been discussed so far. Eventually I realised that it neatly fits as a derivation of Algorithm 12.5.1 (MatrixGrow) and that different views on the nature of Warshall's algorithm allows for the specification of alternative derivation parts of Warshall's algorithm.

The next chapter discusses algorithms derived from the algorithm introduced in this chapter. These algorithms perform the operations specified by this algorithm but apply opposing tiling strategies to govern the order in which the operations of the algorithm are performed.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 17

Concrete Tiling Algorithms

In Chapter 16 an abstract skeleton algorithm is presented for calculating the transitive closure of a relation. It is assumed that the relation is represented using a Boolean matrix M. This skeleton algorithm serves as a template for concrete implementations of the algorithm. Distinct implementations of this algorithm differ from one another only in terms of the application of contrasting tiling strategies. When describing a concrete implementation of the algorithm, it is sufficient to describe the tiling strategy of the algorithm. The tiling strategy should construct a partition of M. As defined in Section 4.5 a partition of a matrix M is a sequence of submatrices of M that are both collectively exhaustive and mutually exclusive with respect to the matrix being partitioned.

The execution of a tiling algorithm iterates over the submatrices in the partition that was constructed using its tiling strategy. Each of the submatrices in the partition is processed in the order in which it appears in the partition. The order of the elements in each submatrix determines the order in which the elements are processed.

To show the correctness of a tiling algorithm, it has to be verified that its tiling strategy indeed constructs a partition on M and that the processing of the elements as specified by this partition, complies with the processing order constraints that were established in Section 16.2.

This chapter describes concrete implementations of the abstract tiling algorithm discussed in Chapter 16. The tiling strategy of each algorithm is defined and verified.

17.1 Warren's algorithm

Besides the trivial tiling strategy applied by Algorithm 12.6.1 (Warshall), the simplest tiling strategy for the calculation of TC is the one proposed by Warren [251].

Warren [251] noticed that M reaches the value of M^+ after only two iterations of Algorithm 13.1.3 (GrowRow) when limiting the first iteration to the entries of

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

the adjacency matrix below its main diagonal and then processing the remaining elements during the second iteration.

The tiling strategy of this algorithm specifies a partition consisting of two submatrices. The first submatrix comprises of the elements below the main diagonal, while the second submatrix comprises those above. Both submatrices are processed in row order. The entries on $M_{\mathbb{I}}$ are excluded. This exclusion is allowed because it was shown in Section 16.3.1 that the processing of the elements on $M_{\mathbb{I}}$ has no effect. Here Warren's tiling strategy is called *diagonal*.

The TM specification of this tiling strategy as an algorithmic technique in the TM of TCA is included in Listing C10 while the TM specification of Warren's algorithm in this TM is given in Listing C28. The topics and relations that are specified in these listings are shown in Figure 17.1.1. The TM specification pointing to implementations of Warren's algorithm can be found in Listing C56. For efficiency, these implementations fuse the specification of the submatrices with their execution.

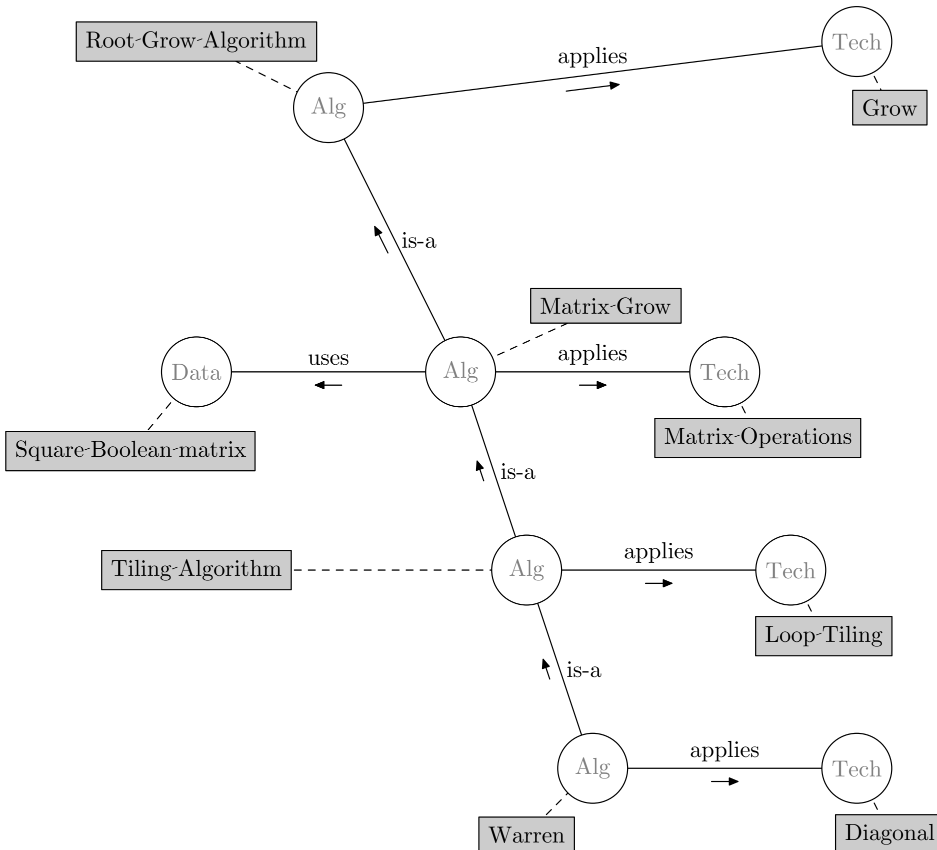


Figure 17.1.1: Topics related to Warren's algorithm

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

17.1.1 Specification

Warren's algorithm is a concrete implementation of the tiling algorithm specified in Algorithm 16.4.1 (TileSkeleton). Algorithm 17.1.1 (Warren) is the implementation of the **tile** function of Algorithm 16.4.1 (TileSkeleton) to constitute Warren's algorithm.

Algorithm 17.1.1: Warren's tile function

---

func tile$(M \in \mathbb{B}[n,n])$: $\langle\!\langle\ t \mid t \in \mathbb{N}, P_t \subseteq M \mid P_t\ \rangle\!\rangle$
  $P_0 := \langle\!\langle\ i,j \mid i > j \mid m_{ij}\ \rangle\!\rangle;$
  $P_1 := \langle\!\langle\ i,j \mid i < j \mid m_{ij}\ \rangle\!\rangle;$
  $\{(P_0, P_1) \vdash M - M_{\mathbb{I}}\}$
  return (P_0, P_1)
cnuf

17.1.2 Complexity

The algorithm starts with a function call. The complexity of this function is constant as it is not dependent on the size of the matrix, i.e. it is $\Theta(1)$. The rest of the body of the algorithm consists of three nested loops called the inner, the middle, and the outer loop.

The complexity of the outer loop is the number of submatrices in the partition i.e. the constant value of 2. Thus the complexity of the outer loop is $\Theta(1)$.

The number of iterations of the middle loop is dependent on the number of elements in the submatrix. Each submatrix has exactly half of the elements of the $n \times n$ matrix after the n elements on $M_{\mathbb{I}}$ are omitted, i.e. $(n \times n - n) \div 2$ elements. Thus the complexity of the middle loop is $\Theta(\frac{n^2-n}{2}) = \Theta(\frac{1}{2}n^2 - \frac{1}{2}n) = \Theta(n^2)$.

The inner loop, if performed, requires n operations to add an entire row in the matrix to another row in the matrix. The complexity of this operation is $\Theta(n)$. Thus the complexity of executing the algorithm is $\Theta(1) + (\Theta(1) \times \Theta(n^2) \times \Theta(n)) = \Theta(1 + n^3) = \Theta(n^3)$.

Listing C38 shows the TM specification of the complexity of Warrens's algorithm. It is the same complexity as Warshall's algorithm. Warren's algorithm may be preferred above Warshall's algorithm because its operations are in row major order, as opposed to the column major order dictated in Warshall's algorithm. Usually matrices are stored in row major order which may cause Warren's algorithm to outperform Warshall's algorithm.

Warren's algorithm saves a linear amount of processing time by omitting the redundant operations of processing the elements on $M_{\mathbb{I}}$. Apart from adding the overhead needed to limit the processing to the appropriate submatrix, this algorithm calculates the transitive closure of R in a single pass, which can be argued to be performed as two halve passes, of Algorithm 13.1.3 (GrowRow).

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

17.1.3 Position in a derivation tree

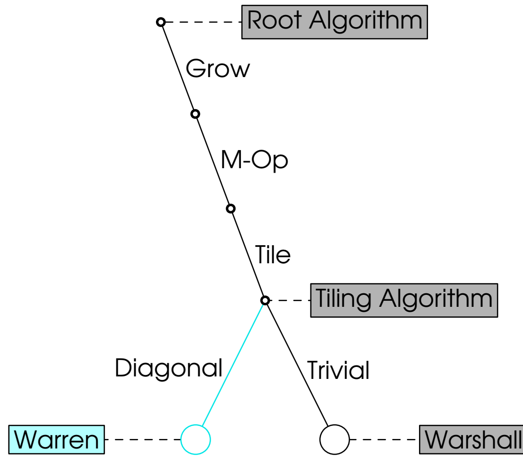


Figure 17.1.3: Derivation tree of Warren's algorithm

The derivation steps of Warren's algorithm is shown in Figure 17.1.3. The first three steps to derive this algorithm from the root TC algorithm are the same as that of Warshall's algorithm if Warshall's algorithm is considered to be a tiling algorithm as shown in Figure 16.5.1. These three steps are the application of the fundamental *grow* technique, the transformation of the implementation to use square Boolean matrices to be able to apply *matrix operations*, and finally to apply *loop tiling*. It differs from Warshall's algorithm by applying an alternative tiling strategy. When compared with Warshall's algorithm, the derivation of Warren's algorithm is achieved by applying the *Diagonal* tiling technique instead of the *Trivial* tiling technique.

17.2 Verification of Warren's algorithm

17.2.1 The tile function returns a valid partition

When Algorithm 17.1.1 (Warren) constructs the sequence (P_0, P_1), each entry of M, with the exclusion of entries on the main diagonal of M, is uniquely assigned to either P_0 or P_1. Thus P_0 and P_1 are mutually exclusive as well as exhaustive with respect to $M - M_{\mathbb{I}}$; i.e. $(P_0, P_1) \vdash M - M_{\mathbb{I}}$.

As explained in Section 16.3.1, the processing of the entries on the main diagonal of the matrix in grow algorithms is redundant and may thus be omitted. The processing of (P_0, P_1) as specified in Algorithm 16.4.1 (TileSkeleton) will thus yield the transitive closure if it can be verified that the processing of the entries in this specified order complies with the required processing order constraints.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

17.2.2 Primary order constraint

Lemma 17.2.2 shows that Algorithm 17.1.1 (Warren) complies with the primary order constraint.

$$
\begin{array}{ll}
 & y_0 < y_1 \\
\Rightarrow & \left\{ \begin{array}{l} \text{All possible positions of } x \text{ in relation to } y_0 \text{ and } y_1 \\ \text{Equality is not included as the diagonal itself} \\ \text{is not processed} \end{array} \right\} \\
 & (x < y_0 < y_1) \vee (y_0 < x < y_1) \vee (y_0 < y_1 < x) \\
\Rightarrow & \{ \text{Definitions of } P_0 \text{ and } P_1 \} \\
 & ((y_0 < y_1) \wedge (M[x, y_0] \in P_1 \wedge M[x, y_1] \in P_1)) \\
 & \vee\ ((y_0 < y_1) \wedge (M[x, y_0] \in P_0 \wedge M[x, y_1] \in P_1)) \\
 & \vee\ ((y_0 < y_1) \wedge (M[x, y_0] \in P_0 \wedge M[x, y_1] \in P_0)) \\
\Rightarrow & \left\{ \begin{array}{l} P_1 \text{ is processed in row order}, \quad P_0 \lhd P_1, \\ P_0 \text{ is processed in row order} \end{array} \right\} \\
 & (M[x, y_0] \lhd M[x, y_1]) \vee (M[x, y_0] \lhd M[x, y_1]) \vee (M[x, y_0] \lhd M[x, y_1]) \\
\Rightarrow & \{ \vee \text{ is idempotent} \} \\
 & M[x, y_0] \lhd M[x, y_1]
\end{array}
$$

Lemma 17.2.2: $(y_0 < y_1) \Rightarrow M[x, y_0] \lhd M[x, y_1]$

17.2.3 Secondary order constraint in the lower triangle

Lemma 17.2.3 shows that Algorithm 17.1.1 (Warren) complies with the secondary order constraint in the lower triangle.

$$
\begin{array}{ll}
 & y < x \\
\Rightarrow & \{ \text{Let } M[x_0, y_0] \in \langle\!\langle\, i, j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \} \\
 & y < x \wedge x_0 = y \wedge y_0 < x_0 \\
\Rightarrow & \{ \wedge \text{ is commutative, associative and idempotent} \} \\
 & (x_0 = y \wedge y < x) \wedge y < x \wedge y_0 < x_0 \\
\Rightarrow & \{ \text{Simplify, definition of } P_0, \text{ definition of } P_0 \} \\
 & x_0 < x, M[x, y] \in P_0, M[x_0, y_0] \in P_0 \\
\Rightarrow & \{ P_0 \text{ is processed in row major order} \} \\
 & M[x_0, y_0] \lhd M[x, y]
\end{array}
$$

Lemma 17.2.3: $y < x \Rightarrow \langle\!\langle\, i, j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd M[x, y]$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

17.2.4 Secondary order constraint in the upper triangle

Lemma 17.2.4 shows that Algorithm 17.1.1 (Warren) complies with the secondary order constraint in the upper triangle.

$$
\begin{array}{ll}
 & y > x \\
\Rightarrow & \{\text{Let } M[x_0, y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \} \\
 & y > x \wedge x_0 = y \wedge y_0 < x_0 \\
\Rightarrow & \{\text{Weaken condition}\} \\
 & y > x \wedge y_0 < x_0 \\
\Rightarrow & \{\text{Definition of } P_1\text{, definition of } P_0\} \\
 & M[x,y] \in P_1, M[x_0,y_0] \in P_0 \\
\Rightarrow & \{P_0 \lhd P_1\} \\
 & M[x_0,y_0] \lhd M[x,y]
\end{array}
$$

Lemma 17.2.4: $x < y \Rightarrow \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd M[x,y]$

17.2.5 Conclusion

Lemma 17.2.2 confirms that the processing of $M[a,b]$ in this algorithm complies with the primary constraint while Lemmata 17.2.3 and 17.2.4 show that the processing order of $M[a,b]$ in this algorithm complies with the secondary constraint. Lemma 17.2.3 shows that it complies in the lower triangle while Lemma 17.2.4 shows that it complies in the upper triangle. As all the requirements are met, it can be concluded that Warren's algorithm is correct.

17.3 Blocked row algorithm

The blocked row algorithm is discussed in publications by Agrawal and various colleagues [2, 3]. A list of related publications can be found on Agrawal's personal home page[1]. These publications describe and evaluate the blocked row algorithm described in this section as well as the blocked column algorithm described in Section 17.5.

Both these algorithms were originally described in terms of successor lists as explained in Section 3.5.2. In this thesis these algorithms are described assuming that the relations are represented using their adjacency matrices and applying the appropriate matrix operations to achieve the same results.

As a consequence of this translation the algorithms described here are simplified versions of the original algorithms proposed by Agrawal and his colleagues.

[1] `http://rakesh.agrawal-family.com/pubs.html`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

In the original description it was acknowledged that the successor lists that are maintained in memory may grow to such an extent that the available memory for the current operation may fill up during operation, requiring dynamic repartitioning to overcome the problem. If the relations are represented in terms of adjacency matrices, the number of bits to represent the successor list of a node is a fixed value, namely the number of nodes in the relation. Thus repartitioning need not be considered.

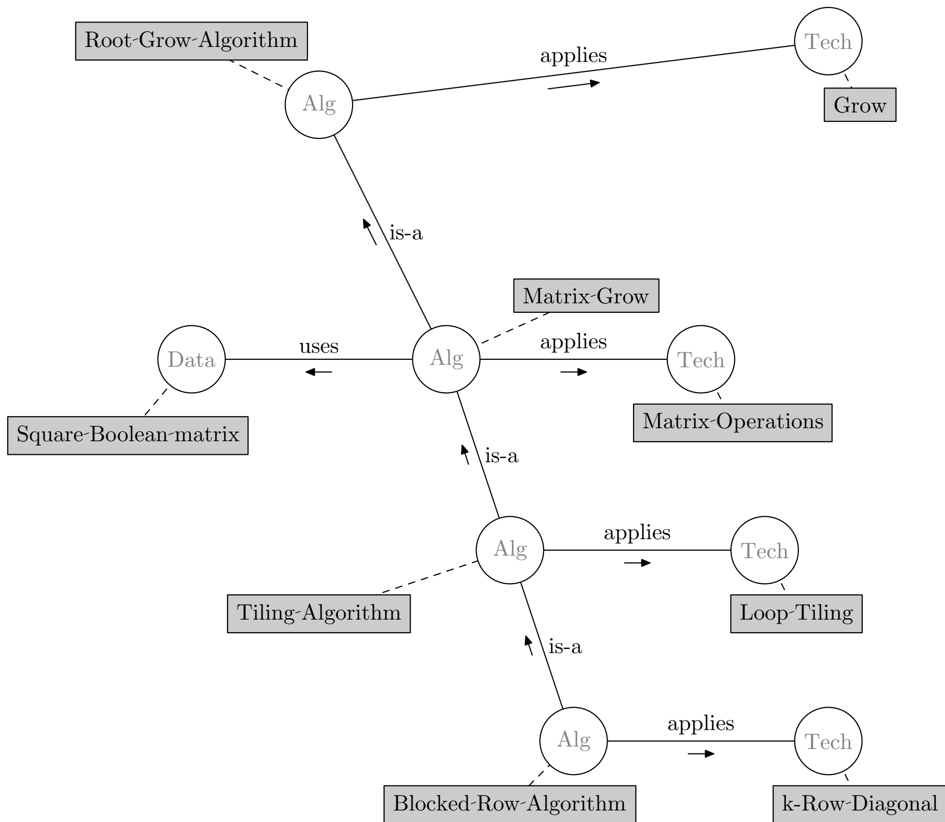


Figure 17.3.1: Topics related to Agrawal's blocked row algorithm

The TM specification of this algorithm in the TM of TCA is given in Listing C29. The topics and relations that are defined in this specification are shown in Figure 17.3.1. The TM specification pointing to implementations of Agrawal's blocked row algorithm can be found in Listing C58. In these implementations a constant is declared that should be used to set the available memory size determined by the hardware configuration. The TM specification of the tiling strategy used by this algorithm in the TM of TCA as an algorithmic technique is included in Listing C10.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 17.3.1: Tile function of the blocked row algorithm

func tile($M \in \mathbb{B}[n,n], m \in \mathbb{N}$) : $\langle\!\langle\, t \mid t \in \mathbb{N}, P_t \subseteq M \mid P_t \,\rangle\!\rangle$
    var $s \in \mathbb{N}$

    $s, t := \lceil n/(m-1) \rceil, 0;$
    do $(t < s)$
        $P_t := \langle\!\langle\, j, i \mid (t \times s) \leq i < ((t+1) \times s), i > j \mid m_{ij} \,\rangle\!\rangle;$
        $P_{t+s} := \langle\!\langle\, j, i \mid (t \times s) \leq i < ((t+1) \times s), i < j \mid m_{ij} \,\rangle\!\rangle;$
        $t := t + 1;$
    od
    $\{ \langle\!\langle\, t \mid t \in \mathbb{N}_{2\times s} \mid P_t \,\rangle\!\rangle \vdash M - M_{\mathbb{I}} \}$
    return $\langle\!\langle\, t \mid t \in \mathbb{N}_{2\times s} \mid P_t \,\rangle\!\rangle$
cnuf

Here m is the number of rows of the given matrix that can fit into memory at one time. This value is determined by the physical requirements for optimal performance which in turn is governed by the algorithms and metrics applied by the operating system to manage allocation of memory blocks to processes. The number of rows that should go into a submatrix is $m - 1$ to enable loading one additional row to access to parts of the relation that are outside the current submatrix. The calculated value $s = n/(m-1)$ is the number of sections formed.

A partition comprising of a total of $2 \times s$ submatrices is formed. For each of the s horizontal sections comprising of at most $m - 1$ rows, two submatrices are formed. One submatrix consists of the entries in that section that are below the main diagonal of the matrix and the other submatrix consists of the entries in the section that are above the main diagonal of the matrix.

The submatrices are numbered from 0 to $2 \times s - 1$ indicating their order in the partition. The submatrices with entries that are below the main diagonal of the matrix are the first s entries in the partition. The submatrices comprising entries above the main diagonal of the matrix are in the second half of the partition. The submatrices within each half of the partition are ordered to have consecutive submatrices follow the matrix row order from top to bottom.

Like Warren's algorithm, the tile function of this algorithm omits elements in $M_{\mathbb{I}}$. The ordering of the submatrices in the constructed partition ensures that submatrices involving entries below $M_{\mathbb{I}}$ are processed before the submatrices involving entries above $M_{\mathbb{I}}$ are processed. Each submatrix consists of entries in a number of consecutive rows limited to entries on one side of the main diagonal of the matrix. I call this tiling strategy *k-Row Diagonal* because submatrices are formed by taking the entries in a block with a fixed number of rows and dividing it into two submatrices using the main diagonal of the matrix as the border between the two submatrices.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Recall from Section 4.4.3 that a row of a Boolean matrix is a subsequence of the matrix and that $M[x, \star]$ denotes row x of matrix M. In this algorithm a partition is formed in which each of the submatrices contains only a portion of the elements in any given row.

The following are predicates that hold as a consequence of the way the submatrices of the partition are constructed. These are used in the arguments verifying the correctness of this algorithm. The number of sections created in the partitioning function is denoted by s. The partition comprises a total of $2 \times s$ submatrices.

$$i > j \iff \langle \exists k \mid 0 < k < s \mid m_{ij} \in P_k \rangle \tag{17.1}$$

$$i < j \iff \langle \exists k \mid s \leq k < 2 \times s \mid m_{ij} \in P_k \rangle \tag{17.2}$$

$$a < b \iff \langle \exists k, t \mid 0 \leq k \leq t < s \mid M[a, \star] \cap P_k \neq \varnothing \wedge M[b, \star] \cap P_t \neq \varnothing \rangle \tag{17.3}$$

$$a < b \iff \langle \exists k, t \mid s \leq k \leq t < 2 \times s \mid M[a, \star] \cap P_k \neq \varnothing \wedge M[b, \star] \cap P_t \neq \varnothing \rangle \tag{17.4}$$

When implementing the algorithm, the size of the available memory for data manipulation and the size of the relation whose transitive closure is to be determined should be known. These values are taken into account when constructing the partition. Submatrices in the partition are constructed to accommodate the maximum number of complete rows of the adjacency matrix that can fit into the available memory at one time. The submatrix size should be one less row than this determined size. It is chosen to be this size to leave enough space to load one additional row. It may be necessary to load additional rows into memory to access elements that are not part of the submatrix that is being processed.

This algorithm prescribes a strategy for memory management. The implementation of the strategy at the level applied by this algorithm is operating system and hardware specific. When the entries in a given submatrix are processed, all the rows that intersect with the submatrix are fully loaded into memory. The processing of elements in the submatrix may require data in parts of the matrix that are not currently loaded. When data are needed to complete the processing of an element, one entire row of a part of the matrix that is needed but not part of the current submatrix, is loaded at a time. This means that at any point during operation the memory contains the rows that intersects with the submatrix being processed plus one additional row as needed.

Entries within the submatrix are processed in column major order so that all entries in memory that need the data of a given row are processed before the row is swopped out. The shape of the submatrices in the partition created by Warren's algorithm is the same as that of the special case of this algorithm where the whole matrix can fit into memory. Although the submatrices of these two partitions have the same elements, they are not the same because the order of the entries differ. Warren's submatrices are processed in row major order whereas the entries in the equivalent submatrices in the special case of this algorithm are processed in column major order. Because of this change in processing order it is likely that Algorithm 17.3.1 (BlockRow) will outperform Algorthm 17.1.1 (Warren) owing to better memory management.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

17.3.1 Visualisation

Figure 17.3.1 is a diagram showing an example of a partition $\langle\!\langle\ i \mid i \in \mathbb{N}_{14} \mid P_i\ \rangle\!\rangle$ that is created by the blocked row algorithm. It comprises 14 submatrices. In this example the values of m and n are such that $\lceil n/(m-1) \rceil = 7$. Note that the bottom horizontal section may be smaller than the other horizontal sections as the required memory size need not be a factor of the matrix size.

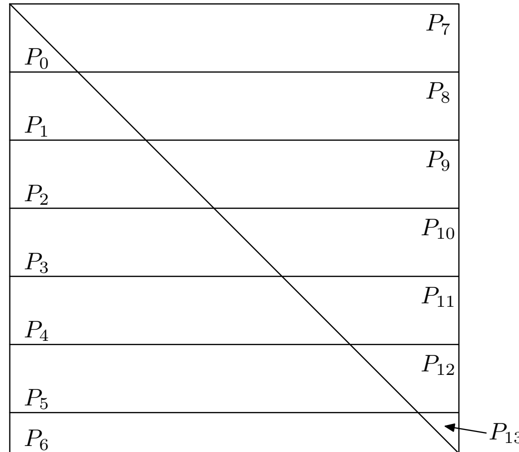


Figure 17.3.1: Submatrices formed by the blocked row algorithm

17.3.2 Complexity

The complexity of this algorithm is governed by the number of submatrices that are formed, which is dependant on the relation size and on the size of the submatrices that are created. These sizes are determined by the number of bytes available for use, which in turn is determined by the physical requirements for optimal performance. Let m be the size of the available memory, p be the number of submatrices formed by the algorithm and n be the total number of rows in the matrix being manipulated.

Typically m will be fixed. While m remains constant, p increases at a faster rate than n. When n doubles, the size of the rows doubles and consequently the number of rows that can fit in the same size memory halves, resulting in having to form four times as many submatrices. There are limits in terms of matrix size and available memory for which the algorithm is feasible.

The algorithm starts with a function call. This function consist of a loop that iterates $\frac{1}{2} \times p$ times. When forming submatrices, a number of sections is formed. Each of these sections is divided into two submatrices. As each of these sections consist of at least one row, there can be at most n sections. Thus, as $p \leq (n \times 2)$, the loop iterates at most $\frac{1}{2} \times (n \times 2) = n$ times. Thus the complexity of this function is linear and bounded by the size of the matrix, i.e. it is $O(n)$. The rest of the body

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

of the algorithm consists of three nested loops. These loops are called the inner loop, the middle loop and the outer loop.

The outer loop is performed p times. It has already been established that $p \leq n \times 2$. Thus complexity of the outer loop is $O(n \times 2) = O(n)$.

In each case the number of iterations of the middle loop is dependent on the number of entries in the submatrix being processed. Let r denote the number of rows in a given submatrix. For any partition r is the same fixed value for all submatrices in the partition. The number of columns in each submatrix in the partition, however, range from 1 to n. This variation is regular in the sense that when pairing the submatrices that share the same rows, the total number of entries in each such pair of submatrices combined is $n \times r$. This pairing ensures that the average number of entries in each submatrix in the partition is $\frac{n \times r}{2}$. Each submatrix has strictly less entries than $n \times r$. The number of iterations of the middle loop is thus bounded by $n \times r$. Furthermore r is bounded by n. Consequently the number of iterations is bounded by $n \times n$. Thus, the complexity of the middle loop is $O(n^2)$.

Although it may often happen that the operation of the inner loop is not performed, the worst case assumes that it is performed. It requires n operations to add an entire row in the matrix to another row in the matrix. The complexity of this operation is $\Theta(n)$. The complexity of the nested loops in this algorithm is the product of the cost of the individual loops i.e. $O(n) \times O(n^2) \times \Theta(n) = O(n \times n^2 \times n) = O(n^4)$. The complexity of the entire algorithm is the sum of the cost of the initial function and the cost of the nested loops i.e. $O(n) + O(n^4) = O(n + n^4) = O(n^4)$.

In the case where the whole matrix can fit into memory the algorithm is similar to Warren's algorithm. It has the same elements in the submatrices in its partition and performs the same operations on the elements in these submatrices. The elements in the submatrices of these algorithms are, however, processed in different orders. Because the order of processing has no impact on the complexity, these algorithms are equivalent in terms of complexity. In Section 17.1.2 it was shown that the complexity of Warren's algorithm is $O(n^3)$.

It can thus be concluded that the complexities of the blocked row algorithm are $O(n^4)$ and $\Omega(n^3)$. The TM specification of these complexities of Agrawal's blocked row algorithm in the TM of TCA is shown in Listing C39.

17.3.3 Position in a derivation tree

Similar to Warren's algorithm, this algorithm is derived from the abstract tiling algorithm. Figure 17.3.3 shows this.

17.4 Verification of the blocked row algorithm

17.4.1 The tile function returns a valid partition

When Algorithm 17.3.1 (BlockRow) constructs the sequence $\langle\!\langle\, t \mid t \in \mathbb{N}_{2\times s} \mid P_t \,\rangle\!\rangle$, each entry of M, with the exclusion of entries on the main diagonal of M, is

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

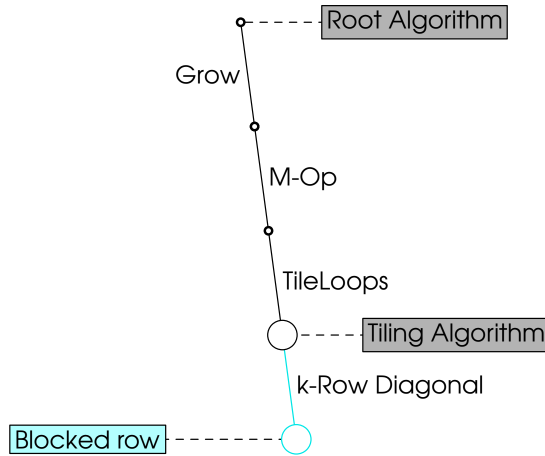


Figure 17.3.3: Derivation tree of the blocked row algorithm

uniquely assigned to one of the submatrices in this sequence. Any given $m_{ij} \in M$ with $i \neq j$ is an entry in exactly one of the submatrices. Thus the submatrices in the sequence are mutually exclusive as well as exhaustive with respect to $M - M_{\mathbb{I}}$; i.e. $\langle\!\langle\ t \mid t \in \mathbb{N}_{2\times s} \mid P_t\ \rangle\!\rangle \vdash M - M_{\mathbb{I}}$.

As explained in Section 16.3.1, the processing of the entries on the main diagonal of the matrix in grow algorithms is redundant and may thus be omitted. The processing of $\langle\!\langle\ t \mid t \in \mathbb{N}_{2\times s} \mid P_t\ \rangle\!\rangle$ as specified in Algorithm 16.4.1 (TileSkeleton) will thus yield the transitive closure if it can be verified that the processing of the entries in this specified order complies with the required processing order constraints.

17.4.2 Primary order constraint - part 1

Lemma 17.4.2 shows that Algorithm 17.3.1 (BlockRow) complies with the primary order constraint if both entries on the row are below the diagonal.

17.4.3 Primary order constraint - part 2

Lemma 17.4.2 shows that Algorithm 17.3.1 (BlockRow) complies with the primary order constraint if the entries on the row are on different sides of the diagonal.

17.4.4 Primary order constraint - part 3

Lemma 17.4.2 shows that Algorithm 17.3.1 (BlockRow) complies with the primary order constraint if both entries on the row are above the diagonal.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$y_0 < y_1 < x$

$\Rightarrow$ { Assume s is the number of sections created in the partitioning function. Select $t < s$ such that $M[x,\star] \cap P_t \neq \varnothing$. It is given that $y_0 < x$ and $y_1 < x$, thus Predicate 17.1 holds. }

$(y_0 < y_1) \wedge (M[x,y_0] \in P_t) \wedge (M[x,y_1] \in P_t)$

$\Rightarrow$ { P_t is processed in column major order from top to bottom, thus $M[\star,y_0] \lhd M[\star,y_1]$ }

$M[x,y_0] \lhd M[x,y_1]$

Lemma 17.4.2: $(y_0 < y_1 < x) \Rightarrow M[x,y_0] \lhd M[x,y_1]$

$y_0 < x < y_1$

$\Rightarrow$ { Assume s is the number of sections created in the partitioning function. Select $t < s$ such that $M[x,\star] \cap P_t \neq \varnothing$. It is given that $y_0 < x$ and $y_1 > x$, thus Predicate 17.1 and Predicate 17.2 hold and the submatrices for these entries are formed from the same section }

$(M[x,y_0] \in P_t) \wedge (M[x,y_1] \in P_{t+s})$

$\Rightarrow$ $\{ t < t+s \Rightarrow P_t \lhd P_{t+s} \}$

$M[x,y_0] \lhd M[x,y_1]$

Lemma 17.4.3: $(y_0 < x < y_1) \Rightarrow M[x,y_0] \lhd M[x,y_1]$

$x < y_0 < y_1$

$\Rightarrow$ { Assume s is the number of sections created in the partitioning function. Select $t < s$ such that $M[x,\star] \cap P_t \neq \varnothing$. It is given that $x < y_0$ and $x < y_1$, thus Predicate 17.2 holds. }

$(y_0 < y_1) \wedge (M[x,y_0] \in P_{t+s}) \wedge (M[x,y_1] \in P_{t+s})$

$\Rightarrow$ { P_{t+s} is processed in column major order from top to bottom, thus $M[\star,y_0] \lhd M[\star,y_1]$ }

$M[x,y_0] \lhd M[x,y_1]$

Lemma 17.4.4: $(x < y_0 < y_1) \Rightarrow M[x,y_0] \lhd M[x,y_1]$

17.4.5 Secondary order constraint in the lower triangle

Lemma 17.4.5 shows that Algorithm 17.3.1 (BlockRow) complies with the secondary order constraint in the lower triangle.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$y < x$

$\Rightarrow$ { Let $M[x_0, y_0] \in \langle\!\langle\, i, j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle$ }

$y < x \wedge x_0 = y \wedge y_0 < x_0$

$\Rightarrow$ { $\wedge$ is commutative and associative. Also $\wedge$ is Idempotent, thus the predicate $y < x$ my be duplicated. }

$(x_0 = y \wedge y < x) \wedge y < x \wedge y_0 < x_0$

$\Rightarrow$ { Simplify the first term }

$x_0 < x \wedge y < x \wedge y_0 < x_0$

$\Rightarrow$ { Assume s is the number of sections created in the partitioning function. Predicate 17.1 holds for both $M[x, y]$ and $M[x_0, y_0]$ }

$x_0 < x \wedge (M[x, y] \in P_t \wedge t < s) \wedge (M[x_0, y_0] \in P_k \wedge k < s)$

$\Rightarrow$ { Predicate 17.3 }

$k < t \wedge (M[x, y] \in P_t) \wedge (M[x_0, y_0] \in P_k)$

$\Rightarrow$ { $k < t \Rightarrow P_k \lhd P_t$ }

$M[x_0, y_0] \lhd M[x, y]$

Lemma 17.4.5: $y < x \Rightarrow \langle\!\langle\, i, j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd M[x, y]$

17.4.6 Secondary order constraint in the upper triangle

Lemma 17.4.6 shows that Algorithm 17.3.1 (BlockRow) complies with the secondary order constraint in the upper triangle.

17.4.7 Conclusion

Let $M[x, y_0]$ and $M[x, y_1]$ be two entries on the same row in M with $y_0 < y_1$. Lemma 17.4.2 shows that $M[x, y_0] \lhd M[x, y_1]$ if $y_0 < y_1 < x$. Lemma 17.4.3 shows that $M[x, y_0] \lhd M[x, y_1]$ if $y_0 < x < y_1$. Lemma 17.4.4 shows that $M[x, y_0] \lhd M[x, y_1]$ if $x < y_0 < y_1$. These three lemmata cover all possible positions of x in relation to y_0 and y_1 and collectively show that the partitioning strategy of this algorithm complies with the primary constraint. Lemmata 17.4.5 and 17.4.6 show that the partitioning strategy of this algorithm complies with the secondary constraint respectively for the case when $x > y$, and when $x < y$. Collectively these lemmata show that the algorithm complies with the secondary constraint. As all the requirements are met, it can be concluded that Algorithm 17.3.1 (BlockRow) is correct.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$y > x$

$\Rightarrow$ { Let $M[x_0, y_0] \in \langle\!\langle\, i, j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle$ }

$(y > x) \wedge (x_0 = y) \wedge (y_0 < x_0)$

$\Rightarrow$ { Weaken predicate }

$y > x \wedge y_0 < x_0$

$\Rightarrow$ { Assume s is the number of sections created in the partitioning function. Predicate 17.1 holds for $M[x_0, y_0]$ while Predicate 17.2 holds for $M[x, y]$ }

$(M[x, y] \in P_t \wedge t \geq s) \wedge (M[x_0, y_0] \in P_k \wedge k < s)$

$\Rightarrow$ { $\wedge$ is associative and commutative }

$(t \geq s \wedge s > k) \wedge M[x, y] \in P_t \wedge M[x_0, y_0] \in P_k$

$\Rightarrow$ { Simplify the first term }

$k < t \wedge (M[x, y] \in P_t) \wedge (M[x_0, y_0] \in P_k)$

$\Rightarrow$ { $k < t \Rightarrow P_k \lhd P_t$ }

$M[x_0, y_0] \lhd M[x, y]$

Lemma 17.4.6: $y > x \Rightarrow \langle\!\langle\, i, j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd M[x, y]$

17.5 Blocked column algorithm

Where Algorithm 17.3.1 (BlockRow) is a tiled version of Algorithm 17.1.1 (Warren), this algorithm is a tiled version of Algorithm 12.6.1 (Warshall). The TM specification of this algorithm in the TM of TCA is given in Listing C30. The topics and relations that are defined in this specification are shown in Figure 17.5.1. Listing C10 contains the definition of the tiling strategy, called *k-Col Triplet*, used by this algorithm.

17.5.1 Tiling strategy of the blocked column algorithm

The submatrices of the partition are determined in terms of sections covering specified columns of the adjacency matrix. When specifying the tiling strategy for this algorithm, the adjacency matrix M is divided into vertical sections each comprising the determined number of columns. Each of these vertical sections are further divided into three submatrices. The submatrices of a given section S are called the top submatrix of S (S_{top}), the middle submatrix of S (S_{mid}) and the bottom submatrix of S (S_{bot}) defined as follows:

$$
\begin{aligned}
S_{top} &= \langle\!\langle\, i, j \mid m_{ij} \in S \wedge m_{ii} \notin S \wedge i < j \mid m_{ij} \,\rangle\!\rangle \\
S_{mid} &= \langle\!\langle\, j, i \mid m_{ij} \in S \wedge m_{ii} \in S \mid m_{ij} \,\rangle\!\rangle \\
S_{bot} &= \langle\!\langle\, i, j \mid m_{ij} \in S \wedge m_{ii} \notin S \wedge i > j \mid m_{ij} \,\rangle\!\rangle
\end{aligned}
$$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

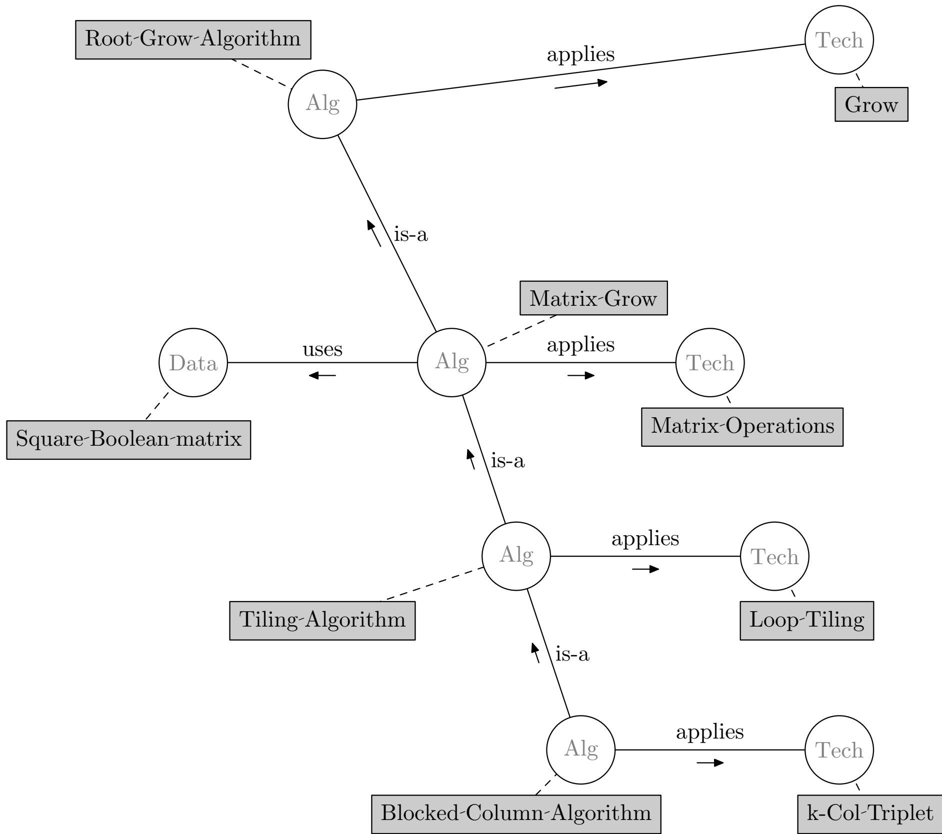


Figure 17.5.1: Topics related to Agrawal's blocked column algorithm

This subdivision is illustrated in Figure 17.5.1. The section S is the shaded area. S_{mid} is the square submatrix that contains the entries in S that are on the main diagonal of the matrix ($M_{\mathbb{I}}$), where the other two submatrices are the remaining parts of S respectively above, and below the square submatrix.

The number of columns in a section for the tiling strategy is determined by the size of the available memory for data manipulation as well as the size of the relation of which the transitive closure is to be determined. Although the tiling strategy of this algorithm differs from that of Algorithm 17.3.1 (BlockRow), the core data manipulation is the same. It involves manipulating complete rows of the adjacency matrix. Thus the number of columns in a section to be used in the definition of the submatrices in the partition is ultimately determined by the number of complete rows of the adjacency matrix that can fit into the available memory. When processing the elements of a submatrix, the rows with numbers that correspond with the column numbers of the elements the submatrix are loaded into memory. The submatrix width is thus typically one less than the

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

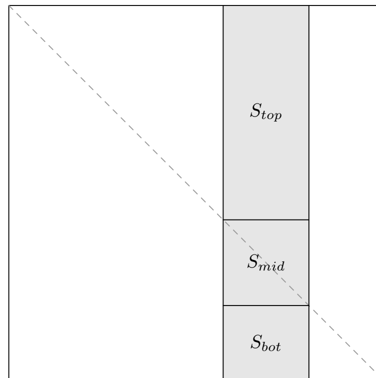


Figure 17.5.1: Subdivision of a vertical section formed by the algorithm

number of rows of the matrix that can fit in memory. This ensures that all data needed to process the middle matrix can fit in memory at the same time. The reason for having room for one more row will become evident when the processing of the top and bottom submatrices are discussed.

When the entries in a given section are processed, all the rows which intersect with S_{mid} are fully loaded into memory and kept in memory while processing the three submatrices which are part of the section. The entries in S_{mid} are processed first. They are processed in column major order. The data needed to process these entries, are exactly the rows that are loaded into memory.

After processing the elements in S_{mid}, the elements in S_{top} and then the elements in S_{bot} are processed. The entries in S_{top} and S_{bot} are processed in row major order. When processing a row, the entire row is loaded into memory. The data needed to process entries in this row that intersect with S are at most this row and the rows that intersect with S_{mid}, which are already in memory. After processing the entries of one row, the row is swopped out to load the next row to be processed.

17.5.2 Visualisation

Figure 17.5.2 is a diagram visualising the 21 submatrices $\langle\!\langle\ i \mid i \in \mathbb{N}_{21} \mid P_i\ \rangle\!\rangle$ of a partition of the matrix that will be created by the blocked column algorithm if the values of m and n are such that 7 column-wise sections are formed when the submatrices of the partition are determined. Note that the rightmost vertical section may be smaller than the other vertical sections as the required memory size need not be a factor of the matrix size. P_1 and P_{20} do not appear in the diagram. According to their definitions P_1 has to intersect with rows above $M[0,\star]$, while P_{20} has to intersect with rows below $M[n-1,\star]$. As the mentioned rows do not exist, both these submatrices are empty.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

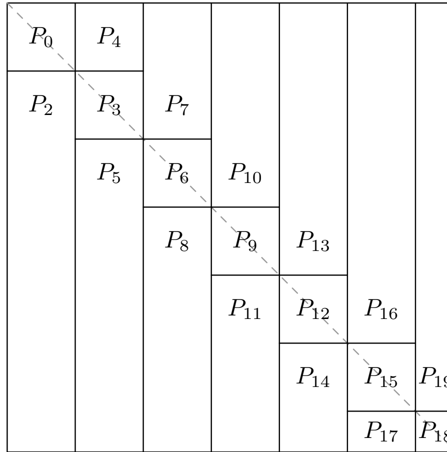


Figure 17.5.2: Submatrices of the partition formed by the algorithm

17.5.3 Specification

Algorithm 17.5.3: Tile function of the blocked column algorithm

---

func tile$(M \in \mathbb{B}[n,n], m \in \mathbb{N}) : \langle\!\langle\, t \mid t \in \mathbb{N}, P_t \subseteq M \mid P_t \,\rangle\!\rangle$
    var $s \in \mathbb{N}$

    $s, t := \lceil n/(m-1) \rceil, 0;$
    do $(t < s)$
        $x := t \times (m-1);$
        $P_{3\times t} := \langle\!\langle\, j, i \mid x < i < x+m, x < j < x+m \mid m_{ij} \,\rangle\!\rangle;$ $\{middle\}$
        $P_{3\times t+1} := \langle\!\langle\, i, j \mid i \leq x, x < j < x+m \mid m_{ij} \,\rangle\!\rangle;$ $\{top\}$
        $P_{3\times t+2} := \langle\!\langle\, i, j \mid i \geq x+m, x < j < x+m \mid m_{ij} \,\rangle\!\rangle;$ $\{bottom\}$
        $t := t+1;$
    od
    $\{\langle\!\langle\, t \mid t \in \mathbb{N}_{3\times s} \mid P_t \,\rangle\!\rangle \vdash M\}$
    return $\langle\!\langle\, t \mid t \in \mathbb{N}_{3\times s} \mid P_t \,\rangle\!\rangle$
cnuf

Assume that $M[x_0, y_0] \in P_k \wedge M[x_1, y_1] \in P_t \wedge (k \neq t)$, i.e. $M[x_0, y_0]$ and $M[x_1, y_1]$ are entries in different submatrices of the partition that was constructed by the tile function of Algorithm 17.5.3 (BlockCol).

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The following are predicates that hold as a consequence of the way the submatrices of the partition are constructed. These are used in the arguments verifying the correctness of this algorithm.

$$y_0 < y_1 \land x_0 = x_1 \iff k < t \tag{17.5}$$

$$k \% 3 = 0 \iff M[x_0, x_0] \in P_k \land M[y_0, y_0] \in P_k \tag{17.6}$$

$$y_0 = y_1 \implies \lfloor k/3 \rfloor = \lfloor t/3 \rfloor \tag{17.7}$$

Predicate 17.5 specifies that if $M[x_0, y_0]$ and $M[x_1, y_1]$ are in different submatrices that both intersect with the same row, the submatrix containing the entry with lowest column value is in the submatrix with a lower index number and vice versa.

Predicate 17.6 specifies that the submatrix in a given vertical section with the lowest index number (its index number is divisable by 3) is the middle submatrix, i.e. the submatrix intersects with $M_{\mathbb{I}}$.

Predicate 17.7 specifies that when different submatrices both intersect with the same column, they are in the same vertical section. If the vertical sections are numbered from left to right, the vertical section number that intersects with submatrix P_t can be expressed as $\lfloor t/3 \rfloor$.

17.5.4 Complexity

Like the Algorithm 17.3.1 (BlockRow), the complexity of this algorithm is governed by the size of the available memory as well as the size of the relation. Let m be the size of the available memory and assume the size of the matrix being manipulated is $n \times n$. Let r be the number of rows that can fit in the available memory at one time. r is a function of m and n and can be calculated with the formula $r = \frac{n}{m}$.

The algorithm starts with a function call. This function consist of a loop that iterates exactly r times where $1 \leq r \leq n$. Thus the loop iterates at most n times. The complexity of this function is linear and bounded by the size of the matrix, i.e. it is $O(n)$. The rest of the body of the algorithm consists of three nested loops. These loops are called the inner loop, the middle loop and the outer loop.

The outer loop is executed p times, where p is the number of submatrices that are formed. When forming the submatrices, r sections are formed. It has already been established that the number of sections are at most n. Each section is divided into three submatrices. Thus complexity of the outer loop is $O(n \times 3) = O(n)$.

In each case the number of iterations of the middle loop depends on the number of entries in the submatrix being processed. The number of columns in a submatrix is fixed at r. The number of rows in a middle submatrix is the same as the number of columns, namely r, but the number of rows in the top and bottom submatrices range from 0 to $n - r$. The total number of entries in the submatrices that intersect with the rightmost section may be smaller. The number of entries in any submatrix is thus strictly smaller than $n \times r$. Furthermore $1 \leq r \leq n$, thus the number of iterations of this loop is bounded by $n \times n$. Consequently the complexity of the middle loop is $O(n^2)$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Although it may often happen that the operation of the inner loop is not performed, the worst case assumes that it is performed. It requires n operations perform the inner loop. The complexity of this operation is $\Theta(n)$.

The complexity of the nested loops in this algorithm is the product of the cost of the individual loops i.e. $O(n) \times O(n^2) \times O(n) = O(n \times n^2 \times n) = O(n^4)$. The complexity of the entire algorithm is the sum of the cost of the initial function and the cost of the nested loops i.e. $O(n) + O(n^4) = O(n + n^4) = O(n^4)$.

In the case where the whole matrix can fit into memory, the algorithm is equivalent to Warshall's algorithm. Its partition will have one section containing a middle submatrix consisting of the entire matrix and two empty submatrices. The processing of the entries in this middle submatrix is exactly the same as Warshall's algorithm. In Section 12.6.5 it was shown that the complexity of Warshall's algorithm is $O(n^3)$. It can thus be concluded that the lower bound for the complexity of the blocked column algorithm is $\Omega(n^3)$. The TM specification of these complexities in the TM of TCA is shown in Listing C40.

The TM specification pointing to implementations of Agrawal's blocked column algorithm in the TM of TCA can be found in Listing C60. In these implementations a constant is declared that should be used to set the available memory size as is determined by the hardware configuration.

17.5.5 Position in a derivation tree

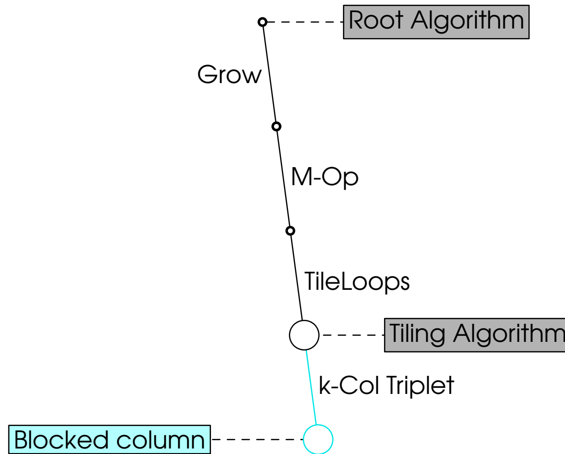


Figure 17.5.5: Derivation tree of Agrawal's blocked column algorithm

As is the case for Warren's algorithm and the blocked row algorithm, this algorithm is derived from the abstract tiling algorithm. Figure 17.5.5 shows this.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

17.6 Verification of the blocked column algorithm

17.6.1 The tile function returns a valid partition

Let s denote the number of vertical sections used during construction of the submatrices. When Algorithm 17.5.3 (BlockCol) constructs the sequence $\langle\!\langle\ t \mid t \in \mathbb{N}_{3\times s} \mid P_t\ \rangle\!\rangle$ each, entry of M is uniquely assigned to one of the submatrices in this sequence. Any given $m_{ij} \in M$ is an entry in exactly one of the constructed submatrices. Thus the submatrices in the sequence are mutually exclusive as well as exhaustive with respect to M; i.e. $\langle\!\langle\ t \mid t \in \mathbb{N}_{3\times s} \mid P_t\ \rangle\!\rangle \vdash M$.

17.6.2 Compliance with the primary processing order constraint

Let $M[x, y_0]$ and $M[x, y_1]$ be two entries on the same row in M with $y_0 < y_1$. I will argue for all possibile allocations of y_0 and y_1 to submatrices in a partition of M that was constructed using the tiling strategy of this algorithm, that $M[x, y_0] \lhd M[x, y_1]$.

$M[x, y_0]$ and $M[x, y_1]$ are either in the same vertical section or not in the same vertical section. If they are not in the same vertical section, $M[x, y_0]$ and $M[x, y_1]$ are allocated to different submatrices. If they are in the same vertical section, $M[x, y_0]$ and $M[x, y_1]$ are allocated to the same submatrix as entries in the same row are allocated to the same submatrix when subdividing a vertical section. Therefore there are only four possibilities:

- $M[x, y_0]$ and $M[x, y_1]$ are allocated submatrices in different vertical sections
- $M[x, y_0]$ and $M[x, y_1]$ are both allocated to a middle submatrix
- $M[x, y_0]$ and $M[x, y_1]$ are both allocated to a top submatrix
- $M[x, y_0]$ and $M[x, y_1]$ are both allocated to a bottom submatrix

If $M[x, y_0]$ and $M[x, y_1]$ are in different submatrices $M[x, y_0] \lhd M[x, y_1]$ because Predicate 17.5 guarantees that the submatrix containing $M[x, y_0]$ has a lower index value and therefore will be processed first.

If $M[x, y_0]$ and $M[x, y_1]$ are both in the same middle submatrix, $M[x, y_0] \lhd M[x, y_1]$ because the middle submatrices are processed in column major order, i.e $M[\star, y_0] \lhd M[\star, y_1]$.

If $M[x, y_0]$ and $M[x, y_1]$ are both in the same top submatrix, $M[x, y_0] \lhd M[x, y_1]$ because the entries in the top submatrix are processed in row major order. The same argument applies if $M[x, y_0]$ and $M[x, y_1]$ are both in the same bottom submatrix.

It can therefore be concluded that the partitioning strategy of this algorithm complies with the primary processing order constraint.

17.6.3 Secondary order constraint - part 1

Let $M[x, y] \in P_t$ and $M[x_0, y_0] \in \langle\!\langle\ i, j \mid i = y \wedge j < i \mid m_{ij}\ \rangle\!\rangle$ and let S is the vertical section containing P_t. Lemma 17.6.3 shows that if $M[x_0, y_0]$ is in S_{mid} and $P_t \neq S_{mid}$ then $M[x_0, y_0]$ is processed before $M[x, y]$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & M[x_0,y_0] \in P_k \wedge M[x,y] \in P_t \wedge t \neq k \\
\Rightarrow & \quad \{ t \neq k \text{ implies that } P_t \neq P_k \} \\
 & M[x_0,y_0] \in P_k \wedge M[x,y] \in P_t \wedge P_k \neq P_t \\
\Rightarrow & \quad \{ P_k \text{ and } P_t \text{ both intersect } S \text{ and } P_k = S_{mid}, \text{ thus } P_k \lhd P_t \} \\
 & M[x_0,y_0] \lhd M[x,y]
\end{array}
$$

Lemma 17.6.3: $M[x_0,y_0] \in P_k \wedge t \neq k \Rightarrow M[x_0,y_0] \lhd M[x,y]$

17.6.4 Secondary order constraint - part 2

Let $M[x,y] \in P_t$ and $M[x_0,y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle$ and let S is the vertical section containing P_t. Lemma 17.6.4 shows that if $M[x_0,y_0]$ is in S_{mid} and $P_t = S_{mid}$ then $M[x_0,y_0]$ is processed before $M[x,y]$.

$$
\begin{array}{ll}
 & M[x_0,y_0] \in P_k \wedge t = k \\
\Rightarrow & \quad \{ \text{k = t implies that } P_t = P_k \} \\
 & M[x_0,y_0] \in P_k \wedge M[x,y] \in P_k \\
\Rightarrow & \quad \{ M[x_0,y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \} \\
 & M[x_0,y_0] \in P_k \wedge y_0 < y \wedge M[x,y] \in P_k \\
\Rightarrow & \quad \left\{ \begin{array}{l} P_k \text{ is processed in column major order;} \\ \text{i.e } M[\star,y_0] \cap P_k \lhd M[\star,y] \cap P_k \end{array} \right\} \\
 & M[x_0,y_0] \lhd M[x,y]
\end{array}
$$

Lemma 17.6.4: $M[x_0,y_0] \in P_k \wedge t = k \Rightarrow M[x_0,y_0] \lhd M[x,y]$

17.6.5 Secondary order constraint - part 3

Let $M[x,y] \in P_t$ and $M[x_0,y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle$ and let S be the vertical section containing P_t. Lemma 17.6.5 shows that if $M[x_0,y_0] \notin S_{mid}$ and $P_t \neq S_{mid}$ then $M[x_0,y_0]$ is processed before $M[x,y]$.

17.6.6 Secondary order constraint - part 4

Let $M[x,y] \in P_t$ and $M[x_0,y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle$ and let S is the vertical section containing P_t. Lemma 17.6.6 shows that if $M[x_0,y_0] \notin S_{mid}$ and $P_t = S_{mid}$ then $M[x_0,y_0]$ is processed before $M[x,y]$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

$$
\begin{array}{ll}
 & M[x_0,y_0] \notin P_k \wedge M[x,y] \in P_t \wedge t \neq k \\
\Rightarrow & \{\, t \neq k \text{ implies that } P_t \neq P_k \,\} \\
 & M[x_0,y_0] \notin P_k \wedge M[x,y] \in P_t \wedge P_k \neq P_t \\
\Rightarrow & \{\, P_k \text{ and } P_t \text{ both intersect } S \text{ and } P_k = S_{mid}, \text{ thus } P_k \lhd P_t \,\} \\
 & M[x_0,y_0] \notin P_k \wedge P_k \lhd M[x,y] \\
\Rightarrow & \{\, M[x_0,y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \text{ and } P_k = S_{mid} \,\} \\
 & M[x_0,y_0] \notin P_k \wedge x_0 = y \wedge y_0 < y \wedge M[y,y] \in P_k \wedge P_k \lhd M[x,y] \\
\Rightarrow & \left\{ \begin{array}{l} \text{Predicate 17.5; i.e. the submatrix containing } M[y,y_0] \\ \text{is processed before the submatrix containing } M[y,y] \end{array} \right\} \\
 & M[x_0,y_0] \lhd M[y,y] \wedge M[y,y] \in P_k \wedge P_k \lhd M[x,y] \\
\Rightarrow & \{\, \lhd \text{ is transitive} \,\} \\
 & M[x_0,y_0] \lhd M[x,y]
\end{array}
$$

Lemma 17.6.5: $M[x_0,y_0] \notin P_k \wedge t \neq k \Rightarrow M[x_0,y_0] \lhd M[x,y]$

$$
\begin{array}{ll}
 & M[x_0,y_0] \notin P_k \wedge M[x,y] \in P_t \wedge t = k \\
\Rightarrow & \{\, t = k \text{ implies that } P_t = P_k \,\} \\
 & M[x_0,y_0] \notin P_k \wedge M[x,y] \in P_k \\
\Rightarrow & \{\, M[x_0,y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \text{ and } P_k = S_{mid} \,\} \\
 & M[x_0,y_0] \notin P_k \wedge x_0 = y \wedge y_0 < y \wedge M[y,y] \in P_k \wedge M[x,y] \in P_k \\
\Rightarrow & \left\{ \begin{array}{l} \text{Predicate 17.5; i.e. the submatrix containing } M[y,y_0] \\ \text{is processed before the submatrix containing } M[y,y] \end{array} \right\} \\
 & M[x_0,y_0] \lhd P_k \wedge M[x,y] \in P_k \\
\Rightarrow & \{\, M[x_0,y_0] \lhd \text{ the submatrix that contains } M[x,y] \,\} \\
 & M[x_0,y_0] \lhd M[x,y]
\end{array}
$$

Lemma 17.6.6: $M[x_0,y_0] \notin P_k \wedge t = k \Rightarrow M[x_0,y_0] \lhd M[x,y]$

17.6.7 Compliance with the secondary order constraint

Let $M[x,y] \in P_t$ and $M[x_0,y_0] \in \langle\!\langle\, i,j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle$. Let S be the vertical section such that $P_t \subset S$. Select k such that $M[y,y] \in P_k$.

As observed in Predicate 17.7, P_t and P_k are in the same vertical section; i.e. $P_k \subset S$. Per definition of S_{mid}, it is evident that $P_k = S_{mid}$.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Lemma 17.6.3 shows that $M[x_0, y_0] \in P_k \wedge t \neq k \Rightarrow M[x_0, y_0] \lhd M[x, y]$.
Lemma 17.6.4 shows that $M[x_0, y_0] \in P_k \wedge t = k \Rightarrow M[x_0, y_0] \lhd M[x, y]$.
Lemma 17.6.5 shows that $M[x_0, y_0] \notin P_k \wedge t \neq k \Rightarrow M[x_0, y_0] \lhd M[x, y]$
Lemma 17.6.6 shows that $M[x_0, y_0] \notin P_k \wedge t = k \Rightarrow M[x_0, y_0] \lhd M[x, y]$.

Collectively the above lemmata establish that

$$\langle\!\langle\, i, j \mid i = y \wedge j < i \mid m_{ij} \,\rangle\!\rangle \lhd M[x, y]$$

i.e. compliance with the secondary processing order constraint.

17.6.8 Conclusion

The proofs of these lemmata are valid irrespective of whether $P_t = S_{top}$, $P_t = S_{mid}$ or $P_t = S_{bot}$. In fact these lemmata hold independent of whether or not the other rows in S_{top} and S_{bot} have been processed. The algorithm is, therefore, correct regardless of the order in which the rows in S_{top} and S_{bot} are picked to be processed. As long as the elements in each row are processed from left to right, the order constraints would not be violated if the rows in S_{top} and S_{bot} were picked in a different order, or even picked in a random order. In fact the rows in S_{top} and S_{bot} may be processed in parallel.

Agrawal and Jagadish [3] propose a parallel algorithm that applies this observation. Many other parallel algorithms exist, among which are those discussed by Houtsma *et al.* [116], Pagourtzis *et al.* [185], Scheiman and Cappello [214], Schudy [221], Ullman and Yannakakis [239]. The discussion of these algorithms are beyond the scope of this thesis.

17.7 Summary

The three algorithms presented in this chapter are implementations of the abstract tiling algorithm presented in Chapter 16. These are all based on algorithms previously described in the literature. Algrorithm 17.1.1 (Warren) was published by Warren [251] in 1975 while both Algorithm 17.3.1 (BlockRow) and Algorithm 17.5.3 (BlockCol) were described by Agrawal and Jagadish [2] in 1987. The way in which these algorithms are interpreted and described is novel.

Warren specifies his algorithm as a variation of Warshall's algorithm but discusses it as a variation of the algorithms of Baker and of Martynyuk. Warshall shows that loop interchange similar to how I have done it in Section 13.1.1 does not work in general. He then offers the solution to use a strategy to perform two passes respectively on the lower half and the upper half of the adjacency matrix of the relation being processed where Baker's algorithm uses a change monitor and Martynyuk's algorithm uses a safe upper bound. After much deliberation I concluded that Warren's algorithm is best described as a tiling algorithm. The proof of the correctness of the algorithm in this chapter is thus done from scratch, based on the theory I have developed in Chapter 16.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Agrawal and Jadish describe their algorithms as variations of the respective algorithms of Warshall and Warren. They use successor lists and predecessor lists to represent the relations being manipulated. Their algorithms in the form in which they describe them, would therefore not be derived from Algorithm 12.5.1 (MatrixGrow) as they use an alternative representation model. Here I reformulate their algorithms as derivations of Algorithm 12.5.1 (MatrixGrow) using the adjacency matrix of a relation as the representation model. These descriptions sparked the idea of specifying the concept of a matrix partition and of specifying an abstract tiling algorithm and then neatly formulating these algorithms in terms of matrix partitions that I have defined in Section 4.5 for this purpose. I then show that these algorithms can be derived from Algorithm 16.4.1 (TileSkeleton) that I have specified in Chapter 16 to serve as the common abstraction of these algorithms. Finally I provide novel proofs of the correctness of these algorithms in this context.

The arguments to support the specification of the complexities of these algorithms were newly formulated to provide the artefacts to be included in the TM of TCA for these algorithms. Although these are fairly straightforward, to my knowledge, they have not been published elsewhere.

The next chapter provides a global perspective on all the algorithms discussed in the preceding chapters and discusses the application of short circuiting as an algorithmic technique to each of the algorithms.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 18

Short Circuiting

This chapter serves a dual purpose. On the one hand it is a consolidation of all the algorithms that are discussed in the preceding chapters in Part III of this thesis. It is, however, more than just a summary to highlight the main commonalities and differences between these algorithms. It describes a new variation of each of these algorithms. These variations apply the algorithmic technique of *short circuiting*. Different short circuiting techniques are proposed for different categories of TC algorithms to speed up the overall performance of these algorithms.

18.1 The short circuit technique

In an electrical circuit, the concept of a short circuit is defined as the alteration of the original path of current flow so that the current follows a path that is shorter than the original path. The idea of taking a short cut when performing a computing operation is similar and named accordingly.

An example of the application of this technique is the Boolean operations `&&` and `||` that are defined in programming languages such as Java and C++. The `&&` operator is a short circuit $\wedge$ operator while `||` is a short circuit $\vee$ operator. When two expressions are joined by `&&` and the first one is false, the answer will aways be false. It doesn't matter what the second expression is and hence, it need not be evaluated. Similarly, when two expressions are joined by `||` and the first one is true, the answer will always be true. These operations are implemented to return the result without completing the evaluation of all the terms in an expression if the final outcome is known sooner.

When an algorithm uses a loop structure, the state of known data being processed can sometimes be applied to predict that further processing of possibly unknown data will not change the final outcome of the operation during the current iteration in a loop. When this is the case, processing time can be saved by skipping over the operations that are known to have no effect. An algorithmic technique that applies this principle is called *short circuit*. It takes a short-cut to produce the correct final outcome without performing all the operations. The specification of this technique in the TM of TCA can be found in Listing C8.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

18.2 Short circuiting opportunities

18.2.1 Opportunities in coat algorithms

Algorithms that are derived from Algorithm 12.2.2 (MatrixCoat) apply matrix multiplication as their core operation. Algorithm 18.2.1 (Boolean matrix multiplication) shows the definition of the algorithm to calculate $M_0 \times M_1$ where $M_0 \in \mathbb{B}[n,n]$ and $M_1 \in \mathbb{B}[n,n]$.

Algorithm 18.2.1: Algorithm to calculate $M_0 \times M_1$

---

const $n \in \mathbb{N}^+$
$M_0 \in \mathbb{B}[n,n]$
$M_1 \in \mathbb{B}[n,n]$
var $M_2 \in \mathbb{B}[n,n]$

---

for $i, j : i, j \in \mathbb{N}_n \;\rightarrow$
$M_2[i,j] := \langle\, \exists k \mid k \in \mathbb{N}_n \mid M_0[i,k] \wedge M_1[k,j] \,\rangle$;
rof
{**Post:** $M_2 = M_0 \times M_1$}

This is in accordance with the definition of multiplication (the $\times$ operator) for matrices in Section 4.6.2 where the number of rows in the first matrix is equal to the number of columns in the second matrix and how the Boolean $\vee$ operator is promoted to the matrix level in Section 4.6.5.

The following conditions describe situations where information about the current row or column being processed or the value of a specific entry can be applied to skip operations that will have no effect when calculating the value of $M_2[i,j]$ in a coat algorithm. The algorithm considers the entries of $M_0[i,\star]$ and $M_1[\star,j]$ to calculate this value.

- If either $M_0[i,\star]$ or $M_1[\star,j]$ is empty, i.e. all its entries are not set, it is known that $\langle\, \nexists k \mid k \in \mathbb{N}_n \mid M[i,k] \wedge M[k,j] \,\rangle$. Thus $M_2[i,j]$ should be cleared. This result is known without having to iterate over the individual entries in $M_0[i,\star]$ and $M_1[\star,j]$.

- Assume $k \in \mathbb{N}_n$ is used to iterate over the entries in $M_0[i,\star]$ and $M_1[\star,j]$ to determine the value of $M_2[i,j]$. As soon as the first k that satisfies $M[i,k] \wedge M[k,j]$ is found, the loop may be short circuited after setting $M[i,j]$ as no further changes to $M[i,j]$ is possible after this action.

When implementing the first of these optimisations, two Boolean vectors are used to store and maintain information respectively about the rows and the columns of the matrices that are to be multiplied. Each value reflects whether there are set bits in the row or column it represents. The values of the entries in

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

these matrices are determined using a function called **any**. This function takes a $\mathbb{B}[n]$ parameter and determines whether any bits in the array that is passed by the parameter are set. The parameter value represents a row or a column in the adjacency matrix. If the value returned by this function is 0, the row is empty. A conditional statement can be used to perform the optimisation in cases where a row or column is empty.

When implementing the second of these optimisations, no extra information has to be stored. The value of $M[i,j]$ can simply be inspected after it was assigned the value of $M[i,k] \wedge M[k,j]$ for each k. The loop that iterates over k may be short circuited if $M[i,j]$ is true.

18.2.2 Opportunities in grow algorithms

The inner loop of an algorithm that is derived form Algorithm 12.5.1 (MatrixGrow) performs the operation that can be expressed as $M[i,\star] = M[i,\star] + M[j,\star]$ for every j for which $M[i,j]$ holds. The following conditions describe situations where information about the rows of M provide reasons to skip calculations in the execution of these algorithms:

- If $M[i,\star]$ is full, i.e. all entries in row i are set, performing this operation will not change the value of any of the entries in $M[i,\star]$, regardless of the value of $M[j,\star]$. The operation can thus be skipped.

- If $M[j,\star]$ is empty, i.e. all entries in row j are not set, performing this operation will not change the value of any of the entries in $M[i,\star]$, regardless of the value of $M[i,\star]$. The operation can thus be skipped.

- If $M[j,\star]$ is full, i.e. all entries in row j are set, performing this operation will render $M[i,\star]$ to be full. Instead of performing the addition, the operation may be replaced by a faster operation that simply sets all entries in $M[i,\star]$. In algorithms that perform this operation in a loop that iterates over the elements of $M[i,\star]$, the loop can be short circuited after the event of filling $M[i,\star]$ as no further changes to $M[i,\star]$ is possible after this action.

When implementing these optimisations in grow algorithms a vector is used to store and maintain information about the rows in the adjacency matrix. The entry for a given row reflects the number of set bits in that row. The values of these entries are determined using a function called **count**. This function takes a $\mathbb{B}[n]$ parameter and determines the number of bits in its parameter that are set. The parameter value represents a row in the adjacency matrix. If the value returned by this function is 0, the row is empty. If the value returned by this function is equal to the dimension of the adjacency matrix, the row is full. The fact that a given row is empty or full can be used in conditional statements to perform the optimisations.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

18.3 The short circuit version of Prosser's algorithm

The implementation of Prosser's algorithm calculates $M_1 \times M_0$. The short circuit opportunities discussed in Section 18.2.1 can be applied when performing the algorithm to calculate $M_1 \times M_0$. The implementation that takes advantage of this optimisation presented here, is an optimised version of Algorithm 12.4.1 (Prosser) that, to my knowledge, has not yet been published. Algorithm 18.3.1 (ShortProsser) is the specification of this algorithm. Figure 18.3.1 shows the topics related to this algorithm in the TM of TCA. Listing C18 is the TM specification of this algorithm in the TM of TCA and The TM specification pointing to implementations of Algorithm 18.3.1 (ShortProsser) in the TM of TCA can be found in Listing C43.

Algorithm 18.3.1: Short circuit version of Prosser's Algorithm

---

const $R \subseteq U \times U$
  $n = |U| \in \mathbb{N}^+$
  $M_0 \in \mathbb{B}[n,n]$
var $M_1 \in \mathbb{B}[n,n]$
  $M_2 \in \mathbb{B}[n,n]$
  $W_0 \in \mathbb{B}[n]$
  $W_1 \in \mathbb{B}[n]$

$M_0, M_1, M_2 := \Phi(R), \Phi(R), \Phi(R)$
for $j : j \in \mathbb{N}_{n-1} \quad \rightarrow$
  $W_1[j] :=$ **any**$(M_0[\star, j])$
rof
for $h : h \in \mathbb{N}_{n-2} \quad \rightarrow$
  for $i : i \in \mathbb{N}_{n-1} \quad \rightarrow$
    $W_0[i] :=$ **any**$(M_1[i, \star])$;
    for $j : j \in \mathbb{N}_n \quad \rightarrow$
      if
        $\neg W_0[i] \vee \neg W_1[j] \quad \rightarrow$ **skip**
      $[\!]$ $W_0[i] \wedge W_1[j] \quad \rightarrow$
        $M_1[i,j] := \langle\, \exists k \mid k \in \mathbb{N}_n \mid M_1[i,k] \wedge M_0[k,j] \,\rangle$;
      fi
    rof
  rof
  $M_2 := M_2 + M_1$
rof
$\{$**Post:** $\Psi(M_2) = R^+\}$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

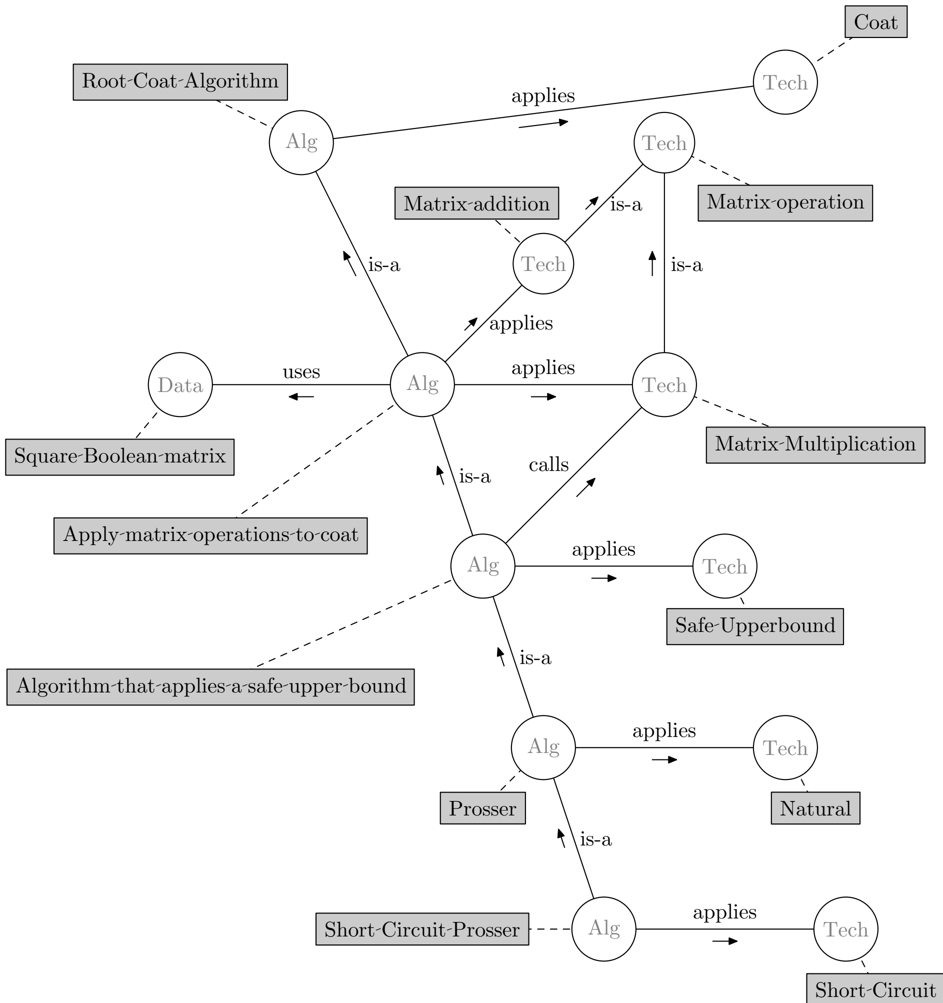


Figure 18.3.1: Topics related to the short circuit version of Prosser's algorithm

18.3.1 Verification

Since it was shown in Section 12.4.3 that Algorithm 12.4.1 (Prosser) is correct, all that is needed here is to give an argument that the short circuiting that is applied here does not change the outcome. The detail of these techniques and the arguments for their validity are discussed in Section 18.2.1.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

18.3.2 Complexity

The algorithm starts with a loop to initialise the one-dimensional Boolean matrix that is used to store and maintain information about the columns of M_0 that is to be multiplied with M_1. This operation is $\Theta(n^2)$ because there are n elements in each of the n columns of M_0 that need to be considered.

The outer loop of the algorithm contains two parts. The first part calculates $M_1 \times M_0$ while the second part calculates $M_2 + M_1$.

The complexity of the first part is $O(n^3)$. It is a three-level loop. The body of the outer loop starts by recalculating the relevant entry in the one-dimensional Boolean matrix that is used to store and maintain information about the rows of M_1 that is multiplied with M_0 in this step. The complexity of this operation is $\Theta(n)$ because there are n elements in the row that are considered. This is followed by a loop that is executed n times and contains the operation to calculate $\langle\ \exists k \mid k \in \mathbb{N}_n \mid M_1[i,k] \wedge M_0[k,j]\ \rangle$. The complexity of this calculation is $\Theta(n)$ as n different values of k need to be considered. The complexity of this operation is $\Theta(n^2)$. Owing to the application of the short circuit technique to decide of this operation needs to be performed, it is possible that the operation is sometimes performed in $\Theta(1)$ time, we conclude that the complexity of this step is $O(n^2)$. Thus the complexity of the body of the outer loop is $\Theta(n) + O(n^2) = O(n^2)$. This operation is repeated n times resulting in its complexity being $n \times O(n^2) = O(n^3)$

The complexity of the second part is $\Theta(n^2)$. This is because there are $n \times n$ values to calculate that is each calculated in constant time.

The complexity of the body of this loop is $O(n^3) + \Theta(n^2) = O(n^3)$. The loop is repeated $n - 2$. Thus the complexity of executing this loop is $O((n-2) \times n^3) = O(n^4 - 2 \times n^3) = O(n^4)$. Listing C18 shows the definition of the complexity of Algorithm 18.3.1 (ShortProsser) in the TM of TCA.

The additional operations to apply short circuiting are likely to impact negatively on the performance of the algorithm. In Section 12.4.4 it was established that the complexity of this algorithm when not applying short circuiting is $\Theta(n^4)$. When comparing this with the complexity of the algorithm established here, it can be seen that the overall complexity is not affected by inserting the additional operations. The gain of short circuiting may outweigh the loss imposed by the extra operations. The thresholds in terms of size and density of the input relation at which stage the application of short circuiting is beneficial need to be established through experimentation.

The steps in the derivation tree of Algorithm 18.3.1 (ShortProsser) are the application of the fundamental coat technique; transformation to use a matrix for data representation and to use the corresponding matrix operations to manipulate the data; application of a safe upper bound and finally the implementation of short circuiting where applicable.

18.4 Other short circuited coat algorithms

A short circuited algorithm is derived from each of the coat algorithms, similar to how Algorithm 18.3.1 (ShortProsser) is derived from Algorithm 12.4.1 (Prosser).

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

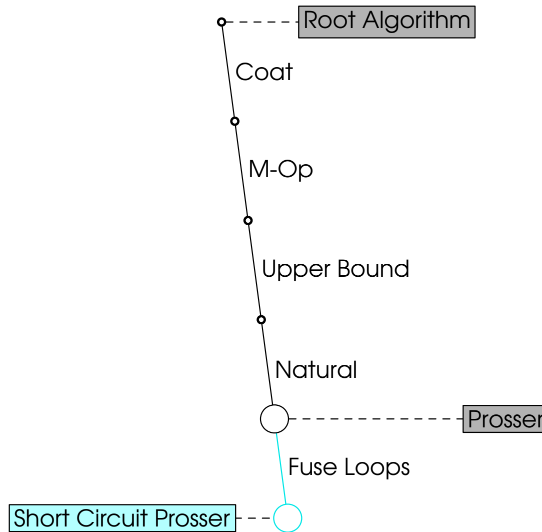


Figure 18.3.2: Derivation tree of the short circuit version of Prosser's algorithm

The detail discussion of these algorithms is not included in this thesis because of their similarity to their respective parents and the correspondence of their detail with that of Algorithm 18.3.1 (ShortProsser).

Table 18.4.1 lists the short circuit algorithms that are derived from Algorithm 12.2.2 (MatrixCoat) for which implementations can be found on `http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html`

Table 18.4.1: Short circuit coat algorithms

Parent	**The TM specification pointing to implementations of the short circuit version of the algorithm**
Algorithm 12.4.1 (Prosser)	Listing C43
Algorithm 14.2.1 (CoatMonitor)	Listing C51
Algorithm 15.1.1 (CoatFuse)	Listing C53
Algorithm 15.2.1 (CoatNeat)	Listing C55

The correctness of these algorithms follows from the fact that each operation that is performed in a short circuited algorithm that is not performed in its parent, is a conditional statement to bypass operations that will have no effect given the condition. The different conditions as well as the reason why each of these conditions cause the operations that are bypassed to have no effect is discussed in Section 18.2.1.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The discussion of the complexity of Algorithm 18.3.1 (ShortProsser), can *mutatis mutandis* be applied to verify that the application of the short circuit technique to derive an algorithm, creates a new algorithm with the same theoretical complexity as its parent in the derivation hierarchy.

As can be seen in Figure 18.5.1 each short circuit algorithm applies all the algorithmic techniques applied by its parent and can be derived from its parent algorithm by applying the short circuit technique. Each short circuit algorithm can thus be positioned as a new leaf node in the derivation tree on the branch extending from its parent. Figure 18.5.1 shows this.

18.5 Derivation tree of coat algorithms

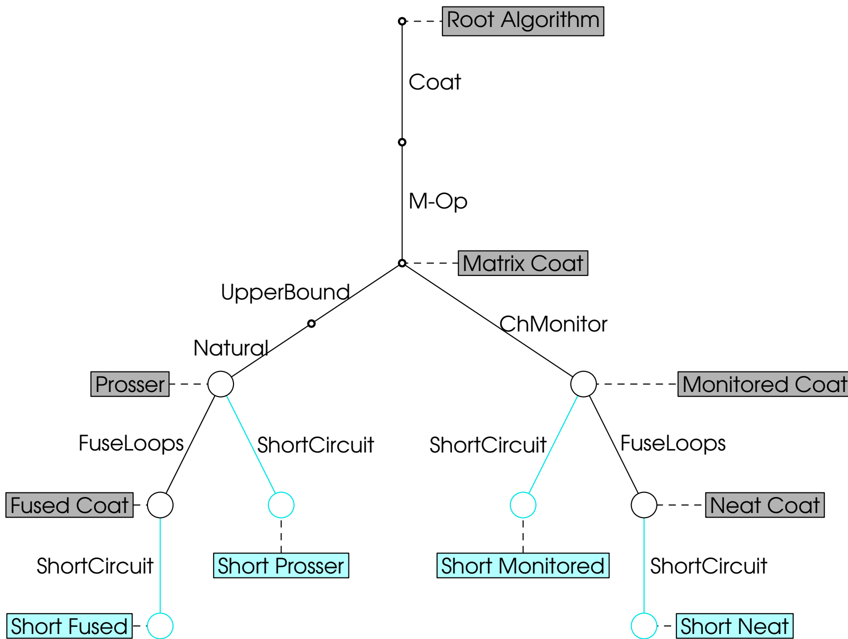


Figure 18.5.1: Derivation tree of the coat algorithms

The coat algorithms are algorithms that apply the basic coat technique to calculate the TC of a given relation discussed in Section 11.3. The matrix operation that corresponds with the coat technique is matrix multiplication. All the algorithms in this portion of the derivation tree of TC algorithms are variants of Algorithm 12.2.2 (MatrixCoat) applying the standard algorithmic techniques individually or combined. To my knowledge, I am the first author to apply these techniques to Prosser's algorithm. In a forthcoming publication the impact of the application of these techniques are measured. The application of a change monitor shows impressive improvements in some cases while the other techniques have

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

minimal effect. Short circuiting has almost no impact except for the extreme case of zero-density.

18.6 The short circuit Warshall algorithm

The implementation of Algorithm 12.6.1 (Warshall) applies the operation discussed in Section 18.2.2. These short circuit opportunities can be applied when performing this operation. The implementation that takes advantage of this optimisation is a version of Algorithm 12.6.1 (Warshall) that, to my knowledge, has not yet been published.

Figure 18.6.1 shows the topics related to this algorithm in the TM of TCA and Listing C20 is the TM specification of this algorithm in the TM of TCA. The TM specification pointing to implementations of the short circuit Warshall algorithm in the TM of TCA can be found in Listing C45.

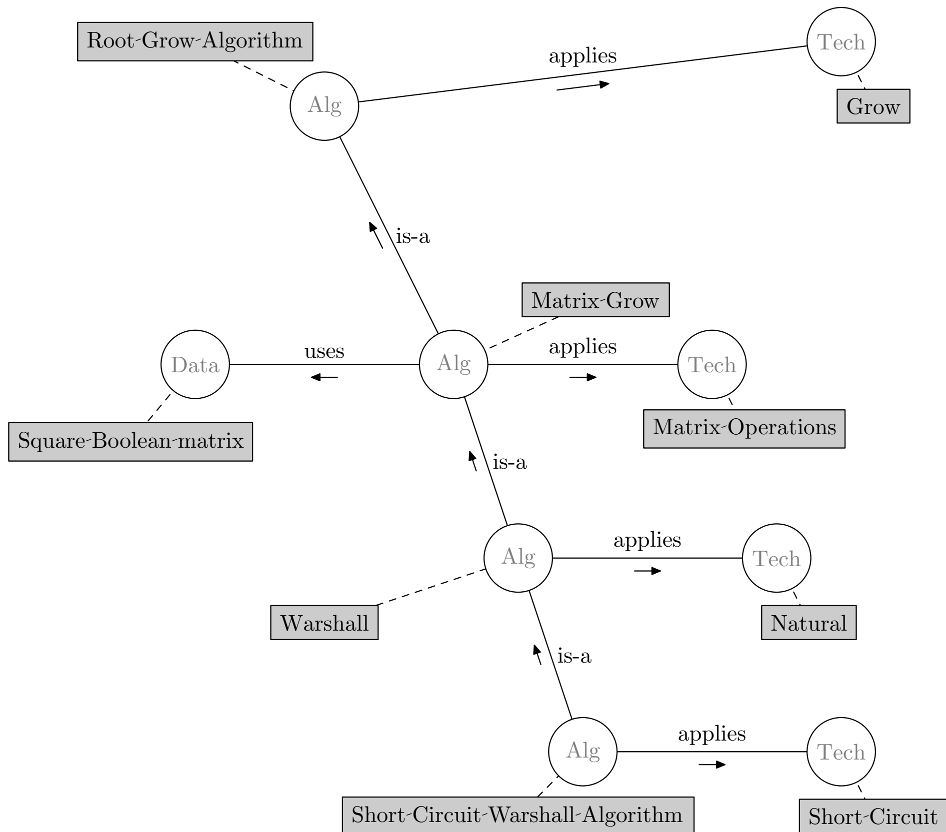


Figure 18.6.1: Topics related to the Short Circuit Warshall algorithm

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Algorithm 18.6.1: Short Circuit version of Warshall's Algorithm

```
const   R ⊆ U × U
        n = |U| ∈ ℕ⁺
var     M ∈ 𝔹[n, n]
        S ∈ ℕ[n]

M = Φ(R)
for i ∈ ℕₙ  →  sᵢ := count(M[i, ⋆]) rof
for j : j ∈ ℕₙ  →
    if sⱼ = 0  →  skip
    [] sⱼ ≠ 0  →
      for i : i ∈ ℕₙ  →
        if sᵢ = n  →  skip
        [] sᵢ ≠ n  →
          if ¬M[i, j] → skip
          [] M[i, j]  →
            if sⱼ = n  →  sᵢ := n;  for k ∈ ℕₙ  →  M[i, k] := true rof;
            [] sⱼ ≠ n  →
              for k ∈ ℕₙ  →  M[i, k] := M[i, k] ∨ M[j, k] rof;
              sᵢ := count(M[i, ⋆])
            fi
          fi
        fi
      rof
    fi
rof
{Post: Ψ(M) = R⁺}
```

The algorithm stores and maintains information about the rows in the adjacency matrix in an integer vector. Each entry in this vector is the number of set bits in a row in the adjacency matrix. This vector is initiated using the **count** function. Inspection of these values are applied to determine if a given row is empty or full.

The outer loop of the algorithm iterates over the columns of the adjacency matrix. When column j is processed, row $M[j,\star]$ is added to each row $M[i,\star]$ for which $M[i,j]$ holds. In the implementation of Algorithm 18.6.1 (ShortWarshall), the body of the outer loop is skipped when row $M[j,\star]$ is empty. It is known that the addition of the empty row $M[j,\star]$ will not change the state of M. Furthermore, it is unnecessary to verify if there are any rows that need to be added during this iteration as it is impossible that there is such a row. A consequence of the fact that $\langle\ \forall i \mid i \in \mathbb{N}_n \mid \neg M[i,j]\ \rangle$ holds is that there will be no situation that would require $M[j,\star]$ to be added to another row of M. This short circuit step thus not only omits futile addition operations but also skips the verification whether this

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

operation is needed or not that is performed by the middle loop.

The middle loop of the algorithm processes the elements of column j. At the point where a decision is made whether or not to add the jth row of M to the ith row of M, the body of the middle loop can be skipped if it is known that the ith row of M is full. In this case $\langle \forall j \mid j \in \mathbb{N}_n \mid M[i,j] \rangle$ already holds. Since performing the body of the middle only potentially changes the entries in row $M[i,\star]$, the loop will thus not change any entries in M.

If the inner loop is reached, and it is known that $M[j,\star]$ is full, the calculation of $M[i,\star]$ that is performed using the iterative operation $M[i,\star] = M[i,\star] + M[j,\star]$ is replaced by a faster operation that simply sets all entries in $M[i,\star]$ and updates the status of the ith row of M by setting it to the size of the adjacency matrix. This short circuit step may not have a big impact as the complexity of the operation stays the same. It may, however, prove beneficial when M is represented using a data structure like `boost :: dyn_bitset` that has a fast atomic operation to set all bits in the variable representing a row in M.

The inner loop of the algorithm is reached after none of the above short circuit operations prevented it, it is guaranteed that the addition operation will change some entries in M. At this point the operation is performed. After updating the entries in $M[i,\star]$ using the usual manner, the status of the ith row of M is updated using the **count** function. The use of the **count** function adds some processing. It is, however, compensated for by the fact that the value calculated here allows the elimination of the inner loop in further iteration of the outer loops. When an optimised data structure is used, this operation may be fast. Furthermore, when an unoptimised data structure is used, the loop performing this operation may be fused with the body of the inner loop of the algorithm.

18.6.1 Verification

Each operation that is performed in this algorithm that is not performed in Algorithm 12.6.1 (Warshall), is a conditional statement to bypass operations that will have no effect given the condition. The different conditions as well as the reason why each of these conditions cause the operations that are bypassed to have no effect is discussed in Section 18.2.2.

Since only operations that have no effect are bypassed, the end result of this algorithm is the same as that of Algorithm 12.6.1 (Warshall) which was proven to be correct in Section 12.6.4.

18.6.2 Complexity

The algorithm starts with a loop to initialise the row monitor. The complexity of this operation, if carried out naively, is $\Theta(n^2)$ as each element in each row in the adjacency matrix is counted.

The complexity of the rest of the algorithm is the same as that of Algorithm 12.6.1 (Warshall). The algorithm body consists of three levels of nested loops. Although some of the iterations in each of the loops may be skipped, each

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

of these loops requires at most n iterations. As opposed to the situation in Algorithm 12.6.1 (Warshall) that has only one loop of complexity $\Theta(n)$ inside the middle loop, Algorithm 18.6.1 (ShortWarshall) has two adjacent loops of which one is the same as the one in Algorithm 12.6.1 (Warshall) and the other is a loop to update the status of the newly changed row by using the **count** function. This is an $O(n)$ operation because there are n elements in the row and it will take maximally n additions to count them. Since these two inner loops are adjacent, the complexity of the middle loop is $\Theta(n) + O(n) = O(n)$. More optimal implementations are possible, for example these two inner loops can be fused. This will, however, not change the complexity. The resulting complexity of the nested loop structure is $O(n^3)$.

The complexity of the entire algorithm is thus $\Theta(n^2) + O(n^3) = O(n^3)$ Listing C20 defines this complexity in the TM of TCA.

When comparing the complexity of this algorithm with the complexity as Warshall's algorithm discussed in Section 12.6 it can be observed that the overall complexity of the short circuited version of the algorithm is not worse than the complexity of the non-short circuited version of the algorithm. When the short circuited version of the algorithm is compared with Warshall's algorithm, it can be seen that it uses substantial additional processing time to perform the checks to determine if operations can be skipped. It also requires additional memory and processing to keep track of the status of the rows in the adjacency matrix. This overhead is only warranted if the processing time gained by skipping operations exceeds the overhead. The thresholds with respect to the size and density of relations at which this short circuit technique is likely to be beneficial need to explored through measuring the performance of implementations of this algorithm.

As can be seen in Figure 12.6.3, Warshall's algorithm appears as a leaf node in the derivation tree. Furthermore, the Short Circuit Warshall algorithm applies all the algorithmic techniques applied by Warshall's algorithm and can be derived from Warshall's algorithm by applying the short circuit technique. Algorithm 18.6.1 (ShortWarshall) can thus be positioned in the derivation tree on the branch extending from Warshall's algorithm. Figure 18.6.2 shows this.

18.7 Other short circuit grow algorithms

A short circuited algorithm is derived from each of the grow algorithms, similar to how Algorithm 18.6.1 (ShortWarshall) is derived from Algorithm 12.6.1 (Warshall). The detail discussion of these algorithms is not included in this thesis because of their similarity to their respective parents and the correspondence of their detail with that of Algorithm 18.6.1 (ShortWarshall).

Table 18.7.1 lists the short circuit algorithms that are derived from Algorithm 12.5.1 (MatrixGrow), for which implementations can be found on `http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html`

The correctness of these algorithms follows from the fact that each operation that is performed in a short circuited algorithm which is not performed in its parent, is a conditional statement to bypass operations that will have no effect given

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

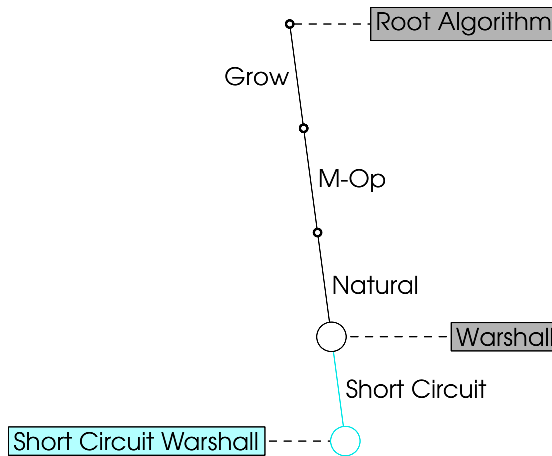


Figure 18.6.2: Derivation tree of the short circuited Warshall algorithm

Table 18.7.1: Short circuit grow algorithms

Parent	**The TM specification pointing to implementations of the short circuit version of the algorithm**
Algorithm 12.6.1 (Warshall)	Listing C45
Algorithm 13.2.1 (Martynyuk)	Listing C47
Algorithm 14.1.1 (Baker)	Listing C49
Algorithm 17.1.1 (Warren)	Listing C57
Algorithm 17.3.1 (BlockRow)	Listing C59
Algorithm 17.5.3 (BlockCol)	Listing C61

the condition. The different conditions as well as the reason why each of these conditions cause the operations to have no effect are discussed in Section 18.2.2.

The discussion of the complexity of Algorithm 18.6.1 (ShortWarshall), can *mutatis mutandis* be applied to verify that the application of the short circuit technique to derive an algorithm, creates a new algorithm with the same theoretical complexity as its parent in the derivation hierarchy.

Each short circuit algorithm applies all the algorithmic techniques applied by its parent and is derived from its parent by applying the short circuit technique. Each short circuit algorithm can thus be positioned as a new leaf node in the derivation tree on the branch extending from its parent. Figure 18.8.1 shows this.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

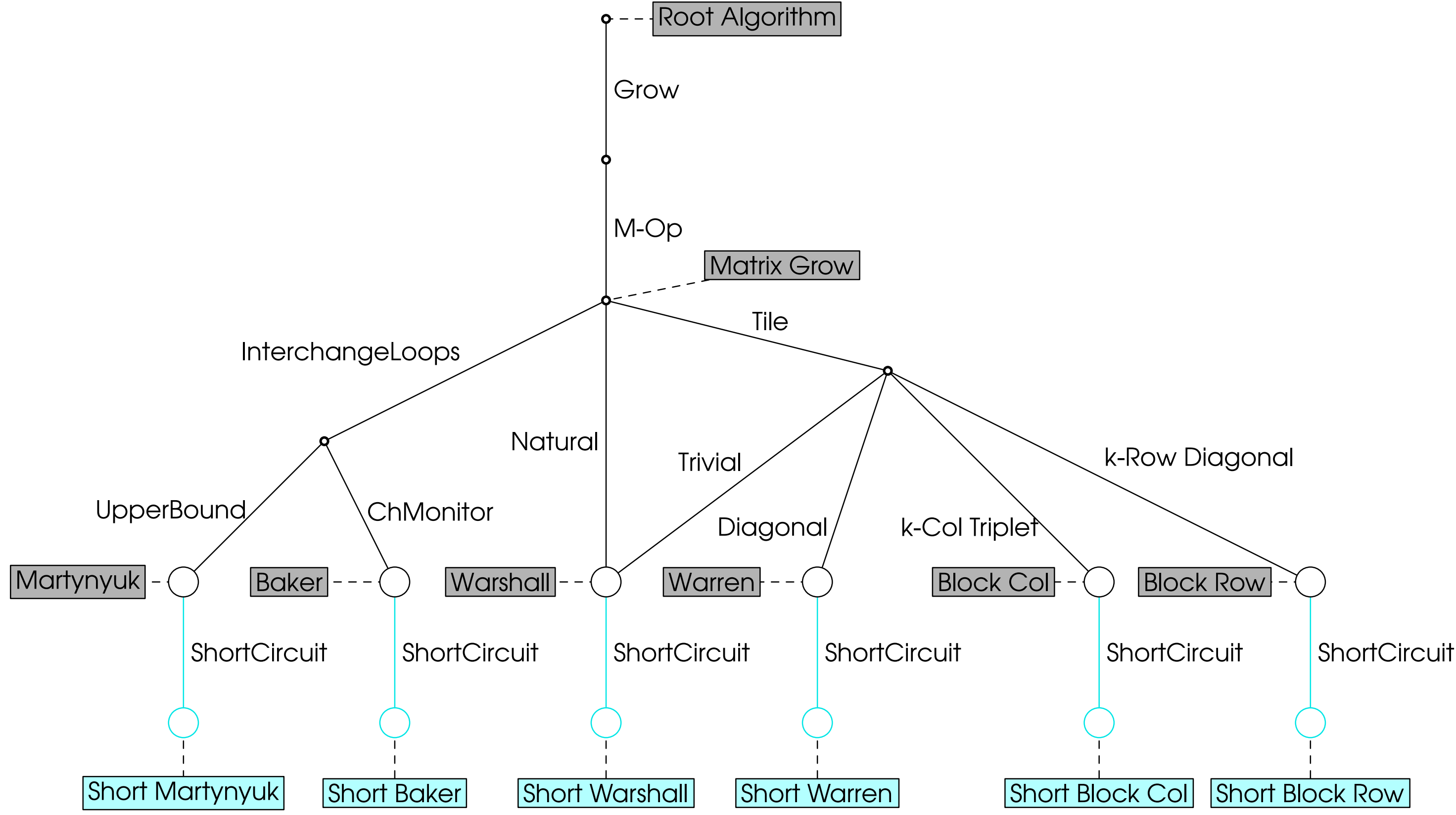


Figure 18.8.1: Derivation tree of the grow algorithms

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

18.8 Derivation tree of grow algorithms

The grow algorithms are algorithms that apply the basic grow technique to calculate the TC of a given relation discussed in Section 11.4. All the algorithms in this portion of the derivation tree of TC algorithms are derived from Algorithm 12.5.1 (MatrixGrow). These algorithms apply alternative techniques to Algorithm 12.5.1 (MatrixGrow) individually or in combination. The six algorithms shown with grey labels in the middle level of the diagram in Figure 18.8.1 were published from left to right

- In 1962 by Martynyuk [162].
- In 1962 by Baker [17].
- In 1962 by Warshall [252].
- In 1975 by Warren [251].
- In 1987 by Agrawal and Jagadish [2].
- In 1987 by Agrawal and Jagadish [2].

It is hard to believe that more than ten years have elapsed after the first variations of the algorithm were published in 1962, before Warren's [251] algorithm, that applies a simple tiling technique, was published. A similar time elapsed during which no new variations of these algorithms were published, before Agrawal and Jagadish [2] published their improvements that apply more complex tiling techniques. In this thesis, to my knowledge, the improvement to apply short circuiting to all these variations is published for the first time. This seemingly obvious improvement is offered here almost 20 years after the previous improvement was proposed.

Unlike the case in the coat algorithms where short circuiting seems not to make a big difference, a forthcoming publication reports impressive performance gains when applying the short circuiting techniques discussed in Section 18.2.2 in the grow algorithms reported in this chapter.

18.9 Summary

This chapter discuss short circuiting as a general algorithmic technique. The opportunities for short circuiting the operations in both coat and grow algorithms are described in Section 18.2. These opportunities were observed while I traced the operations of the various algorithms manually during the time I was writing this thesis. I performed these traces to gain deeper understanding of the algorithms.

The application of these techniques to all the algorithms are discussed by means of two examples. Section 18.3 discuss the short circuited version of Algorithm 12.4.1 (Prosser) as a representative example of how short circuiting can be applied in coat algorithms while Section 18.6 offers the short circuited version of Algorithm 12.6.1 (Warshall) as an example of this technique for grow algorithms. In both cases the artefacts needed for including these algorithms in the TM of TCA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

are created. These include correctness reasoning, discussions of the complexity of the algorithms as well as GCL specifications and C++ implementations of the algorithms. To my knowledge these have not been described or implemented before.

Because this technique is applied to all the algorithms that were discussed in Part III of this thesis, the opportunity to discuss the two branches of algorithms developed in this thesis was used in Sections 18.5 and 18.8. In each of these sections the derivation tree showing the relation between all the algorithms in the particular branch is shown and briefly discussed.

The next part of this thesis contains a single chapter which highlights the main contributions made in this thesis.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Part IV

Epilogue

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Epilogue Overview

The Epilogue consists of the conclusion chapter, Chapter 19. This chapter summarises the highlights and key contributions offered in this thesis. It contains a concise representation of the lessons learnt while writing the thesis. It proposes an agenda of future work and concludes with my vision of the role of this work in a forthcoming technology-rich world.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Chapter 19

Conclusion

This thesis describes artefacts that have been created in the design science paradigm. This paradigm is briefly explained in Section 1.1. A novel algorithm data model shows that it is possible to design a data model for sharing and exchange of information related to algorithms. The analysis in this thesis confirms the need for standardisation of domain-specific algorithmic information in order to enhance the accessibility and usability of this knowledge. There is no claim that the features outlined here, or their organisation, are the only way to present algorithmic information, or that they are the best possible way to do so. These ideas are meant to suggest a possible line of attack; the major claims lie in the approach, and the minor claims in the set of features chosen at each level.

While creating the artefacts and writing this thesis I made contributions to the advancement of knowledge and science. Instead of summarising the thesis, I decided to highlight only a few specific contributions in Section 19.1 that I deem to be of some significance. Section 19.2 is a self-critique of my work using the criteria postulated for the evaluation of design science research. The final two sections list a few immediate actions to extend this work and how I believe that my work might enable future advancement.

19.1 Novel contributions

The work reported in this thesis is mainly within the field of algorithmics. The main focus is to address the problem that different algorithms solving a specific problem, though published, are often hard to find and difficult to compare. While conducting the research and creating the artefacts resulting from the research, a number of novel contributions to the fields of mathematical science, formal aspects of computing, information science and software engineering emerged. The contributions to each of these areas are briefly highlighted.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

19.1.1 Algorithmics

The foremost aim of this work is to bring order to the algorithmic domain. The current state of the art of categorisation and classification of algorithms is still in its infancy. The work presented in this thesis paves a way towards the use of a standardised vocabulary for the description of algorithms and their attributes and proposes a process to support the development of a standardised taxonomy of algorithms.

The use of the proposed vocabulary and the application of the process are illustrated by means of an example. A prototype categorisation of algorithms to calculate the transitive closure of a relation serves this purpose. This prototype is presented as a topic map and is called the topic map of transitive closure algorithms (TM of TCA). Although the TM of TCA covers a relatively small subset of the algorithms in this domain, the case study is deemed sufficient to illustrate the viability of the concepts and processes proposed in this thesis. The prototype application has turned out to be surprisingly effective to create new knowledge about the algorithms used in the example.

The analysis of these algorithms instigated the naming and definition of the two fundamental algorithmic techniques, namely coat and grow, used by the algorithms in the TM of TCA. These are general techniques that are not specific to the TC problem; they are likely to be used by many algorithms solving other problems. The mathematical concepts of the coat and grow techniques are not new, but to the best of my knowledge, they have not been named or described as algorithmic techniques before.

The common use of well-known general techniques such as loop fusion, loop interchange, loop tiling and short circuiting as powerful optimisation steps are confirmed in this thesis. It was surprising to discover that short circuiting has not been applied before in the implementation of a number of well-known algorithms including the renowned Algorithm 12.6.1 (Warshall). The usefulness of change monitors has been rediscovered. The application of some of these techniques to algorithms where their use is not obvious has lead to the discovery of new algorithms such as Algorithm 14.2.1 (CoatMonitor) and Algorithm 15.2.1 (CoatNeat).

The routine work applied during the creation of the TM of TCA has led to the identification of the concept of a tiling algorithm that serves as an abstraction of a number of algorithms. These algorithms have not been recognised as (degenerate) tiling algorithms before. Notable algorithms that fit this description are Algorithm 12.6.1 (Warshall) and Algorithm 17.1.1 (Warren).

19.1.2 Formal aspects of computing

A large portion of the descriptions of the algorithms in Part III of this thesis is devoted to the mathematical confirmation of the correctness of these algorithms as well as to mathematical reasoning regarding the complexity of these algorithms. Most of these descriptions are original. In the cases where the work is based on previously published proofs or reasoning, the existing work is transformed to

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

adhere to the format and rigour applied in this thesis. In many cases this required substantial novel enhancement.

I am particularly proud of the mathematical derivation of the base algorithms to calculate the transitive closure of a binary relation presented in Sections 6.5.1 and 6.5.2. Both algorithms are derived by applying the Knaster-Tarski fixpoint theorem to a continuous function that was custom-defined to yield the algorithm. To my knowledge, this thesis is the first publication of these elegant derivations.

Another highlight of this kind worth mentioning is the proof of the correctness of Algorithm 16.4.1 (TileSkeleton) in Section 16.4.1. Agrawal and Jagadish [2] have derived constraints that should hold for the processing order of the entries in the adjacency matrix when executing Algorithm 12.5.1 (MatrixGrow), which I suspected could be applied to prove the correctness of this algorithm. The technical memorandum that was published in 1987 by Agrawal an Jagadish which contains the derivation of these constraints, cited in Agrawal and Jagadish's [2] article, could however not be located. I accepted the challenge of independently reconstructing their work and succeeded in producing a correctness argument that assumes only two of the three constraints they proposed. My work is thus an improvement on theirs. It was not a trivial task and is to my knowledge a unique contribution presented in this thesis.

19.1.3 Mathematical science

A contribution in the domain of mathematical science involves a rigorous proof of the equivalence of the representation of a relation as a set of pairs and its representation in the form of an adjacency matrix. The fact that these have been proven to be isomorphic enabled me to apply elegant mathematical correctness reasoning in one view and then apply the isomorphism to prove that the result holds in the other view.

A second contribution is the invention of notations. Mathematical notations are introduced to provide concise and accurate ways to communicate complex yet well understood concepts. Brevity is the leading characteristic of mathematical elegance, but not the only requirement. Symbolic notation may be introduced merely to save space [255], yet most mathematicians agree on the value of clever notations. Dijkstra and van Gasteren [80] emphasise that the use of apt notation can make a difference in mathematical work.

Following the example set by authors such as Backhouse and Ferreira [16], Cleophas [60], van Gasteren and Dijkstra [245], Venter *et al.* [247] and Watson [253], who have worked on the creation of taxonomies of algorithms, I adopted a notation similar to the one used by Dijkstra [78] for sets as well as quantifications of commutative operations. Instead of using the notation in the same way as my predecessors, I spent time investigating its origins and compared the different nuances of the notation they used. I also compared their notations with the notations for these constructs as they appear in the international standard ISO 80000-2:2009 [126]. Finally I designed a new dialect of the Dijkstra notation that retains the gist of Dijkstra's original proposal and adheres to the ISO standard. I

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

also introduced a generalisation of the Dijkstra notation to enable the specification of sequences whereas the original notation was limited to the specification of sets.

Another mathematical contribution made in this thesis is the definition of a new mathematical concept. The concept I defined for the first time is that of a partition of a series. As in the case of the notational development, this concept is an enrichment of the same concept in relation to sets. The classification of tiling algorithms necessitated the specification of a subdivision of a matrix that complies with restrictions similar to the concept of the partition of a set. To my knowledge the definition I introduced to serve my purpose is consistent with the definition of a partition of set as defined by Erdös and Rado [86] and has not been used before.

19.1.4 Information science

When I started this work, the original intention was to compile a taxonomy of transitive closure algorithms similar to the taxonomies of Cleophas [60] and Watson [254]. My primary supervisor suggested that, instead of creating a hierarchical taxonomy, I might consider presenting a lattice. This seemed a promising option as illustrated by Cleophas *et al.* [58]. In order to adhere to this advice I deemed it appropriate to include a short section in my thesis to explain the different possible structures and justify my choice.

My search for definitions revealed that classification-related terms such as taxonomy, thesaurus, lattice and ontology are often used inconsistently both in and across different research fields and have no definitive definitions. I realised that this terminological conundrum has led to misconceptions and has impeded communication among researchers. A common nomenclature is needed to incorporate the vast body of semantic information embedded in existing classifications when developing new systems. Interoperability among diverse systems has to be facilitated. I therefore provide my own definitions of the above-mentioned terms as well as other terms such as catalogue, index, lexicon, knowledge base and topic map. These appear in an article co-authored by my primary supervisor that has been published in *Knowledge Organization* [199].

Having gained a deeper understanding of the various data structures used for knowledge representation and the tools that have been developed to support the use of these structures, for the reasons pointed out in Section 7.4.2, I am convinced that a topic map is the most appropriate structure to capture information about algorithms.

19.1.5 Software engineering

The work done for this thesis that relates to software engineering is rooted in the TAxonomy-BAsed Software COnstriction (TABASCO) process discussed in Section 1.2.

The topic map presented in Chapter 9 provides a framework that has the potential to support developers when they assemble software using the most appropriate algorithm for a given situation without the need to be an expert in algorithmics. It is a platform where the existing taxonomies that have been created

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

so far can be integrated into a coherent ontology of algorithmic information. The use of this framework is likely to contribute to the increased practical usability of provably correct algorithms.

Implementations of all algorithms included in the TM of TCA constructed in Part III are made available in the public domain on my website at `http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html`. This site is mentioned in Wikipedia [264].

One of the steps in the TABASCO process is to measure the performance of the algorithms. I have thoroughly familiarised myself with the state of practice in experimental evaluation in computer science and have published guidelines on how to measure software performance as an official technical document for the American National Institute of Standards and Technology (NIST) [198]. I have also performed performance measurements of the algorithms in the TM of TCA. An article to publish the results is forthcoming.

19.2 Self-reflection

The design science paradigm seeks problem solving, creation and innovation [102]. The research reported in this thesis complies with these criteria.

19.2.1 Creation to solve a problem

The problems that are addressed here are the inaccessibility of algorithms as well as the difficulty that researchers and practitioners experience when having to compare algorithms. The artefacts presented in this thesis were created specifically to serve as a solution to these problems.

The tools that were used in this creation process had to be customised. Apart from the extension of the mathematical notation and concepts discussed in Section 19.1.3, the standard notations of GCL and LTM were adapted to address shortcomings. The particular adaptations are discussed in Sections 2.2.6, 2.2.7, 2.3.2, 2.3.1 and 2.3.9.

19.2.2 Innovation

On the surface it may seem as if this thesis lacks innovation. The artefacts presented in the thesis are not the first of their kind and long-established technologies were applied to create them. This is explained in Chapter 7. The overview of work that has been done with respect to gathering and capturing information about algorithms includes the work of Jonkers [130] in 1982. The technology applied in this work, namely topic maps, was conceptualised by Biezunski and Newcomb [29] and established as the ISO/IEC 13250 standard [124] in 1999. The discussion about techniques and technologies for gathering and representing information that informed the technologies in this domain dates back to the work of Linnaeus [259] published in 1735, the work of Roget [209] in 1952 and Ranganathan [205] in 1962.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

The innovation lies in the application of topic maps in a domain where it has not been applied before and in doing so, taking the creation, capturing and representation of algorithmic knowledge to a new level. Hevner *et al.* [113] calls the adoption of solutions from another domain to solve a known problem *exaptation*, i.e. the expropriation of artefacts in one field to solve problems in another field. In exaptation research, the researcher needs to demonstrate that the extension of known design knowledge into a new field is nontrivial and interesting [104]. This is indeed the case with this research. Particular challenges that I have had to overcome to apply the topic map technology to the domain of organisation algorithmic information were the following:

- The lack of structure in the domain of algorithmic information.
- The lack of support offered for the representation of commonly found relations between elements in the domain of algorithmic information.

19.2.3 Generalisability

Dresch *et al.* [83] stress that it is important for the developed artefact to be generalised for a class of problems to enable the advancement of knowledge.

The thesis contributes to descriptive knowledge in the form of general knowledge about algorithmic information as well as specific knowledge about transitive closure algorithms. It also contributes to prescriptive knowledge via a greater understanding of the use of topic map technologies and its application to enhance the usability of algorithmic information for various applications such as software construction and education.

19.3 Future agenda

19.3.1 Performance of algorithms

Questions regarding the performance of specific algorithms have to be answered through measuring the performance of different implementations of the algorithms. In particular, thresholds with respect to the size and density of relations being processed where one of the following algorithm variations outperforms the other have to be determined:

- In Section 17.3 the need is identified to determine the threshold where Algorithm 17.3.1 (BlockRow) will outperform Algorthm 17.1.1 (Warren).
- The thresholds where the short-circuit techniques discussed in Chapter 18 are likely to be beneficial need to be explored.
- In Chapter 15 it was established that the fusion of loops in algorithms does not affect the complexity of the algorithms. The expectation that the application of loop fusion may impact positively on the performance needs to be verified.

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

19.3.2 Formal aspects

The proof of the correctness of Algorithm 16.4.1 (TileSkeleton) in Section 16.4.1 assumes two processing order constraints. These constraints are formulated in terms of row-wise specifications. It may be interesting to investigate if the correctness of the algorithm can be established using the dual of these constraints that use column-wise specifications. If this is the case, it opens more possibilities for discovering new algorithms based on column-wise tiling strategies.

19.3.3 Knowledge organisation

The topic map that was created in this thesis has the potential to advance the use of Topic Maps for knowledge organisation towards the creation of a semantic web in which the artefacts are shared by various software agents. It is a small beginning of a big dream of having a comprehensive ontology of algorithms.

The following steps have the potential to move closer to this goal:

- Currently the TM of A (the core topic map) and the TM of TCA are not yet accessible on the Internet. It is likely that it will soon be deployed on a knowledge platform like Kamala[1] that enables organisations and people to link their data and share their knowledge.

- The next step is to find collaborators to participate in the extension of the topic map. The creation of further extensions and the maintenance of a growing knowledge base of this kind will require constant attention. Once the TM of TCA and other extensions of the TM of A have been deployed, they can be applied to find information about the algorithms in them. Developers will also be able to use the extensions of the TM of A to find the information as well as the implementations needed to construct software.

- A large number of TC algorithms still have to be deployed in the TM of TCA, in particular algorithms that use data structures other than adjacency matrices to represent relations. A substantial number of algorithms that apply parallelism using multiple processors have not yet been added to the TM of TCA.

- Cleophas and Watson [56] include a list of existing taxonomies of algorithms that were created as part of the TABASCO effort. It may be beneficial to gradually redeploy them into the future live version of the topic map presented in this thesis.

- I am the co-editor of a mirror version of a dictionary of data structures and algorithms (DADS) that was created in 1996 as a service to the American People. It is hosted on the website of the National Institute of Standards and Technology (NIST). The mirror is hosted on the FASTAR website[2]. The future plan is to redeploy the entire DADS as part of the topic map.

[1] http://kamala-cloud.com/
[2] http://fastar.org/dads/

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

19.3.4 Mathematical notation

Dijkstra [78] enhanced his set notation to specify quantifications of commutative operations. The same technique that was applied to formulate this elegant way to specify quantifications can be applied to my new notation for the specification of sequences. This will lead to the formulation of a new way to specify quantifications involving operations that are not necessarily commutative. To my knowledge there is no generally adopted notation for the specification of sequences or quantifications involving non-commutative operations.

19.3.5 Sociology

I referred in Chapter 1 to the sociological problem that is dubbed the small world problem when I illustrated the concept of a relation and the transitive closure of the relation. Research testing the six degrees of separation hypothesis related to this problem produced contradicting results. On the one hand the average length of connecting paths between people seems to shrink, yet the number of chains that could not be established during the experimentation in the reported research increased drastically between the experiment that was conducted 1968 and the one that was conducted in 2003. The latter fact seems to be ignored while an unconfirmed hypothesis that anyone in the world is connected to any other person in the world through a chain of acquaintances is commonly accepted. It may be interesting to establish if the transitive closure of the "is connected" relation between people is in fact a relation that is always true.

19.4 Final remarks

The topic map of algorithms presented in Chapter 9 serves as a domain thesaurus for the field of algorithmics. It provides the vocabulary for the concepts in this domain and describes the relationships between these concepts. It forms a basis for the development of a comprehensive ontology of algorithms. The topic map of transitive closure algorithms presented in Part III serves as an example of how this topic map can be extended by using the detailed guidelines presented in Chapter 10. This thesaurus is a starting point for the development of an ontology of algorithms to be deployed for the semantic web. The future extensions of this thesaurus are likely to be useful as a teaching and learning aid. Once the semantic web has reached adequate power and interoperability, such an ontology of algorithms will embody knowledge which may enable high levels of automation of the software construction process.

© University of Pretoria

Bibliography

[1] Agrawal R, Dar S and Jagadish HV (1990) Direct transitive closure algorithms: design and performance evaluation, *ACM Transactions on Database Systems (TODS)*, 15(3):427–458. (Cited on pages 7, 338 and 339)

[2] Agrawal R and Jagadish HV (1987) Direct Algorithms for Computing the Transitive Closure of Database Relations, in: *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, 255–266, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., URL `http://0-dl.acm.org.innopac.up.ac.za/citation.cfm?id=645914.671624`. (Cited on pages 7, 159, 216, 217, 219, 220, 232, 238, 256, 273, 281, 338 and 339)

[3] Agrawal R and Jagadish HV (1988) Multiprocessor transitive closure algorithms, in: *DPDS '88: Proceedings of the first international symposium on Databases in parallel and distributed systems*, 56–66, Los Alamitos, CA, USA: IEEE Computer Society Press. (Cited on pages 7, 80, 216, 238 and 256)

[4] Agrawal R and Jagadish HV (1988) Multiprocessor Transitive Closure Algorithms, in: *Proceedings of the First International Symposium on Databases in Parallel and Distributed Systems*, DPDS '88, 56–66, Los Alamitos, CA, USA: IEEE Computer Society Press, URL `http://dl.acm.org/citation.cfm?id=62597.62605`. (Cited on page 8)

[5] Agrawal R and Jagadish HV (1990) Hybrid Transitive Closure Algorithms, in: *VLDB '90: Proceedings of the 16th International Conference on Very Large Data Bases*, 326–334, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (Cited on pages 7 and 8)

[6] Aho AV, Garey MR and Ullman JD (1972) The Transitive Reduction of a Directed Graph, *SIAM Journal on Computing*, 1(2):131–137, URL `http://dx.doi.org/10.1137/0201008`. (Cited on page 7)

[7] Aitchison J, Gilchrist A and Bawden D (1997) *Thesaurus construction and use : a practical manual*, London: Aslib, 3rd edn. (Cited on page 101)

[8] Alturki A, Gable GG and Bandara W (2011) A Design Science Research Roadmap, in: Jain H, Sinha AP and Vitharana P (Eds.) *Service-Oriented Perspectives in Design Science Research*, vol. 6629 of *Lecture Notes in Computer*

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Science, 107–123, Springer Berlin Heidelberg, URL http://dx.doi.org/10.1007/978-3-642-20633-7_8. (Cited on page 1)

[9] Alves C, Cáceres E, de Castro J AA, Song S and Szwarcfiter J (2013) Parallel transitive closure algorithm, *Journal of the Brazilian Computer Society*, 19(2):161–166, URL http://dx.doi.org/10.1007/s13173-012-0089-z. (Cited on page 8)

[10] Andrews GE (1998) *The Theory of Partitions*, Cambridge mathematical library, Cambridge University Press, URL http://books.google.co.za/books?id=Sp7z9sK7RNkC. (Cited on page 58)

[11] ANSI/NISO Z39-19 (2005) Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies, http://www.niso.org/standards/standard_detail.cfm?std_id=814. (Cited on page 102)

[12] Antoniou G and van Harmelen F (2004) Web Ontology Language: OWL, in: Staab S and Studer R (Eds.) *Handbook on Ontologies*, International Handbooks on Information Systems, 67–92, Springer Berlin Heidelberg, URL http://dx.doi.org/10.1007/978-3-540-24750-0_4. (Cited on page 5)

[13] APA (2009) PsycINFO — the American Psychological Association's bibliographic database, http://apa.org/pubs/databases/psycinfo/index.aspx. [Online; accessed 2009-02-23]. (Cited on page 4)

[14] Arora S and Barak B (2009) *Computational Complexity: A Modern Approach*, New York, NY, USA: Cambridge University Press, 1st edn. (Cited on page 317)

[15] Ashton K (2009) That 'Internet of Things' Thing: In the real world, things matter more than ideas., http://www.rfidjournal.com/articles/view?4986. [Online; accessed 2016-11-10]. (Cited on page 2)

[16] Backhouse R and Ferreira JaF (2011) On Euclid's algorithm and elementary number theory, *Science of Computer Programming*, 76(3):160–180, URL http://dx.doi.org/10.1016/j.scico.2010.05.006. (Cited on pages 19 and 281)

[17] Baker JJ (1962) A note on multiplying Boolean matrices, *Communications of the ACM*, 5(2):102. (Cited on pages 7, 198, 204, 273 and 334)

[18] Banach R, Poppleton M, Jeske C and Stepney S (2007) Engineering and Theoretical Underpinnings of Retrenchment, *Science of Computer Programming*, 67(2-3):301–329, URL http://dx.doi.org/10.1016/j.scico.2007.04.002. (Cited on page 95)

[19] Bancilhon F (1986) Naive evaluation of recursively defined relations, in: *On knowledge base management systems: integrating artificial intelligence and database technologies*, 165–178, New York, NY, USA: Springer-Verlag New York, Inc. (Cited on page 7)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[20] Barla-Szabo G, Watson B and Kourie D (2004) Taxonomy of directed graph representations, *Software, IEE Proceedings*, 151(6):257–264. (Cited on pages 5 and 94)

[21] Barta R (2006) AsTMa= Language Definition, http://astma.it.bond.edu.au/astma=-spec-xtm.dbk. [Online; accessed 2012-08-03]. (Cited on page 22)

[22] Basoglu U and Morrison J (1978) The Efficient Hierarchical Data Structure for the US Historical Boundary File, *Harvard Papers on GIS*, 4:1–21. (Cited on page 128)

[23] Baswana S, Hariharan R and Sen S (2007) Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths, *Journal of Algorithms*, 62(2):74–92, URL http://www.sciencedirect.com/science/article/B6WH3-4DXB9BR-2/2/304496d808ea174a6c46fcf81eb5b130. (Cited on page 8)

[24] Bender MA, Farach-Colton M, Pemmasani G, Skiena S and Sumazin P (2005) Lowest common ancestors in trees and directed acyclic graphs, *Journal of Algorithms*, 57(2):75 – 94, URL http://www.sciencedirect.com/science/article/pii/S0196677405000854. (Cited on page 8)

[25] Berander P and Andrews AA (2005) Requirements Prioritization, *Engineering and Managing Software Requirements*, s:69 – 94. (Cited on page 140)

[26] Bergmann G, Ráth I, Szabó T, Torrini P and Varró D (2012) Incremental Pattern Matching for the Efficient Computation of Transitive Closure, in: Ehrig H, Engels G, Kreowski HJ and Rozenberg G (Eds.) *Graph Transformations*, vol. 7562 of *Lecture Notes in Computer Science*, 386–400, Springer Berlin Heidelberg, URL http://dx.doi.org/10.1007/978-3-642-33654-6_26. (Cited on page 8)

[27] Berners-Lee T, Hendler J and Lassila O (2001) The Semantic Web, *Scientific American*, 284(5):34–43. (Cited on page 2)

[28] Bielecki W, Kraska K and Klimek T (2013) Transitive Closure of a Union of Dependence Relations for Parameterized Perfectly-Nested Loops, in: Malyshkin V (Ed.) *Parallel Computing Technologies: 12th International Conference, PaCT 2013, St. Petersburg, Russia, September 30 - October 4, 2013. Proceedings*, 37–50, Berlin, Heidelberg: Springer Berlin Heidelberg, URL http://dx.doi.org/10.1007/978-3-642-39958-9_4. (Cited on page 8)

[29] Biezunski M and Newcomb SR (2001) XML Topic Maps: Finding Aids for the Web, *IEEE MultiMedia*, 8:104–108, URL http://0-dl.acm.org.innopac.up.ac.za/citation.cfm?id=614670.615049. (Cited on pages 105, 117 and 283)

[30] Bini D, Capovani M, Romani F and Lotti G (1979) $O(n^{2.7799})$ complexity for matrix multiplication, *Information Processing Letters*, 8(5):234–235. (Cited on page 174)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[31] Bitton D, Boral H, DeWitt DJ and Wilkinson WK (1983) Parallel Algorithms for the Execution of Relational Database Operations, *ACM Transactions on Database Systems (TODS)*, 8(3):324–353, URL http://0-doi.acm.org.innopac.up.ac.za/10.1145/319989.319991. (Cited on page 6)

[32] Black PE (1998) Dictionary of Algorithms and Data Structures, U.S. National Institute of Standards and Technology. http://xlinux.nist.gov/dads/. (accessed 17 January 2012). (Cited on pages 110 and 115)

[33] Bloom SL and Ésik Z (1993) Equational axioms for regular sets, *Mathematical Structures in Computer Science*, 3(1):1–24, URL https://www.cambridge.org/core/article/equational-axioms-for-regular-sets/5FE0310AB012061486DAB01DE5627D7C. (Cited on page 80)

[34] Booth KS and Lueker GS (1976) Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, *Journal of Computer and System Sciences*, 13(3):335 – 379, URL http://www.sciencedirect.com/science/article/pii/S0022000076800451. (Cited on page 128)

[35] Borgo S (2004) Classifying Medical Ontologies, http://www.slideworld.org/viewslides.aspx/Classifying--Medical--Ontologies-ppt-86089. [Online; accessed 2011-04-30. (Cited on page 102)

[36] Bosman RP (2005) *A Taxonomy of Approximate Pattern Matching Algorithms in Strings*, Master's thesis, Department of Mathematics and Computer Science, Technishe Universiteit Eindhoven, Eindhoven, The Netherlands. (Cited on page 94)

[37] Bozga M, Iosif R and Konecný F (2010) *Fast Acceleration of Ultimately Periodic Relations*, Tech. Rep. TR-2010-3, Verimag Technical Report. Version: 1. (Cited on page 8)

[38] Brainerd B (1970) Semi-Lattices and Taxonomic Systems, *Noûs*, 4(2):189 – 199, URL http://www.jstor.org/stable/2214321. (Cited on page 97)

[39] Briabrin VM (Ed.) (1976) *Conference on Artificial Intelligence: Question-Answering Systems: June 23-25, 1975*, 2361 Laxenburg, Austria: International Institute for Applied Systems Analysis. (Cited on page 103)

[40] Broekstra J, Klein M, Decker S, Fensel D, van Harmelen F and Horrocks I (2002) Enabling knowledge representation on the Web by extending RDF Schema, *Computer Networks*, 39(5):609 – 634, URL http://www.sciencedirect.com/science/article/B6VRG-45KT113-2/2/8720d2f32cc938a614a35b3dded9af9c. (Cited on page 5)

[41] Broughton V (2002) Facet analytical theory as a basis for a knowledge organization tool in a subject portal, in: López-Huertas MJ (Ed.) *Challenges in knowledge representation and organization for the 21st century: integration of*

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

knowledge across boundaries: proceedings of the the Seventh International ISKO Conference, 10-13 July 2002, Granada, Spain, 135 –142, Würzburg: Ergon Verlag. (Advances in Knowledge Organization; Vol 8). (Cited on page 98)

[42] Broughton V (2006) The need for a faceted classification as the basis of all methods of information retrieval, *Aslib Proceedings*, 58(1/2). (Cited on page 96)

[43] Broy M (1983) Program construction by transformations: a family tree of sorting programs, in: Biermann A and Guiho G (Eds.) *Computer Program Synthesis Methodologies*, 1–49, Dordrecht: Reidel. (Cited on pages 5, 94, 147 and 151)

[44] Caicedo AE (2006) The Knaster-Tarski theorem, `http://caicedoteaching.files.wordpress.com/2009/01/118-handout3.pdf`. [Online; accessed 2011-05-13]. (Cited on page 81)

[45] Cappello P, Egecioglu O and Scheiman C (2000) Processor-time-optimal Systolic Arrays, *International Journal of Parallel, Emergent and Distributed Systems*, 15(3):167–199. (Cited on page 216)

[46] Ceri S, Gottlob G and Tanca L (1989) What you always wanted to know about Datalog (and never dared to ask), *Knowledge and Data Engineering, IEEE Transactions on*, 1(1):146–166. (Cited on page 6)

[47] Chaffin R and Herrmann DJ (1987) Relation Element Theory: A New Account of the Representation and Processing of Semantic Relations, in: Gorfein DS and Hoffman RR (Eds.) *Memory and Learning: The Ebbinghaus Centennial Conference*, 221 – 246, Hillsdale, NJ: Lawrence Erlbaum Associates. (Cited on page 101)

[48] Chakradhar S, Agrawal V and Rothweiler S (1993) A transitive closure algorithm for test generation, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 12(7):1015–1028. (Cited on page 7)

[49] Chandra AK and Merlin PM (1977) Optimal implementation of conjunctive queries in relational data bases, in: *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, 77–90, New York, NY, USA: ACM. (Cited on page 7)

[50] Charras C and Lecroq T (1997) Handbook of exact string matching algorithms, `http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf`. [Online; accessed 2012-01-21]. (Cited on pages 112, 113, 114 and 115)

[51] Chen MY, Wei JD, Huang JH and Lee DT (2006) Design and applications of an algorithm benchmark system in a computational problem solving environment, *SIGCSE Bullitin*, 38(3):123–127. (Cited on page 140)

[52] Cheng J, Huang S, Wu H and Fu AWC (2013) TF-Label: A Topological-folding Labeling Scheme for Reachability Querying in a Large Graph, in: *Proceedings of the 2013 ACM SIGMOD International Conference on Management*

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

of Data, SIGMOD '13, 193–204, New York, NY, USA: ACM, URL http://doi.acm.org/10.1145/2463676.2465286. (Cited on page 8)

[53] Cheng J, Shang Z, Cheng H, Wang H and Yu JX (2012) K-reach: Who is in Your Small World, *Proceedings of the VLDB Endowment*, 5(11):1292–1303, URL http://dx.doi.org/10.14778/2350229.2350247. (Cited on page 8)

[54] Cleophas L, Kourie DG, Pieterse V, Schaefer I and Watson BW (2016) Correctness-by-Construction $\wedge$ Taxonomies $\Rightarrow$ Deep Comprehension of Algorithm Families, in: Margaria T and Steffen B (Eds.) *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques: 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10–14, 2016, Proceedings, Part I*, Cham: Springer International Publishing, URL http://dx.doi.org/10.1007/978-3-319-47166-2_54. (Cited on page 2)

[55] Cleophas L, Vosloo I and Watson BW (2005) TABASCO: a Taxonomy-based Domain Engineering Method, in: *Proceedings of JACQUARD2005, First Conference for the Software Engineering Community*, Zeist, The Netherlands. (Cited on page 1)

[56] Cleophas L and Watson B (2013) Applying and spicing-up TABASCO: Taxonomy-based software and how to increase its usability, in: Gruner S and Watson B (Eds.) *Formal Aspects of Computing: Essays Dedicated to Derrick Kourie on the Occasion of His 65th Birthday*, chap. 10, Aachen, Germany, Germany: Shaker Verlag GmbH, Germany. (Cited on pages 1, 6 and 285)

[57] Cleophas L, Watson BW, Kourie DG and Boake A (2005) TABASCO: a taxonomy-based domain engineering method, in: *SAICSIT '05: Proceedings of the 2005 annual research conference*, 38 – 47, Republic of South Africa: South African Institute for Computer Scientists and Information Technologists. (Cited on page 1)

[58] Cleophas L, Watson BW, Kourie DG, Boake A and Obiedkov S (2006) TABASCO: using concept-based taxonomies in domain engineering, *South African Computer Journal*, 30–40. (Cited on pages 100, 103 and 282)

[59] Cleophas L, Watson BW and Zwaan G (2010) A New Taxonomy of Sublinear Right-to-left Scanning Keyword Pattern Matching Algorithms, *Science of Computer Programming*, 75(11):1095–1112, URL http://dx.doi.org/10.1016/j.scico.2010.04.012. (Cited on pages 94 and 151)

[60] Cleophas LGWA (2008) *Tree Algorithms: Two Taxonomies and a Toolkit*, Ph.D. thesis, Technische Universiteit Eindhoven. (Cited on pages 5, 19, 35, 94, 98, 144, 147, 281 and 282)

[61] Codd EF (1970) A relational model of data for large shared data banks, *Communications of the ACM*, 13(6):377–387. (Cited on page 7)

[62] Codd EF (1990) *The relational model for database management: version 2*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on page 7)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[63] Coppersmith D (1982) Rapid multiplication of rectangular matrices, *SIAM Journal on Computing*, 11(3):467–471. (Cited on page 174)

[64] Coppersmith D (1997) Rectangular Matrix Multiplication Revisited, *Journal of Complexity*, 13(1):42–49. (Cited on page 174)

[65] Coppersmith D and Winograd S (1987) Matrix multiplication via arithmetic progressions, in: *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1–6, New York, NY, USA: ACM. (Cited on page 174)

[66] Coppersmith D and Winograd S (1990) Matrix multiplication via arithmetic progressions, *Journal of Symbolic Computation*, 9(3):251–280. (Cited on page 174)

[67] Dar S and Agrawal R (1993) Extending SQL with Generalized Transitive Closure, *IEEE Transactions on Knowledge and Data Engineering*, 5(5):799–812. (Cited on page 7)

[68] Dar S and Jagadish HV (1992) A spanning tree transitive closure algorithm, in: *Eighth International Conference on Data Engineering*, 2 – 11, Wisconsin Univ., Madison, WI, URL `http://0-ieeexplore.ieee.org.innopac.up.ac.za:80/iel2/386/5565/00213213.pdf`. [Online; accessed 2008-08-15]. (Cited on page 7)

[69] Daraghmi EY and Yuan SM (2014) We are so close, less than 4 degrees separating you and me!, *Computers in Human Behavior*, 30:273 – 285, URL `http://www.sciencedirect.com/science/article/pii/S0747563213003427`. (Cited on pages 9 and 10)

[70] Darlington J (1978) A synthesis of several sorting algorithms, *Acta Informatica*, 11(1):1–30. (Cited on page 94)

[71] Davis M (1958) *Computability & unsolvability*, McGraw-Hill series in information processing and computers, Dover. (Cited on page 133)

[72] Demetrescu C and Italiano GF (2005) Trade-offs for fully dynamic transitive closure on DAGs: breaking through the $O(n^2)$ barrier, *Journal of the ACM (JACM)*, 52(2):147–156. (Cited on page 8)

[73] Demetrescu C and Italiano GF (2008) Mantaining Dynamic Matrices for Fully Dynamic Transitive Closure, *Algorithmica*, 51(4):387–427. (Cited on page 8)

[74] Diamantini C and Potena D (2008) Representing Service Information in a Collaborative KDD Environment, in: *Proceedings of the International Symposium on Collaborative Technologies and Systems*, 331 – 338, Irvine, CA, USA: IEEE. (Cited on page 112)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[75] Diamantini C, Potena D and Storti E (2009) KDDONTO: An Ontology for Discovery and Composition of KDD Algorithms, in: *Proceedings of the ECML PKDD 2009 Workshop on Service-oriented Knowledge Discovery*, 13 – 24. (Cited on page 115)

[76] Dijkstra EW (1974) Determinism and recursion versus non-determinism and the transitive closure, `http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD456.PDF`. Citeulike:872534, [Online: 2015-07-17]. (Cited on page 6)

[77] Dijkstra EW (1976) *A discipline of programming*, Englewood Cliffs, N.J.: Prentice-Hall. (Cited on pages 32 and 39)

[78] Dijkstra EW (2002) EWD1300: The Notational Conventions I Adopted, and Why, *Formal Aspects of Computing*, 14:99–107, URL `http://dx.doi.org/10.1007/s001650200030`. (Cited on pages 19, 281 and 286)

[79] Dijkstra EW and Scholten CS (1990) *Predicate calculus and program semantics*, New York, NY, USA: Springer-Verlag New York, Inc. (Cited on pages 37 and 39)

[80] Dijkstra EW and van Gasteren AJM (1986) On notation, `http://www.cs.utexas.edu/users/EWD/ewd09xx/EWD950a.PDF`. Citeulike:873118, [Online: 2013-11-27]. (Cited on page 281)

[81] Ding Z, Shu W and Wu MY (2011) FPGA Based Parallel Transitive Closure Algorithm, in: *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, 393–394, New York, NY, USA: ACM, URL `http://doi.acm.org/10.1145/1982185.1982270`. (Cited on page 8)

[82] Dodds PS, Muhamad R and Watts DJ (2003) An Experimental Study of Search in Global Social Networks, *Science*, 301(5634):827–829. (Cited on pages 9 and 10)

[83] Dresch A, Lacerda DP and Antunes JAV (2014) *Design Science Research: A Method for Science and Technology Advancement*, Springer Publishing Company, Incorporated. (Cited on page 284)

[84] Ebert J (1981) A sensitive transitive closure algorithm, *Information Processing Letters*, 12(5):255–258. (Cited on page 7)

[85] Edelkamp S, Elmasry A and Katajainen J (2012) A Catalogue of Algorithms for Building Weak Heaps, in: Arumugam S and Smyth W (Eds.) *Combinatorial Algorithms*, vol. 7643 of *Lecture Notes in Computer Science*, 249–262, Springer Berlin Heidelberg, URL `http://dx.doi.org/10.1007/978-3-642-35926-2_27`. (Cited on page 103)

[86] Erdös P and Rado R (1956) A partition calculus in set theory, *Bulletin of the American Mathematical Society*, 427 – 489. (Cited on pages 58, 68 and 282)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[87] Eve J and Kurki-Suonio R (1977) On computing the transitive closure of a relation, *Acta Informatica*, 8:303 – 314. (Cited on page 7)

[88] Eyas EQ (2004) Quick Matrix Multiplication on Clusters of Workstations, *Informatica*, 15(2):203–218. (Cited on page 174)

[89] Fellbaum C (Ed.) (1998) *WordNet : an electronic lexical database*, Cambridge, Mass: MIT Press. (Cited on page 26)

[90] Fensel D (2004) *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, New York: NY: Springer Verlag. (Cited on page 5)

[91] Fischer MJ and Meyer A (1971) Boolean matrix multiplication and transitive closure, in: *Switching and Automata Theory, 1971., 12th Annual Symposium on*, 129–131. (Cited on page 7)

[92] Floyd RW (1962) Algorithm 96: Ancestor, *Communications of the ACM*, 5(6):344–345. (Cited on pages 8, 180 and 329)

[93] Fouh E, Akbar M and Shaffer CA (2012) The Role of Visualization in Computer Science Education, *Computers in the Schools*, 29(1-2):95–117, URL `http://www.tandfonline.com/doi/abs/10.1080/07380569.2012.651422`. (Cited on page 140)

[94] Fox R (2005) Cataloguing our information architecture, *OCLC Systems and Services: International Digital Library Perspectives*, 21(1):23 – 29. (Cited on page 96)

[95] Frank AU (2006) Distinctions Produce a Taxonomic Lattice: Are These the Units of Mentalese?, in: *Proceeding of the 2006 conference on Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006)*, 27–38, Amsterdam, The Netherlands, The Netherlands: IOS Press. (Cited on page 97)

[96] Ganter B (1999) Attribute exploration with background knowledge, *Theoretical Computer Science*, 217(2):215 – 233, URL `http://www.sciencedirect.com/science/article/B6V1G-3WSV1V8-3/2/6ec16e9dc233199c165577eafd359c1f`. ORDAL'96. (Cited on page 100)

[97] Garshol LM (2004) Metadata? Thesauri? Taxonomies? Topic Maps! Making Sense of it all, *Journal of Information Science*, 30(4):378–391, URL `http://jis.sagepub.com/cgi/content/abstract/30/4/378`. (Cited on page 93)

[98] Garshol LM (2006) The Linear Topic Map Notation: Definition and introduction, version 1.3, `http://www.ontopia.net/download/ltm.html`. [Online; accessed 2011-11-23]. (Cited on pages 22 and 38)

[99] Garshol LM (n.d.) Topic maps in content management: The rise of the ITMS, `http://www.ontopia.net/topicmaps/materials/itms.html`. [Online; accessed 2011-11-23]. (Cited on pages 21 and 22)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[100] Gibbons A, Pagourtzis A, Potapov I and Rytter W (2003) Coarse-Grained Parallel Transitive Closure Algorithm: Path Decomposition Technique, *The Computer Journal*, 46(4):391–400, URL `http://comjnl.oxfordjournals.org/content/46/4/391.abstract`. (Cited on page 8)

[101] Gilchrist A (2003) Thesauri, taxonomies and ontologies – an etymological note, *Journal of Documentation*, 59(1):7 – 18. (Cited on page 4)

[102] Goes PB (2014) Editor's Comments: Design Science Research in Top Information Systems Journals, *MIS Q.*, 38(1):iii–viii, URL `http://0-dl.acm.org.innopac.up.ac.za/citation.cfm?id=2600518.2600519`. (Cited on page 283)

[103] Gonnet G and Tompa F (1983) A constructive approach to the design of algorithms and their data structures, *Communications of the ACM*, 26(11):912–920. (Cited on page 128)

[104] Gregor S and Hevner AR (2013) Positioning and Presenting Design Science Research for Maximum Impact, *MIS Quaterly*, 37(2):337–356, URL `http://dl.acm.org/citation.cfm?id=2535658.2535660`. (Cited on pages 1 and 284)

[105] Griem G and Oliker L (2003) Transitive closure on the imagine stream processor, `http://escholarship.org/uc/item/7kb6v5gj`. (Cited on page 216)

[106] Gruner S (2013) Abstraction, Refinement, Enrichment, in: Gruner S and Watson B (Eds.) *Formal Aspects of Computing: Essays dedicated to Derrick Kourie on the occasion of his 65th Birthday*, chap. 1, 13–43, Aachen, Germany, Germany: Shaker Verlag GmbH, Germany. (Cited on page 95)

[107] Guarino N and Giaretta P (1995) Ontologies and Knowledge Bases: Towards a Terminological Clarification, *Towards Very Large Knowledge Bases*, 1(9):25 – 32, URL `http://www.cs.umbc.edu/courses/771/papers/KBKS95.pdf.Z`. (Cited on page 102)

[108] Guibas LJ, Kung H and Thompson CD (1979) Direct VLSI implementation of combinatorial algorithms, in: *Proceedings of the CalTech Conference on VLSI*, 509 – 525, California Institute of Technology. (Cited on page 7)

[109] Gurevitch M (1961) *The social structure of acquaintanceship networks*, Ph.D. thesis, Massachusetts Institute of Technology, Dept. of Economics and Social Science. (Cited on page 9)

[110] Hartmanis J (1977) Relations between diagonalization, proof systems, and complexity gaps, in: *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC '77, 223–227, New York, NY, USA: ACM, URL `http://0-doi.acm.org.innopac.up.ac.za/10.1145/800105.803412`. (Cited on page 134)

[111] Hedden H (2010) *The Accidental Taxonomist*, Medford, NJ: Information Today Inc. (Cited on pages 97 and 101)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[112] Henzinger MR and King V (1995) Fully dynamic biconnectivity and transitive closure, *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 664 – 672. (Cited on page 7)

[113] Hevner AR, March ST, Park J and Ram S (2004) Design Science in Information Systems Research, *MIS Quarterly*, 28(1):75–105, URL `http://dl.acm.org/citation.cfm?id=2017212.2017217`. (Cited on pages 1 and 284)

[114] Hjørland B (2002) Domain analysis in information science: Eleven approaches - traditional as well as innovative, *Journal of Documentation*, 58(4):422 – 462. (Cited on page 4)

[115] Hopkins T (2009) The collected algorithms of the ACM (CALGO), *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(3):316–324, URL `http://dx.doi.org/10.1002/wics.40`. (Cited on pages 180 and 191)

[116] Houtsma MAW, Wilschut AN and Flokstra J (1993) Implementation and Performance Evaluation of a Parallel Transitive Closure Algorithm on PRISMA/DB, in: *VLDB '93: Proceedings of the 19th International Conference on Very Large Data Bases*, 206–217, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (Cited on page 256)

[117] Hundhausen CD, Douglas SA and Stasko JT (2002) A Meta-Study of Algorithm Visualization Effectiveness, *Journal of Visual Languages & Computing*, 13(3):259 – 290, URL `http://www.sciencedirect.com/science/article/pii/S1045926X02902375`. (Cited on page 140)

[118] Hungerford TW (2014) *Abstract Algebra: An Introduction*, Boston, USA: BROOKS/COLE Cengage Learning, third edn. (Cited on page 64)

[119] Ingleton M and Ahmed K (2009) Enterprise information management specialist brings 'Topic Map' solution to the UK, `http://www.targetwire.com/targetwire/2006/02/27/tw173/tw173_uk.html`. [Online; accessed 2012-01-19]. (Cited on page 105)

[120] Ioannidis Y, Ramakrishnan R and Winger L (1993) Transitive closure algorithms based on graph traversal, *ACM Transactions on Database Systems (TODS)*, 18(3):512–576. (Cited on page 7)

[121] Ioannidis YE (1986) On the Computation of the Transitive Closure of Relational Operators, in: *Proceedings of the 12th International Conference on Very Large Data Bases*, VLDB '86, 403–411, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., URL `http://0-dl.acm.org.innopac.up.ac.za/citation.cfm?id=645913.671476`. (Cited on page 7)

[122] Ioannidis YE and Ramakrishnan R (1988) Efficient Transitive Closure Algorithms, in: *VLDB '88: Proceedings of the 14th International Conference on Very Large Data Bases*, 382–394, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (Cited on page 7)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[123] Ioannidis YE and Wong E (1989) Transforming Nonlinear Recursion into Linear Recursion, in: Kerschberg L (Ed.) *Proceedings from the Second International Conference on Expert Database Systems*, 401–421, Redwood City, Calif.: Benjamin/Cummings Publishing. (Cited on page 7)

[124] ISO/IEC 13250 (1999) Topic Maps, http://www1.y12.doe.gov/capabilities/sgml/sc34/document/0129.pdf. [Online; accessed 2011-01-10]. (Cited on pages 17, 104, 105, 106 and 283)

[125] ISO/IEC 13250 (2002) Topic Maps. Information Technology. Document Description and Processing Languages. Second Edition, http://www1.y12.doe.gov/capabilities/sgml/sc34/document/0322_files/iso13250-2nd-ed-v2.pdf. [Online; accessed 2012-07-16]. (Cited on pages 5 and 17)

[126] ISO/TC 12 Quantities and units (2009) ISO 80000-2:2009 Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology, http://www.iso.org/iso/catalogue_detail.htm?csnumber=31887. [Online; accessed 2013-12-22]. (Cited on page 281)

[127] Italiano G (1986) Amortized efficiency of a path retrieval data structure, *Theoretical Computer Science*, 48:273 – 281, URL http://www.sciencedirect.com/science/article/pii/0304397586900988. (Cited on page 7)

[128] Jiang CJ and Wu ZH (1997) Two new algorithms for matrix multiplication and vector convolution, *International Journal of Computer Mathematics*, 63(1):27–36, URL http://dx.doi.org/10.1080/00207169708804549. (Cited on page 174)

[129] Jones ND (1967) Classes of automata and transitive closure, in: *Switching and Automata Theory, 1967. SWAT 1967. IEEE Conference Record of the Eighth Annual Symposium on*, 296–306. (Cited on page 7)

[130] Jonkers HBM (1982) *Abstraction, specification and implementation techniques : with an application to garbage collection*, Ph.D. thesis, Technische Hogeschool Eindhoven. (Cited on pages 2, 94, 151 and 283)

[131] Kaldewaij A (1990) *Programming: The Derivation of Algorithms*, Upper Saddle River, NJ, USA: Prentice-Hall, Inc. (Cited on page 107)

[132] Kang JM, Im YS, Lee KY, Lim MJ, Lee YD, Kang Ey, Kang MK and Oh S (2009) A study for semantics participation platform architecture using RDF/OWL, in: *Proceedings of the 2009 International Conference on Hybrid Information Technology*, ICHIT '09, 512–515, New York, NY, USA: ACM, URL http://0-doi.acm.org.innopac.up.ac.za/10.1145/1644993.1645088. (Cited on page 5)

[133] Karinthy F (1929) Chain links, https://djjr-courses.wdfiles.com/local--files/soc180:karinthy-chain-links/Karinthy-Chain-Links_1929.pdf. [Online; accessed 2015-11-05]. (Cited on page 9)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[134] Karp RM (1990) The transitive closure of a random digraph, *Random Structures & Algorithms*, 1(1):73–93, URL http://dx.doi.org/10.1002/rsa.3240010106. (Cited on page 7)

[135] Karp RM, Miller RE and Winograd S (1967) The Organization of Computations for Uniform Recurrence Equations, *J. ACM*, 14(3):563–590, URL http://0-doi.acm.org.innopac.up.ac.za/10.1145/321406.321418. (Cited on page 7)

[136] Ketcha Ngassam E, Kourie DG and Watson BW (2006) A taxonomy of DFA-based string processors, in: *SAICSIT '06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, 238–246, , Republic of South Africa: South African Institute for Computer Scientists and Information Technologists. (Cited on page 94)

[137] Kilp M, Knauer U and Mikhalev AV (2000) *Monoids, Acts, and Categories: With Applications to Wreath Products and Graphs : a Handbook for Students and Researchers*, De Gruyter expositions in mathematics, W. de Gruyter. (Cited on page 52)

[138] King V (1999) Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs, in: *Foundations of Computer Science, 1999. 40th Annual Symposium on*, 81–89. (Cited on page 7)

[139] King V and Sagert G (2002) A Fully Dynamic Algorithm for Maintaining the Transitive Closure, *Journal of Computer and System Sciences*, 65(1):150–167. (Cited on page 8)

[140] Kleinberg J and Tardos É (2006) *Algorithm Design*, Alternative Etext Formats, Pearson/Addison-Wesley, URL https://books.google.co.za/books?id=OiGhQgAACAAJ. (Cited on page 107)

[141] Knuth DE (1997) *The art of computer programming, volume 1 (3rd ed.): fundamental algorithms*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on pages 107 and 122)

[142] Knuth DE (1997) *The art of computer programming, volume 2 (3rd ed.): Seminumerical Algorithms*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on page 107)

[143] Knuth DE (1998) *The art of computer programming, volume 3 (2nd ed.): Sorting and Searching*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on page 107)

[144] Knuth DE (2005) *The art of computer programming, volume 1 (Fascicle 1): MMIX – A RISC Computer for the New Millennium*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on page 107)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[145] Knuth DE (2011) *The art of computer programming, volume 4A : Combinatorial Algorithms*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on page 107)

[146] Konečný F (2016) PTIME Computation of Transitive Closures of Octagonal Relations, in: Chechik M and Raskin JF (Eds.) *Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, 645–661, Berlin, Heidelberg: Springer Berlin Heidelberg, URL `http://dx.doi.org/10.1007/978-3-662-49674-9_42`. (Cited on page 8)

[147] Kourie D and Watson B (2012) *The Correctness-By-Construction Approach to Programming*, Springer, URL `http://books.google.co.za/books?id=5Ig6ELUQFM4C`. (Cited on pages 32, 35, 36, 39 and 95)

[148] Kourie DG (1989) An Approach to Defining Abstractions, Refinements and Enrichments, *Quæstiones Informaticæ*, 6(4):174–178. (Cited on pages 95, 96 and 106)

[149] Kouwenberg JJC (2003) *A Taxonomy of Lempel-Ziv Compression Algorithms*, Master's thesis, Department of Mathematics and Computer Science, Technishe Universiteit Eindhoven, Eindhoven, The Netherlands. (Cited on page 94)

[150] Kovács L (2014) Role of negative properties in knowledge modeling, in: *Proceedings of the* 9th *International Conference on Applied Informatics*, vol. 1, 67 – 74, Eger, Hungary. (Cited on page 95)

[151] Ksiezyk R (1998) Plato, SGML and revolution, in: *Proceedings of the SGML/XML Europe '98 Conference*, Alexandria: GCA. Available online at `http://www.infoloom.com/gcaconfs/WEB/paris98/ksiezyk.HTM`. (Cited on page 105)

[152] Kung SY, Lo SC and Lewis PS (1987) Optimal Systolic Design for the Transitive Closure and the Shortest Path Problems, *IEEE Transactions on Computers*, 36(5):603–614. (Cited on page 216)

[153] Lam MD, Rothberg EE and Wolf ME (1991) The Cache Performance and Optimizations of Blocked Algorithms, in: *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IV, 63–74, New York, NY, USA: ACM, URL `http://0-doi.acm.org.innopac.up.ac.za/10.1145/106972.106981`. (Cited on page 215)

[154] Lamping J, Rao R and Pirolli P (1995) A focus+context technique based on hyperbolic geometry for visualizing large hierarchies, in: *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, 401–408, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co. (Cited on page 102)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[155] Librelotto GR, Ramalho JC and Henriques PR (2008) A framework to specify, extract and manage topic maps driven by ontology, in: *Proceedings of the 26th annual ACM international conference on Design of communication*, SIGDOC '08, 155–162, New York, NY, USA: ACM. (Cited on page 105)

[156] Łącki J (2013) Improved Deterministic Algorithms for Decremental Reachability and Strongly Connected Components, *ACM Trans. Algorithms*, 9(3):27:1–27:15, URL `http://doi.acm.org/10.1145/2483699.2483707`. (Cited on page 8)

[157] Loveman DB (1977) Program Improvement by Source-to-Source Transformation, *Journal of the ACM*, 24(1):121–145, URL `http://0-doi.acm.org.innopac.up.ac.za/10.1145/321992.322000`. (Cited on page 205)

[158] Manber U (1989) *Introduction to Algorithms: A Creative Approach*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on page 107)

[159] Marcelis AJJM (1990) On the classification of attribute evaluation algorithms, *Science of Computer Programming*, 14(1):1–24. (Cited on page 94)

[160] Marchetti-Spaccamela A, Nanni U and Rohnert H (1996) Maintaining a topological order under edge insertions, *Information Processing Letters*, 59(1):53–58, URL `http://www.sciencedirect.com/science/article/B6V0F-3VVCJTY-C/2/5273cacdc116cadf51ac84df8c094eca`. (Cited on page 7)

[161] Markov A and Nagorny N (1988) *The Theory of Algorithms*, Mathematics and its Applications (Kluwer Academic).: Soviet Series, Springer. (Cited on page 122)

[162] Martynyuk VV (1962) An efficient construction of the transitive closure of a binary relation, *Zhurnal vychislitel'noi matematiki i matematicheskoi fiziki*, 2(4):723–725. (Cited on pages 7, 191, 196 and 273)

[163] Martynyuk VV (1963) The economical construction of a transitive closure of a binary relation, *USSR Computational Mathematics and Mathematical Physics* , 2(4):817 – 821, URL `http://www.sciencedirect.com/science/article/pii/0041555363905469`. (Cited on page 191)

[164] Martynyuk VV (1973) Transitively equivalent directed graphs, *Journal Cybernetics and Systems Analysis*, 9(1):45 – 49. Translated from Kibernetika, No. 1, pp. 39 – 43, January – February, 1973. (Cited on pages 7, 8 and 333)

[165] McCarthy J (1980) Circumscription–A Form of Non-Monotonic Reasoning, *Artificial Intelligence*, 13(1):27–39. (Cited on page 102)

[166] Merriam-Webster (2010) index — Merriam-Webster Online Dictionary, `http://www.merriam-webster.com/dictionary/index`. [Online; accessed 17-December-2010]. (Cited on page 104)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[167] Merritt SM (1985) An inverted taxonomy of sorting algorithms, *Communications of the ACM*, 28(1):96–99. (Cited on pages 94 and 127)

[168] Merritt SM (1994) An Expanded Taxonomy of Sorting Algorithms, *Computer Science Education*, 5(1):103–110, URL `http://www.informaworld.com/10.1080/0899340940050107`. (Cited on page 127)

[169] Milgram S (1967) The small world problem, *Psychology Today*, 60 – 67. (Cited on pages 9 and 10)

[170] Miller FP, Vandome AF and McBrewster J (2010) *Big O Notation*, Mauritius: VDM Publishing House. (Cited on page 134)

[171] Milovanović IZ, Milovanović EI and Randjelović BM (2005) Computing Transitive Closure Problem on Linear Systolic Array, in: Li Z, Vulkov L and Wasniewski J (Eds.) *Numerical Analysis and its Applications*, vol. 3401/2005 of *Lecture Notes in Computer Science*, 416–423, Springer Berlin / Heidelberg. (Cited on page 216)

[172] Minsky ML (1967) *Computation: finite and infinite machines*, Upper Saddle River, NJ, USA: Prentice-Hall, Inc. (Cited on page 122)

[173] Moore EF (1959) The shortest path through a maze, in: *Proceedings of the International Symposium on the Theory of Switching*, 285–292, Harvard University Press. (Cited on page 7)

[174] Morgan C (1990) *Programming from Specifications*, 666 Wood Lane End, Hemel Hempstead, Hertfordshire: Prentice-Hall International(UK), Ltd. (Cited on pages 94, 95 and 96)

[175] Moschovakis YN (2000) What is an algorithm?, in: Engquist B and Schmidt W (Eds.) *Mathematics Unlimited: 2001 and Beyond*, 919 – 936, Springer. (Cited on page 122)

[176] Munro I (1971) Efficient Determination of the Transitive Closure of a Directed Graph, *Information Processing Letters*, 56 – 58. (Cited on page 7)

[177] Newcomb SR (2003) A Perspective on the Quest for Global Knowledge Interchange, in: Hunting S and Park J (Eds.) *XML Topic Maps: Creating and Using Topic Maps for the Web*, chap. 3, 22 – 34, Addison-Wesley Professional. (Cited on pages 105 and 117)

[178] Niesink P, Poulin K and ŠAjna M (2013) Computing Transitive Closure of Bipolar Weighted Digraphs, *Discrete Applied Mathematics*, 161(1-2):217–243, URL `http://dx.doi.org/10.1016/j.dam.2012.06.013`. (Cited on page 8)

[179] NLM (1999) MEDLINE — the National Library of Medicine's bibliographic database, `http://www.nlm.nih.gov/databases/databases_medline.html`. [Online; accessed 2009-02-23]. (Cited on page 4)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[180] Nuutila E (1994) An efficient transitive closure algorithm for cyclic digraphs, *Information Processing Letters*, 52(4):207 – 213, URL http://www.sciencedirect.com/science/article/pii/0020019094901287. (Cited on page 7)

[181] OED (1989) Taxonomy — The Oxford English Dictionary, *OED Online* 2nd Ed., URL http://0-dictionary.oed.com.innopac.up.ac.za/cgi/entry/50247829. [Online; accessed 2008-08-15]. (Cited on page 97)

[182] OED (2009) Reification — The Oxford English Dictionary, *OED Online* 3rd Ed., http://0-dictionary.oed.com.innopac.up.ac.za/view/entry/161512. [Online; accessed 2012-01-05]. (Cited on page 31)

[183] Ogievetsky N (2003) Creating and Maintaining Enterprise Web Sites with Topic Maps and XSLT, in: Hunting S and Park J (Eds.) *XML Topic Maps: Creating and Using Topic Maps for the Web*, chap. 9, 122–146, Addison-Wesley Professional. (Cited on page 104)

[184] Osipov V, Sanders P and Singler J (2009) The Filter-Kruskal Minimum Spanning Tree Algorithm, in: *Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments*, ALENEX 09, 52 – 61, Philadelphia, USA: Society for Industrial and Applied Mathematics. (Cited on page 134)

[185] Pagourtzis A, Potapov I and Rytter W (2002) Observations on Parallel Computation of Transitive and Max-Closure Problems, in: Di Martino B, Kranzlmüller D and Dongarra J (Eds.) *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 2474/2002 of *Lecture Notes in Computer Science*, 217–225, Berlin / Heidelberg: Springer. (Cited on pages 216 and 256)

[186] Pan V (1984) How can we speed up Matrix Multiplication?, *SIAM Review*, 26(3):393–415. (Cited on page 174)

[187] Pan V (1984) *How to multiply matrices faster*, New York, NY, USA: Springer-Verlag New York, Inc. (Cited on page 174)

[188] Pan V (2015) Matrix Multiplication, Trilinear Decompositions, APA Algorithms, and Summation, *CUNY Academic Works*. (Cited on page 174)

[189] Papadimitriou CH (2003) Computational complexity, in: *Encyclopedia of Computer Science*, 260–265, Chichester, UK: John Wiley and Sons Ltd., URL http://0-dl.acm.org.innopac.up.ac.za/citation.cfm?id=1074100.1074233. (Cited on pages 133 and 134)

[190] Peffers K, Tuunanen T, Rothenberger MA and Chatterjee S (2007) A Design Science Research Methodology for Information Systems Research, *Journal of Management Information Systems*, 24(3):45–77, URL http://www.tandfonline.com/doi/abs/10.2753/MIS0742-1222240302. (Cited on page 145)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[191] Pelekis N and Theodoridis Y (2014) *Mobility Database Management*, 75–99, New York, NY: Springer New York, URL `http://dx.doi.org/10.1007/978-1-4939-0392-4_4`. (Cited on page 103)

[192] Penner M and Prasanna VK (2006) Cache-Friendly implementations of transitive closure, *J. Exp. Algorithmics*, 11:1–3. (Cited on pages 215, 216 and 232)

[193] Pepper S (1999) Navigating haystacks and discovering needles: introducing the new topic map standard, *Markup Languages: Theory & Practice*, 1:47–74, URL `http://0-portal.acm.org.innopac.up.ac.za/citation.cfm?id=335435.335445`. (Cited on pages 22, 29, 105, 117 and 154)

[194] Pepper S (2002) Ten Theses on Topic Maps and RDF, `http://www.ontopia.net/topicmaps/materials/rdf.html`. [Online; accessed 2011-05-08]. (Cited on pages 5, 21 and 104)

[195] Pepper S (2010) Topic Maps, in: Bates MJ and Maack MN (Eds.) *Encyclopedia of Library and Information Sciences*, 5247 – 5259, Taylor & Francis, 3rd edn. (Cited on pages 31 and 104)

[196] Pepper S and Moore G (2001) XML Topic Maps (XTM) 1.0: TopicMaps.Org Specification, `http://www.topicmaps.org/xtm/`. [Online; accessed 2011-05-08]. (Cited on pages 22 and 26)

[197] Pettie S and Ramachandran V (2002) An optimal minimum spanning tree algorithm, *Journal of the ACM*, 49(1):16–34, URL `http://0-doi.acm.org.innopac.up.ac.za/10.1145/505241.505243`. (Cited on page 134)

[198] Pieterse V and Flater D (2014) *The ghost in the machine: don't let it haunt your software performance measurements*, Tech. Rep. NIST TN 1830, National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899. `dx.doi.org/10.6028/NIST.TN.1830`. (Cited on page 283)

[199] Pieterse V and Kourie DG (2014) Lists, Taxonomies, Lattices, Thesauri and Ontologies: Paving a pathway through a terminological jungle, *Knowledge Organization*, 41(3):217 – 229. (Cited on pages 4, 93, 102, 106 and 282)

[200] Pouchet LN, Bondhugula U, Bastoul C, Cohen A, Ramanujam J, Sadayappan P and Vasilache N (2011) Loop Transformations: Convexity, Pruning and Optimization, in: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, 549–562, New York, NY, USA: ACM, URL `http://0-doi.acm.org.innopac.up.ac.za/10.1145/1926385.1926449`. (Cited on page 205)

[201] Pries-Heje J and Baskerville R (2008) The Design Theory Nexus., *MIS Quarterly*, 32(4):731–755, URL `http://dblp.uni-trier.de/db/journals/misq/misq32.html#Pries-HejeB08`. (Cited on page 5)

[202] Priss U (2006) Formal concept analysis in information science, *Annual Review of Information Science and Technology*, 40(1):521 – 543. (Cited on page 102)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[203] Prosser RT (1959) Applications of Boolean matrices to the analysis of flow diagrams, in: *Proceedings of the Eastern Joint Computer Conference No. 16*, 133–138. (Cited on pages 7, 8, 30, 175, 184 and 327)

[204] Purdom P Jr (1970) A transitive closure algorithm, *BIT Numerical Mathematics*, 10(1):76–94. (Cited on page 7)

[205] Ranganathan SR (1962) *Elements of library classification*, Bombay: Asia Publishing House. (Cited on pages 95, 106 and 283)

[206] Ranganathan SR (1967) *Prolegomena to library classification*, New York: Asia Publishing House. (Cited on pages 95 and 101)

[207] Rao S, Citron T and Kailath T (1985) Mesh-connected processor arrays for the transitive closure problem, in: *Decision and Control, 1985 24th IEEE Conference on*, 1565–1570. (Cited on page 7)

[208] Roditty L (2008) A faster and simpler fully dynamic transitive closure, *ACM Transansactions on Algorithms*, 4(1):1–16. (Cited on page 8)

[209] Roget PM (1912) *Thesaurus of English words and phrases*, London: Dent. (Cited on pages 101 and 283)

[210] Roy B (1959) Transitivité et connexité, *Comptes Rendus de l'Académie des Sciences Paris*, 249:216 – 218, URL http://gallica.bnf.fr/ark:/12148/bpt6k3201c/f222. (Cited on page 7)

[211] Russell B (1903) *The principles of mathematics*, Cambridge: Cambridge Univiversity Press, URL http://fair-use.org/bertrand-russell/the-principles-of-mathematics/. (Cited on pages 7 and 80)

[212] Sankowski P (2004) Dynamic transitive closure via dynamic matrix inverse: extended abstract, *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, 509–517. (Cited on page 8)

[213] Sankowski P and Mucha M (2010) Fast Dynamic Transitive Closure with Lookahead, *Algorithmica*, 56(2):180–197, URL http://dx.doi.org/10.1007/s00453-008-9166-2. (Cited on page 8)

[214] Scheiman CJ and Cappello P (1992) A processor-time-minimal systolic array for transitive closure, *Parallel and Distributed Systems, IEEE Transactions on*, 3(3):257–269. (Cited on pages 216 and 256)

[215] Schmitz L (1983) An improved transitive closure algorithm, *Computing*, 30:359 – 371. (Cited on page 7)

[216] Schneider G, Gersting J and Miller K (2009) *Invitation to Computer Science*, Introduction to CS Series, Course Technology/Cengage Learning, URL http://books.google.co.za/books?id=gQK0pJONyhgC. (Cited on pages 122 and 143)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[217] Schnorr C (1987) A hierarchy of polynomial time lattice basis reduction algorithms, *Theoretical Computer Science*, 53(2):201 – 224, URL http://www.sciencedirect.com/science/article/pii/0304397587900648. (Cited on page 94)

[218] Schnorr CP (1978) An Algorithm for Transitive Closure with Linear Expected Time, *SIAM Journal on Computing*, 7(2):127–133, URL http://link.aip.org/link/?SMJ/7/127/1. (Cited on page 7)

[219] Schock R (1979) On classifications and hierarchies, *Journal for General Philosophy of Science*, 10:98–106. (Cited on page 97)

[220] Schonhage A (1981) Partial and Total Matrix Multiplication, *SIAM Journal on Computing*, 10(3):434–455, URL http://link.aip.org/link/?SMJ/10/434/1. (Cited on page 174)

[221] Schudy WJ (2008) Finding strongly connected components in parallel using o(log2n) reachability queries, in: *SPAA '08: Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, 146–151, New York, NY, USA: ACM. (Cited on page 256)

[222] Sen S, Chatterjee S and Dumir N (2002) Towards a theory of cache-efficient algorithms, *Journal of the ACM (JACM)*, 49(6):828–858. (Cited on page 215)

[223] Shaffer CA, Naps TL, Rodger SH and Edwards SH (2010) Building an online educational community for algorithm visualization, in: *Proceedings of the 41st ACM technical symposium on Computer Science Education*, SIGCSE '10, 475–476, New York, NY, USA: ACM, URL http://0-doi.acm.org.innopac.up.ac.za/10.1145/1734263.1734421. (Cited on page 141)

[224] Simmons B (2012) Interval Notation, http://www.mathwords.com/i/interval_notation.htm. [Online; 2012-11-08]. (Cited on page 21)

[225] Simon HA (1996) *The Sciences of the Artificial (3rd Ed.)*, Cambridge, MA, USA: MIT Press. (Cited on page 1)

[226] Skiena SS (2009) *The algorithm design manual*, New York, NY, USA: Springer-Verlag New York, Inc., 2nd edn. (Cited on pages 80, 108, 109, 110, 115, 128 and 134)

[227] Sowa JF (1984) *Conceptual structures: information processing in mind and machine*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. (Cited on pages 93 and 102)

[228] Spiteri LF (1998) A Simplified Model for Facet Analysis, *the Canadian Journal of Information and Library Science*, 23(1):1 – 30. (Cited on page 95)

[229] Stern L, Søndergaard H and Naish L (1999) A strategy for managing content complexity in algorithm animation, in: *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and Technology in Computer*

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Science Education, ITiCSE '99, 127–130, New York, NY, USA: ACM, URL http://0-doi.acm.org.innopac.up.ac.za/10.1145/305786.305891. (Cited on page 141)

[230] Stone HS (1971) *Introduction to computer organization and data structures*, McGraw-Hill computer science series, McGraw-Hill. (Cited on page 122)

[231] Strassen V (1969) Gaussian Elimination is not Optimal, *Numerical Mathematics*, 13:354–356. (Cited on page 174)

[232] Strassen V (1986) The asymptotic spectrum of tensors and the exponent of matrix multiplication, in: *SFCS '86: Proceedings of the 27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 49–54, Washington, DC, USA: IEEE Computer Society. (Cited on page 174)

[233] Strolenberg R (2007) *ForestFIRE & FIREWood, A Toolkit & GUI for Tree Algorithms*, Master's thesis, Department of Mathematics and Computer Science, Technishe Universiteit Eindhoven, Eindhoven, The Netherlands. (Cited on page 144)

[234] Stumme G (2002) Formal Concept Analysis on Its Way from Mathematics to Computer Science, in: Priss U, Corbett D and Angelova G (Eds.) *Conceptual Structures: Integration and Interfaces*, vol. 2393 of *Lecture Notes in Computer Science*, 2–19, Springer Berlin / Heidelberg. (Cited on page 100)

[235] Tarjan R (1972) Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, 1(2):146–160, URL http://link.aip.org/link/?SMJ/1/146/1. (Cited on page 7)

[236] Tegarden DP (2011) Relation Element Theory Driven Concept Maps, http://www.acis.pamplin.vt.edu/faculty/tegarden/3516/handouts/RET-CMaps.pdf. [Online; accessed 2011-02-28]. (Cited on page 101)

[237] Thomason RH (1969) Species, Determinates and Natural Kinds, *Noûs*, 3(1):95 – 101. (Cited on page 97)

[238] Travers J and Milgram S (1969) An Experimental Study of the Small World Problem, *Sociometry*, 32(4):425–443, URL http://www.jstor.org/stable/2786545. (Cited on pages 9 and 10)

[239] Ullman JD and Yannakakis M (1990) High-probability parallel transitive closure algorithms, in: *SPAA '90: Proceedings of the second annual ACM symposium on Parallel algorithms and architectures*, 200–209, New York, NY, USA: ACM. (Cited on pages 7 and 256)

[240] Valduriez P and Boral H (1986) Evaluation of recursive queries using join indices, in: Charleston SC (Ed.) *Proceedings of the 1st International Conference on Expert Database Systems*, 197–208, Menlo Park, Calif.: Benjamin/Cummings. (Cited on page 7)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[241] Valduriez P and Khoshafian S (1988) Parallel Evaluation of the Transitive Closure of a Database Relation, *International Journal of Parallel Programming*, 17(1):19–42, URL http://0-dx.doi.org.innopac.up.ac.za/10.1007/BF01379321. (Cited on pages 7 and 216)

[242] Valduriez P and Khoshafian S (1989) Transitive Closure of Transitively Closed Relations, in: Kerschberg L (Ed.) *Proceedings from the Second International Conference on Expert Database Systems*, 377–400, Redwood City, Calif.: Benjamin/Cummings Publishing. (Cited on page 7)

[243] van de Rijdt M (2005) *Two-dimensional pattern matching*, Master's thesis, Department of Mathematics and Computer Science, Technishe Universiteit Eindhoven, Eindhoven, The Netherlands. (Cited on pages 94 and 151)

[244] van den Eijnde JPHW (1992) *Program derivation in acyclic graphs and related problems*, Tech. Rep. Computing Science Report 92/04, Technische Universiteit, Eindhoven. (Cited on page 35)

[245] van Gasteren AJ and Dijkstra EW (1990) *On the Shape of Mathematical Arguments*, Lecture Notes in Computer Science, Springer, URL http://books.google.co.za/books?id=VAeV-u8Y3GcC. (Cited on pages 19 and 281)

[246] van Schaik SJ and de Moor O (2011) A Memory Efficient Reachability Data Structure Through Bit Vector Compression, in: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 913–924, New York, NY, USA: ACM, URL http://doi.acm.org/10.1145/1989323.1989419. (Cited on page 8)

[247] Venter F, Kourie DG and Watson BW (2009) FCA-Based Two Dimensional Pattern Matching, in: Ferré S and Rudolph S (Eds.) *Formal Concept Analysis*, vol. 5548 of *Lecture Notes in Computer Science*, 299 – 313, Springer Berlin Heidelberg, URL http://0-dx.doi.org.innopac.up.ac.za/10.1007/978-3-642-01815-2_22. (Cited on pages 19 and 281)

[248] Verdoolaege S, Cohen A and Beletska A (2011) Transitive Closures of Affine Integer Tuple Relations and Their Overapproximations, in: Yahav E (Ed.) *Static Analysis*, vol. 6887 of *Lecture Notes in Computer Science*, 216–232, Springer Berlin Heidelberg, URL http://dx.doi.org/10.1007/978-3-642-23702-7_18. (Cited on page 6)

[249] Wakatani A and Wolfe M (1994) A new approach to array redistribution: Strip mining redistribution, in: Halatsis C, Maritsas D, Philokyprou G and Theodoridis S (Eds.) *PARLE'94 Parallel Architectures and Languages Europe*, vol. 817 of *Lecture Notes in Computer Science*, 323–335, Springer Berlin Heidelberg, URL http://dx.doi.org/10.1007/3-540-58184-7_112. (Cited on page 215)

[250] Walter E (2008) *Cambridge Advanced Learner's Dictionary*, PONS-Wörterbücher, Cambridge University Press, URL http://books.google.co.za/books?id=PDHCFSRmjSMC. (Cited on page 122)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[251] Warren HS Jr (1975) A modification of Warshall's algorithm for the transitive closure of binary relations, *Communications of the ACM*, 18(4):218 – 220. (Cited on pages 7, 187, 188, 216, 222, 232, 233, 256, 273 and 337)

[252] Warshall S (1962) A Theorem on Boolean Matrices, *Journal of the ACM*, 9(1):11–12. (Cited on pages 7, 152, 180, 184, 273 and 329)

[253] Watson BW (1995) *Taxonomies and Toolkits of Regular Language Algorithms*, Ph.D. thesis, Technische Universiteit Eindhoven. (Cited on pages 2, 5, 19, 71, 76, 94, 103, 147, 151 and 281)

[254] Watson BW (2010) *Constructing Minimal Acyclic Deterministic Finite Automata*, Ph.D. thesis, University of Pretoria. (Cited on pages 21, 30, 31, 94, 144 and 282)

[255] Wees D (2012) Mathematical notation is broken, `http://davidwees.com/content/mathematical-notation-broken`. [Online; accessed 27-November-2013]. (Cited on page 281)

[256] Weisstein EW () Interval *From MathWorld*, A Wolfram Web Resource, `http://mathworld.wolfram.com/Interval.html`. [Online; accessed 2012-11-08]. (Cited on page 21)

[257] Weisstein EW (n.d.) Caveman Graph *From MathWorld*, A Wolfram Web Resource, URL `http://mathworld.wolfram.com/CavemanGraph.html`. [Online; accessed 2016-11-11]. (Cited on page 9)

[258] Weisstein EW (n.d.) Matrix Multiplication *From MathWorld*, A Wolfram Web Resource, URL `http://mathworld.wolfram.com/MatrixMultiplication.html`. [Online; accessed 2009-06-15]. (Cited on page 80)

[259] Wikipedia (2008) Systema Naturae — Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/w/index.php?title=Systema_Naturae&oldid=241992965`. [Online; accessed 2008-08-08]. (Cited on pages 97 and 283)

[260] Wikipedia (2008) Taxonomy — Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/w/index.php?title=Taxonomy&oldid=231989736`. [Online; accessed 2008-08-18]. (Cited on page 97)

[261] Wikipedia (2010) Ontology — Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/w/index.php?title=Ontology&oldid=377803633`. [Online; accessed 2010-08-10]. (Cited on page 102)

[262] Wikipedia (2011) Thesaurus — Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/w/index.php?title=Thesaurus&oldid=411745255`. [Online; accessed 2011-02-08]. (Cited on page 100)

[263] Wikipedia (2012) Interval (mathematics) — Wikipedia, The Free Encyclopedia, `http://en.wikipedia.org/w/index.php?title=Interval_(mathematics)&oldid=521776535`. [Online; accessed 2012-11-07]. (Cited on page 21)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

[264] Wikipedia (2015) Transitive closure — Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Transitive_closure&oldid=661007841. [Online; accessed 2015-05-11]. (Cited on page 283)

[265] Wikipedia (2016) Kleene star — Wikipedia, The Free Encyclopedia, URL \url{https://en.wikipedia.org/w/index.php?title=Kleene_star&oldid=740553183}. [Online; accessed 2016-09-21]. (Cited on page 80)

[266] Winston ME, Chaffin R and Herrmann D (1987) A taxonomy of part-whole relations, *Cognitive Science*, 11(4):417 – 444. (Cited on page 101)

[267] Wlodzimierz B, Tomasz K, Marek P and Beletska A (2010) An Iterative Algorithm of Computing the Transitive Closure of a Union of Parameterized Affine Integer Tuple Relations, in: Wu W and Daescu O (Eds.) *Combinatorial Optimization and Applications*, vol. 6508 of *Lecture Notes in Computer Science*, 104–113, Springer Berlin Heidelberg, URL http://dx.doi.org/10.1007/978-3-642-17458-2_10. (Cited on page 8)

[268] Yuan L, Lu Y and Li M (2009) Genetic Algorithm Based on Good Character Breed for Traveling Salesman Problem, in: *Information Science and Engineering (ICISE), 2009 1st International Conference on*, 234 –237. (Cited on page 134)

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Appendices

Appendix A: Symbols

Symbols are used as shorthands for expressions and operations that commonly occur in the text. The meaning of the symbols used in this manuscript is shown in the following tables.

Table A1: Operator Symbols

Symbol	Meaning	Page
$+$	Sum	59, 60
$\times$	Product / Cartestian product	43, 59, 60
$\cup$	Union of sets	43
$\cap$	Intersection of sets	43
$-$	Difference of sets	43
$\wedge$	Boolean 'and'	42
$\vee$	Boolean 'or'	42
$\neg$	Not	41
$\circ$	Composition of relations or functions	44
$\Vert$	Concatenation of matrices	71
$\upharpoonleft$	Left take of a path	71
$\upharpoonright$	Right take of a path	71
$\downharpoonleft$	Left drop of a path	71
$\downharpoonright$	Right drop of a path	71
$\lceil x \rceil$	The smallest integer greater or equal to x	193
$\lfloor x \rfloor$	The largest integer smaller or equal to x	193

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table A2: Symbols for special objects

Symbol	Meaning	Page
$\varnothing$	The empty set, empty relation or *0-tuple*	44, 44, 45
$\mathbb{B}$	The set of Booleans	18
$\mathbb{N}$	The natural numbers	18
$\mathbb{N}^+$	The set of positive integers	18
$\mathbb{N}_n$	The set containing the first n natural numbers	18
R^{-1}	The inverse of a relation R	43
$R<$	Left domain of relation R	43
$R>$	Right domain of relation R	43
R^n	$R \circ R \circ R \ldots R$ (n times)	46
R^+	The transitive closure of relation R	79
R^*	The Kleene closure of relation R	80
E_U	The identity relation on $U \times U$	44
$\perp$	Bottom of a CPO	81
$Sup.S$	Supremum of S	81
$U[n]$	The set of all vectors of size n and data type U	53
$\mathbb{B}[n,n]$	The set of square Boolean matrices of size $n \times n$	55
$\mathbb{O}_n$	The always false matrix $\in \mathbb{B}[n,n]$	55
$\mathbb{J}_n$	The always true matrix $\in \mathbb{B}[n,n]$	55
$\mathbb{I}_n$	The identity matrix $\in \mathbb{B}[n,n]$	55
$\mathbb{I}_A$	$\langle\!\langle\, i,j \mid i,j \in \mathbb{N}_n \mid i = j \wedge i \in A \,\rangle\!\rangle$	55
$\mathbb{I}_{\{k\}}$	$\langle\!\langle\, i,j \mid i,j \in \mathbb{N}_n \mid i = j \wedge j = k \,\rangle\!\rangle$	56
$M_{\mathbb{I}}$	$\langle\!\langle\, i,j \mid i = j \mid m_{ij} \,\rangle\!\rangle$	58
$M[i,\star]$	The ith row in M	57
$M[\star,j]$	The jth column in M	57
M^n	$M \times M \times M \ldots M$ (n times)	60

© University of Pretoria

Table A3: Special functions

Symbol	Meaning	Page
$\|U\|$	The number of elements in set U	42
$\mathcal{P}(X)$	The power set of set X	43
$\Phi(R)$	The square Boolean matrix representation of the relation R	65
$\Psi(M)$	The binary relation represented by the square Boolean matrix M	65
$\mathcal{R}(P)$	The relational equivalent of a vector P	69
$\ell(P)$	The number of elements in $\mathcal{R}(P)$	70
μf	The least fixpoint of f	81
$O(g(n))$	$g(n)$ expresses an upper bound for the complexity of a process	135
$\Omega(g(n))$	$g(n)$ expresses a lower bound for the complexity of a process	135
$\Theta(g(n))$	$g(n)$ expresses both a lower and an upper bound for the complexity of a process	135

Table A4: Quantifier symbols (Page 19-20)

Symbol	Meaning	Quantified operator	Unity element of the operator
$\sum$	Sum of a series	$+$	0
$\prod$	Product of a series	$\times$	1
$\bigcup$	Union of a series	$\cup$	$\varnothing$
$\bigcap$	Intersection of a series	$\cap$	The universal set in the current context
$\exists$	There exists	$\vee$	*false*
$\forall$	For all	$\wedge$	*true*
$\oplus$	Generic quantifier symbol		$1_{\oplus}$

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table A5: Relational Symbols

Symbols without page references are deemed general enough
to use without formally introducing them

Symbol	Meaning	Page
$\in$	Is a member of	
$\notin$	Is not a member of	
$\subset$	Is a proper subset of / is a proper subsequence of	43, 56
$\subseteq$	Is a subset of / is a subsequence of	43
$\not\subseteq$	Is not a subset of / is not a subseqence of	43
$\sqsubset$	Is strictly stronger than	67
$\sqsubseteq$	Is stronger than	67
$\not\sqsubseteq$	Is a not stronger than	67
$\Subset$	Is a sub-path of	70
$\cong$	Is isomorphic to	62
$=$	Is equal to	
$\neq$	Is not equal to	
$\preccurlyeq$	Generic symbol for a partial order	81
$\Longrightarrow$	Implies	
$\Longleftrightarrow$	If and only if	
$\equiv$	Is equivalent to	
$\mapsto$	Maps to	51
$\vdash$	Is a partition of	58
$\lhd$	Is processed before	218
$\rhd$	Is processed after	218

Table A6: Multiplicity Symbols (Chapter 9)

Symbol	Meaning
n	Exactly n
$m \ldots n$	At least m and at most n
$\star$	Zero or more
$1 \cdots \star$	One or more

© University of Pretoria

Appendix B: Acronyms and abbreviations

Acronyms and abbreviations are used to refer to for phrases and entities with long names that commonly occur in writing. The meaning of the acronyms and abbreviations used in this manuscript is shown in the following tables.

Table B1: Institutions and organisations

Abbreviation	Meaning
DARPA	US Defense Advanced Research Projects Agency
NIST	US National Institute of Standards and Technology
TU/e	Eindhoven University of Technology
TABASCO project	Collaboration effort between TU/e and UP to promote TAxonomy BAsed Software COnstruction
UP	University of Pretoria
W3C	World Wide Web Consortium

Table B2: Research fields

Abbreviation	Meaning
AI	Artificial Intelligence
CI	Computational Intelligence
IA	Information Architecture
KO	Knowledge Organisation
LIS	Library and Information Science

Table B3: Authoritative resources

Abbreviation	Name	URL
CALGO	The collected algorithms of the ACM	`calgo.acm.org`
DADS	Dictionary of Algorithms and Data Structures	`www.nist.gov/dads`
Wikipedia	Wikipedia	`www.wikipedia.org`

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table B4: Processes and procedures

Abbreviation	Meaning
DL	Description Logic
FCA	Formal Concept Analysis
FOPL	First order predicate logic
IR	Information retrieval
KBMT	Knowledge-based machine translation
MT	Machine translation
NLP	Natural language processing
SOPL	Second order predicate logic
TABASCO	TAxonomy BAsed Software COnstruction

Table B5: Topic map entities (Page 22 – 31)

Abbreviation	Name
AC	Association type
BN	Base name
DN	Display name
PSI	Published subject indicator
P_i	Player type of the ith player in an association
R_i	Role type of the ith player in an association
SC	Scope identifier
SI	Subject indicator
SN	Sort name
TI	Topic identifier
TL	Topic occurrence locator
TP	Topic occurrence type
TT	Topic type
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VN	Variant name

© University of Pretoria

Table B6: Computational Complexity classes (Section 9.4.3)

Abbreviation	**Meaning**	**Definition**
L	Logarithmic space	Problems that can be solved in logarithmic space.
NC	Nick's Class[3]	Problems that can be solved in $O((\log\ n)^k)$ parallel time, for some fixed integer k, on polynomially many processors.
P	Polynomial	Problems that can be solved by polynomial-time algorithms.
NP	Non-deterministic Polynomial	Problems that can be described using a language that can be decided by a non-deterministic polynomial Turing Machine.
NP-Complete	NP-Complete	A problem in NP is NP-complete if all other problems in NP reduce to it in polynomial time.
NP-Hard	NP-Hard	Problems that are intrinsically harder than NP problems. When a decision version of a combinatorial optimisation problem is proved to belong to the class of NP-complete problems, then the optimisation version is NP-hard.
PSPACE	Polynomial Space	Problems that can be described using a language that can be recognised by a computer using an amount of memory that is bounded by a polynomial in the size of the input.
EXP	Exponential	Problems that can be solved by exponential-time algorithms. These problems are also called intractable.

[1]Stephen Cook coined the name "Nick's class" after Nick Pippenger [14]

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Table B7: Protocols and standards for knowledge management (Page 22)

Abbreviation	Meaning
AsTMa=	Asymptotic Topic Map notation
CTM	Compact Topic Map syntax
HTML	Hypertext Markup Language
LTM	Linear Topic Map notation
OWL	Web Ontology Language
RDF	Resource Description Framework
SQL	Structured Query Language
XML	Extensible Markup Language
XTM	XML Topic Maps

Table B8: Mathematical structures

Abbreviation	Meaning	Page
monotype	Subset of an identity relation	44
point	Monotype with one element	44
PO	Partial order	47
poset	A partially ordered set	81
CPO	Complete partial order	81
DCPO	Directed complete partial order	81

Table B9: Other Acronyms

Abbreviation	Meaning
GCL	Guarded Command Language (Section 2.3)
MADFA	Minimal Acyclic Deterministic Finite Automata (Page 144)
TC	Transitive Closure (Section 1.7)
TCA	TC algorithms i.e. Algorithms solving the TC problem
TM	Topic map (Section 2.2)
TM of TCA	Topic map of algorithms solving the TC problem

© University of Pretoria

Appendix C: LTM Listings of the TM of TCA

Listing C1: The author of this topic map (TM)

```
[VP : Author]
{VP, FullName, "Vreda Pieterse"}
{VP, Profile, "http://www.cs.up.ac.za/cs/vpieterse/"}
{VP, Profile, "http://www.linkedin.com/pub/vreda-pieterse/38/62/58b"}
{VP, Profile, "http://dl.acm.org/author_page.cfm?id=81100481696"}
{VP, Profile, "http://www.researchgate.net/profile/Vreda_Pieterse"}
{VP, Profile, "http://scholar.google.co.za
     /citations?user=CeJSo8oAAAAJ&hl=en"}
```

Problem areas and problems

Listing C2: The problem area of the transitive closure problem

```
[Combinatorial : Area ="Combinatorial problem"
   @"http://en.wikipedia.org/wiki/Category:Combinatorial_algorithms" ]
[GraphProblem : Combinatorial ="Graph problem"
   @"http://en.wikipedia.org/wiki/Category:Graph_algorithms"]
```

Listing C3: The problem area of the matrix multiplication problem

```
[NumAnalysis : Area ="Numerical analysis"
   @"http://en.wikipedia.org/wiki/Category:Numerical_analysis" ]
[NumLinAlgebra : NumAnalysis ="Numerical Linear Algebra"
   @"http://en.wikipedia.org/wiki/Category:Numerical_linear_algebra" ]
```

Listing C4: The matrix multiplication problem

```
[MatrixMultiplication : NumLinAlgebra Problem ="Matrix Multiplication"
   @"http://www.nist.gov/dads/HTML/matrixMultiplication.html" ]
{MatrixMultiplication, Author, VP}
{MatrixMultiplication, Date, [[ 2013-09-09 ]]}
{MatrixMultiplication, ComplexityClass, P}
{MatrixMultiplication, Discussion,
     "http://en.wikipedia.org/wiki/Matrix_multiplication" }
{MatrixMultiplication, Discussion,
     "http://www.cs.sunysb.edu/~algorith/files/
     matrix-multiplication.shtml" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C5: The transitive closure problem

```
[TCProblem : GraphProblem Problem ="Transitive Closure"
    @"http://www.nist.gov/dads/HTML/transitiveClosure.html"]
{TCProblem, Author, VP}
{TCProblem, Date, [[ 2013−03−21 ]]}
{TCProblem, ComplexityClass, P}
{TCProblem, ComplexityClass, NC}
{TCProblem, Precondition, [[ R ⊆ U × U ]]}
{TCProblem, Postcondition, [[ THIS−Reference: Section 6.1.1: Page 79 ]]}
{TCProblem, Discussion,
    [[ THIS−Reference: Section 1.7: Page 8 ]]}
{TCProblem, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html"}
{TCProblem, Discussion,
    "http://en.wikipedia.org/w/index.php?title=Transitive_closure"}
{TCProblem, Discussion,
    "http://mathworld.wolfram.com/TransitiveClosure.html"}
{TCProblem, Visualisation,
    [[ THIS−Reference: Figure 8.1.1: Page 108 ]]}
{TCProblem, Visualisation,
    "http://www.cs.sunysb.edu/~algorith/files/transitive−closure.shtml"}
{TCProblem, Visualisation,
    "http://anh.cs.luc.edu/363/notes/09dynProg.html"}
{TCProblem, Visualisation,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html"}
belongsTo (TCProblem : Problem, GraphProblem : Area)
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Data Structures

Listing C6: Boolean matrix and its ascendants

```
[Matrix : DataStructure ="Matrix"
   @"http://www.nist.gov/dads/HTML/matrix.html" ]
[UniformMatrix : Matrix ="Uniform Matrix"
   @"http://www.nist.gov/dads/HTML/uniformmatrx.html" ]
[SquareMatrix : UniformMatrix ="Square Matrix"
   @"http://www.nist.gov/dads/HTML/squarematrix.html" ]
[BooleanMatrix : SquareMatrix ="Boolean Matrix"
   @"http://www.nist.gov/dads/HTML/adjacencyMatrixRep.html" ]
[BooleanMatrix
   ("logical matrix" /alsoKnownAs) ("(0,1) matrix" /alsoKnownAs)
   ("binary matrix" /alsoKnownAs) ("relation matrix" /alsoKnownAs)
   ("adjacency matrix" /alsoKnownAs) ]
{BooleanMatrix, Discussion,
    "http://en.wikipedia.org/w/index.php
    ?title=Logical_matrix&oldid=566595505"}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Techniques

Listing C7: Basic algorithmic techniques applied by TC algorithms

```
[Coat : Technique ="Coat"]
{Coat, Definition,
    [[Construct the TC of a relation by coating the relation with additional
      edges until it is transitive]]}
[Grow : Technique ="Grow"]
{Grow, Definition,
    [[ Construct the TC of R by growing the argument of a function f
    (that ultimately maps U to the TC of R) from the empty set to U ]]}
```

Listing C8: General algorithmic techniques

```
[Natural : Technique ="Natural processing order" ]
{Natrural, Description,
    [[ Perform operations for which a non−deterministic order is specified,
      in a natural sequential order ]]}
[UpperBound : Technique ="Safe upper bound" ]
{UpperBound, Description, [[ Iterate a calculated number of times. ]]}
[ChMonitor : Technique ="Change monitor" ]
{ChMonitor, Description,
    [[ Keep track of changes during loop execution and stop iterating when a
     complete pass during which no changes was made has occurred. ]]}
[InterchangeLoops : Technique ="Loop interchange" ]
{InterchangeLoops, Description,
    [[ The variable used in the inner loop switches to the outer loop,
    and vice versa ]]}
[FuseLoops : Technique ="Loop fusion" ]
{FuseLoops, Description,
    [[ Combine two adjacent loops to share the same loop counters. ]]}
[TileLoops : Technique ="Loop tiling" ]
{TileLoops, Description,
    [[ Partition operations into blocks, so that multiple operations on the same
    elements can be performed while they remain in the cache ]]}
[ShortCircuit : Technique ="Short circuit" ]
{ShortCircuit, Description,
    [[ Return the final outcome of the operation without completing all the
    steps required to determine the outcome, if it is known that the outcome
    will not change should the ommitted operations be performed. ]]}
```

© University of Pretoria

Listing C9: Algorithmic techniques specific to TC Algorithm optimisation

```
[RowMonitor : Technique ="Row state monitor" ]
{RowMonitor, Description,
    [[ Keep track of the state of the rows in the matrix. Three states are set:
     Unknown, empty and full. An empty row need not be processed or added.
     A full row need not be processed. Rows with unknown state have to be
     processed and added. ]]}
```

Listing C10: Tiling strategies applied by implementations of the tiling algorithm

```
[Trivial : TileLoops Technique ="Trivial partition"]
{Trivial, Definition,
    [[ Specify a partition of a matrix consisting of one submatrix which is the
    entire matrix. process the entries in column major order ]]}
[Diagonal : TileLoops Technique ="Diagonal partition"]
{Diagonal, Definition,
    [[ Create a partition of the matrix using the main diagonal as the boundry
    between the submatrices. Process the entries in these submatrices
    in row order. ]]}
[kRowDiagonal : TileLoops Technique ="k−Row Diagonal partition"]
{kRowDiagonal, Definition,
    [[ Create a partition of the matrix consisting of blocks with k rows in each
     block. Split each of these blocks into two submatrices using the main
     diagonal as the boundry between the submatrices. Process the entries
     in these submatrices in column order. ]]}
[kCol : TileLoops Technique ="k−Col Triplet partition"]
{kCol, Definition,
    [[ Create a partition of the matrix consisting of blocks with k columns in
    each block. Split each of these blocks into three submatrices; a square
    submatrix where the block overlaps the main diagonal; one submatrix
    above this square submatix; and one submatrix below it. Process the
    middle submatrix in column order. Process the entries in the other two
    submatrices in row order. ]]}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C11: Algorithmic techniques applied when operating on matrices

```
[M-Op : Technique ="Matrix operations"]
{M-Op, Definition, [[ Applies operations on a matrix ]]}
[MatAdd : PrimAlg M-Op ="Matrix addition" ]
{MatAdd, Definition, [[ Addition of matrices ]]}
{MatAdd, Description, [[ Add two equally sized matrices ]]}
{MatAdd, Discussion, "http://en.wikipedia.org/wiki/Matrix_addition"}
[MatMpy : PrimAlg M-Op ="Matrix multiplication"]
{MatMpy, Definition, [[ Multiplication of matrices ]]}
{MatMpy, Description, [[ Multiply two equally sized square matrices ]]}
{MatMpy, Discussion,
     "http://en.wikipedia.org/wiki/Matrix_multiplication"}
uses (MatMpy : Algorithm, SquareMatrix : DataStructure)
solves (MatMpy: Algorithm, MatrixMultiplication: Problem)
{MatMpy, Discussion, [[ THIS—Reference: Section 12.3: Page 174 ]]}
```

© University of Pretoria

Algorithms

Listing C12: The root algorithm solving the TC problem

```
[TCRoot : PrimAlg ="TC Root Algorithm"]
solves (TCRoot : Algorithm, TCProblem : Problem)
{TCRoot, Author, VP}
{TCRoot, Date, [[ 2013−02−14 ]]}
{TCRoot, Description,
    [[ The root algorithm solving the TC problem ]]}
{TCRoot, Specification,
    [[ THIS−Reference: Algorithm 11.2.2 (Root): Page 163 ]]}
{TCRoot, Verification,
    [[ The algorithm is trivially correct because it is the definition of TC. ]]}
```

Listing C13: The root coat algorithm

```
[CoatAlg : PrimAlg TCRoot ="Root coat algorithm"]
{CoatAlg, Description,
    [[Algorithm calculating R+ by coating R]]}
solves (CoatAlg : Algorithm, TCProblem : Problem)
applies (CoatAlg : Algorithm, Coat : Technique)
{CoatAlg, Precondition,
    [[ n = |R| ∈ ℕ+ ]]}
{CoatAlg, Specification,
    [[ THIS−Reference: Algorithm 11.3.1 (Coat): Page 165 ]]}
{CoatAlg, Verification,
    [[ THIS−Reference: Section 11.3.2: Page 166 ]]}
```

Listing C14: The root grow alorithm

```
[GrowAlg : TCRoot PrimAlg ="Root grow algorithm"]
{GrowAlg, Description,
   [[Algorithm calculating R+ by growing the argument of a
  continious function that maps U to R+]]}
solves (GrowAlg : Algorithm, TCProblem : Problem)
applies (GrowAlg : Algorithm, Grow : Technique)
{GrowAlg, Specification,
   [[ THIS−Reference: Algorithm 11.4.1 (Grow): Page 168 ]]}
{GrowAlg, Verification,
   [[ THIS−Reference: Section 6.5.2: Page 86 ]]}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C15: Matrix coat implementation

```
[CoatM-Op : CompAlg CoatAlg ="Coat using matrix operations" ]
{CoatM-Op, Description,
   [[Derivation of the coat algorithm. It uses a square boolean
     matrix to store and manupulate relations ]]}
{CoatM-Op, SubAlgorithm, BoolMatrixMultipy}
{CoatM-Op, Precondition, [[ n = |U| ∈ ℕ+ ]]}
solves (CoatM-Op : Algorithm, TCProblem : Problem)
uses (CoatM-Op : Algorithm, BooleanMatrix : DataStructure)
applies (CoatM-Op : Algorithm, Coat : Technique)
applies (CoatM-Op : Algorithm, M-Op : Technique)
{CoatM-Op, Specification,
    [[ THIS−Reference: Algorithm 12.2.2 (MatrixCoat): Page 174 ]]}
{CoatM-Op, Verification,
    [[ THIS−Reference: Section 12.2.2: Page 173 ]]}
```

Listing C16: Matrix grow implementation

```
[GrowM-Op : Algorithm ="Grow using matrix operations" ]
[GrowM-Op : GrowAlg TCRoot]
{GrowM-Op, Description,
   [[ Derivation of the root grow algorithm. It uses a square boolean
     matrix to store and manupulate relations ]]}
{GrowM-Op, Precondition, [[ n = |U| ∈ ℕ+ ]]}
solves (GrowM-Op : Algorithm, TCProblem : Problem)
uses (GrowM-Op : Algorithm, BooleanMatrix : DataStructure)
applies (GrowM-Op : Algorithm, Grow : Technique)
applies (GrowM-Op : Algorithm, M-Op : Technique)
{GrowM-Op, Specification,
    [[ THIS−Reference: Algorithm 12.5.1 (MatrixGrow): Page 179 ]]}
{GrowM-Op, Verification,
    [[ THIS−Reference: Section 12.5.2: Page 180 ]]}
```

© University of Pretoria

Listing C17: Prosser's algorithm

```
[Prosser : CompAlg CoatM-Op ="Prosser"]
{Prosser, SubAlgorithm, BoolMatrixMultiply}
{Prosser, Description,
    [[Derivation of the coat algorithm using matrix operations. It stops at a
     mathematically determined safe upper bound ]]}
solves (Prosser : Algorithm, TCProblem : Problem)
uses (Prosser : Algorithm, BooleanMatrix : DataStructure)
applies (Prosser : Algorithm, Coat : Technique)
applies (Prosser : Algorithm, M-Op : Technique)
applies (Prosser : Algorithm, UpperBound : Technique)
{Prosser, Author, VP }
{Prosser, Date, [[ 2013-07-17 ]]}
{Prosser, Publication,
    "http://citeseerx.ist.psu.edu/showciting?cid=5016384" }
{Prosser, Publication, [[ THIS-Citation: Prosser [203] ]]}
{Prosser, Discussion,
    [[ THIS-Reference: Section 12.4: Page 175 ]]}
{Prosser, Complexity, ProsserTimeComp}
{Prosser, Specification,
    [[ THIS-Reference: Algorithm 12.4.1 (Prosser): Page 175 ]]}
{Prosser, Verification,
    [[ THIS-Reference: Section 12.4.3: Page 177 ]]}
{Prosser, Implementation, ProsserImpStd}
{Prosser, Implementation, ProsserImpBoost}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C18: Short circuited version of Prosser's algorithm

```
[ProsserShort : Prosser PrimAlg
     ="ProsserShort";;"Short circuited version of Prosser"]
{ ProsserShort, Description,
   [[ Short circuited version of the neat coat algorithm ]]}
solves (ProsserShort : Algorithm, TCProblem : Problem)
uses (ProsserShort : Algorithm, BooleanMatrix : DataStructure)
applies (ProsserShort : Algorithm, Coat : Technique)
applies (ProsserShort : Algorithm, M-Op : Technique)
applies (ProsserShort : Algorithm, ShortCircuit : Technique)
{ ProsserShort, Author, VP }
{ ProsserShort, Date, [[ 2015-01-07 ]]}
{ ProsserShort, Verification,
    [[ THIS-Reference: Section 18.3.1: Page 263 ]]}
{ ProsserShort, Complexity, ProsserTimeComp }
{ ProsserShort, Discussion,
    [[ THIS-Reference: Section 18.3 : Page 262 ]]}
{ ProsserShort, Specification,
    [[ THIS-Reference: Algorithm 18.3.1 (ShortProsser): Page 262 ]]}
{ ProsserShort, Implementation, ProsserShortImpStd }
{ ProsserShort, Implementation, ProsserShortImpBoost }
```

© University of Pretoria

Listing C19: Warshall's algorithm

```
[Warshall : GrowM-Op] /* Parent in the derivation tree */
[Warshall : PrimAlg ="Warshall";"warshall";"Warshall's TC algorithm"
     ("Warshall−Floyed" /alsoKnownAs)
     ("Algorithm 96: Ancestor" /alsoKnownAs)]
solves (Warshall : Algorithm, TCProblem : Problem)
uses (Warshall : Algorithm, BooleanMatrix : DataStructure)
applies (Warshall : Algorithm, Grow : Technique)
applies (Warshall : Algorithm, M-Op : Technique)
applies (Warshall : Algorithm, Natural : Technique)
{Warshall, Description,
   [[ Derivation of the grow algorithm using matrix operations. It iterates
     through the entries in the matrix in column order ]]}
{Warshall, Author, VP }
{Warshall, Date, [[ 2013−07−17 ]]}
{Warshall, Publication,
    "http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.7172" }
{Warshall, Publication, [[ THIS−Citation: Warshall [252] ]]}
{Warshall, Publication, [[ THIS−Citation: Floyd [92] ]]}
{Warshall, Complexity, WarshallTimeComp}
{Warshall, Specification,
     [[ THIS−Reference: Algorithm 12.6.1 (Warshall) : Page 181 ]]}
{Warshall, Verification,
   [[ THIS−Reference: Section 12.6.4: Page 183 ]]}
{Warshall, Discussion,
     "http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/
     LN250_Tremblay/L17−Warshall.htm" }
{Warshall, Discussion, [[ THIS−Reference: Section 12.6 : Page 180 ]]}
{Warshall, Visualisation,
     "http://ww2.cs.mu.oz.au/aia/demoindex.html#TransClosure" }
{Warshall, Visualisation, [[ THIS−Reference: Figure 12.6.2 : Page 182 ]]}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C20: Short circuited version of Warshall's algorithm

```
[WarshallShort : Warshall PrimAlg
     ="WarshallShort";;
     "Short circuited version of Warshall's algorithm to solve TC"]
{ WarshallShort, Description,
   [[ Short circuited version of Warshall's algorithm ]]}
solves ( WarshallShort : Algorithm, TCProblem : Problem)
uses ( WarshallShort : Algorithm, BooleanMatrix : DataStructure)
applies ( WarshallShort : Algorithm, Grow : Technique)
applies ( WarshallShort : Algorithm, M-Op : Technique)
applies ( WarshallShort : Algorithm, Natural : Technique)
applies ( WarshallShort : Algorithm, ShortCircuit : Technique)
{ WarshallShort, Author, VP }
{ WarshallShort, Date, [[ 2015-01-07 ]]}
{ WarshallShort, Verification,
    [[ THIS-Reference: Section 18.6.1: Page 269 ]]}
{ WarshallShort, Complexity, WarshallTimeComp }
{ WarshallShort, Discussion,
    [[ THIS-Reference: Section 18.6 : Page 267 ]]}
{Warshall, Specification,
    [[ THIS-Reference: Algorithm 18.6.1 (ShortWarshall) : Page 268 ]]}
{ WarshallShort, Implementation, WarshallShortImpStd }
{ WarshallShort, Implementation, WarshallShortImpBoost }
```

© University of Pretoria

Listing C21: Fused coat algorithm

```
[CoatFuse : PrimAlg CoatM-Op ="Fused coat algorithm"]
{CoatFuse, Description,
   [[ Derivation of the coat algorithm that uses matrix operations and a
     safe upper bound. The optimisation technique to fuse adjacent loops
     is applied. ]]}
solves (CoatFuse : Algorithm, TCProblem : Problem)
uses (CoatFuse : Algorithm, BooleanMatrix : DataStructure)
applies (CoatFuse : Algorithm, Coat : Technique)
applies (CoatFuse : Algorithm, M-Op : Technique)
applies (CoatFuse : Algorithm, UpperBound : Technique)
applies (CoatFuse : Algorithm, FuseLoops : Technique)
{CoatFuse, Author, VP }
{CoatFuse, Date, [[ 2013-12-17 ]]}
{CoatFuse, Discussion,
    [[ THIS-Reference: Section 15.1: Page 205 ]]}
{CoatFuse, Complexity, CoatFuseComp}
{CoatFuse, Specification,
    [[ THIS-Reference: Algorithm 15.1.1 (CoatFuse): Page 207 ]]}
{CoatFuse, Verification,
    [[ THIS-Reference: Section 15.1.1: Page 207 ]]}
{CoatFuse, Implementation, CoatFuseImpStd}
{CoatFuse, Implementation, CoatFuseImpBoost}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C22: Naïve grow implementation in row order

```
[NaiveGrowRow : GrowM-Op PrimAlg
    ="Grow using matrix operations in row order" ]
{NaiveGrowRow, Description,
   [[ Derivation of Warshall's algorithm by applying loop interchange.
      It iterates through the entries in the matrix in row order ]]}
solves (NaiveGrowRow : Algorithm, TCProblem : Problem)
uses (NaiveGrowRow : Algorithm, BooleanMatrix : DataStructure)
applies (NaiveGrowRow : Algorithm, Grow : Technique)
applies (NaiveGrowRow : Algorithm, M-Op : Technique)
applies (NaiveGrowRow : Algorithm, InterchangeLoops : Technique)
{NaiveGrowRow, Author, VP}
{NaiveGrowRow, Date, [[ 2013-06-02 ]]}
{NaiveGrowRow, Specification,
    [[ THIS-Reference: Algorithm 13.1.3 (GrowRow) : Page 190 ]]}
{NaiveGrowRow, Verification,
    [[ THIS-Reference: Section 13.1.3: Page 189 ]]}
{NaiveGrowRow, Visualisation,
    [[ THIS-Reference: Figure 13.1.1 : Page 188 ]]}
```

Listing C23: Skeleton tile algorithm

```
[TileSkeleton : GrowM-Op CompAlg
    ="Skeleton tiling algorithm" ]
{TileSkeleton, SubAlgorithm, Tile}
{TileSkeleton, Description,
   [[ Derivation of the grow algorithm using matrix operations. It iterates
      through the entries in the matrix in an order determined by a specified
      loop tiling strategy ]]}
solves (TileSkeleton : Algorithm, TCProblem : Problem)
uses (TileSkeleton : Algorithm, BooleanMatrix : DataStructure)
applies (TileSkeleton : Algorithm, Grow : Technique)
applies (TileSkeleton : Algorithm, M-Op : Technique)
applies (TileSkeleton : Algorithm, TileLoops : Technique)
{TileSkeleton, Author, VP}
{TileSkeleton, Date, [[ 2014-01-07 ]]}
{TileSkeleton, Specification,
   [[ THIS-Reference: Algorithm 16.4.1 (TileSkeleton) : Page 229 ]]}
{TileSkeleton, Verification,
   [[ THIS-Reference: Section 16.4.1: Page 227 ]]}
```

© University of Pretoria

Listing C24: Martynyuk's algorithm

```
[Martynyuk : NaiveGrowRow PrimAlg
     ="Martynyuk";;"Martynyuk's algorithm to solve TC"]
{Martynyuk, Description,
   [[ Derivation of the grow algorithm using matrix operations. It iterates
    through the entries in the matrix in row order. It stops at a mathematically
    determined safe upper bound]]}
solves (Martynyuk : Algorithm, TCProblem : Problem)
uses (Martynyuk : Algorithm, BooleanMatrix : DataStructure)
applies (Martynyuk : Algorithm, Grow : Technique)
applies (Martynyuk : Algorithm, M-Op : Technique)
applies (Martynyuk : Algorithm, InterchangeLoops : Technique)
applies (Martynyuk : Algorithm, UpperBound : Technique)
{Martynyuk, Author, VP }
{Martynyuk, Date, [[ 2013−07−17 ]]}
{Martynyuk, Publication, [[ THIS−Citation: Martynyuk [164] ]]}
{Martynyuk, Specification,
   [[ THIS−Reference: Algorithm 13.2.1 (Martynyuk) : Page 193 ]]}
{Martynyuk, Verification, [[ THIS−Reference: Section 13.2.2: Page 193 ]]}
{Martynyuk, Complexity, MartynyukTimeComp }
{Martynyuk, Discussion, [[ THIS−Reference: Section 13.2 : Page 191 ]]}
{Martynyuk, Implementation, MartynyukImpStd }
{Martynyuk, Implementation, MartynyukImpBoost }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C25: Baker's algorithm

```
[Baker : NaiveGrowRow PrimAlg
    ="Baker";;"Baker's algorithm to solve TC"]
{Baker, Description,
   [[Derivation of the grow algorithm using matrix operations. It iterates
     through the entries in the matrix in row order. It uses a change monitor
     and stops when a pass with no changes has occurred. ]]}
solves (Baker : Algorithm, TCProblem : Problem)
uses (Baker : Algorithm, BooleanMatrix : DataStructure)
applies (Baker : Algorithm, Grow : Technique)
applies (Baker : Algorithm, M-Op : Technique)
applies (Baker : Algorithm, InterchangeLoops : Technique)
applies (Baker : Algorithm, ChMonitor : Technique)
{Baker, Author, VP }
{Baker, Date, [[ 2013-07-17 ]]}
{Baker, Publication,
    "http://citeseerx.ist.psu.edu/showciting?cid=634265" }
{Baker, Publication, [[ THIS-Citation: Baker [17] ]]}
{Baker, Complexity, BakerTimeComp }
{Baker, Verification, [[ THIS-Reference: Section 14.1.2: Page 199 ]]}
{Baker, Specification,
    [[ THIS-Reference: Algorithm 14.1.1 (Baker) : Page 199 ]]}
{Baker, Discussion,
    [[ THIS-Reference: Section 14.1 : Page 197 ]]}
{Baker, Implementation, BakerImpStd }
{Baker, Implementation, BakerImpBoost
```

© University of Pretoria

Listing C26: Monitored coat algorithm

```
[CoatCh : CompAlg CoatM-Op ="Monitored coat algorithm"]
{CoatCh, SubAlgorithm, BoolMatrixMultiply}
{CoatCh, Description,
    [[Derivation of the coat algorithm using matrix operations. It uses a
    change monitor and stops when a pass with no changes has occurred. ]]}
solves (CoatCh : Algorithm, TCProblem : Problem)
uses (CoatCh : Algorithm, BooleanMatrix : DataStructure)
applies (CoatCh : Algorithm, Coat : Technique)
applies (CoatCh : Algorithm, M-Op : Technique)
applies (CoatCh : Algorithm, ChMonitor : Technique)
{CoatCh, Author, VP }
{CoatCh, Date, [[ 2013-09-13 ]]}
{CoatCh, Discussion,
     [[ THIS-Reference: Section 14.2: Page 201 ]]}
{CoatCh, Complexity, CoatChTimeComp}
{CoatCh, Specification,
     [[ THIS-Reference: Algorithm 14.2.1 (CoatMonitor): Page 203 ]]}
{CoatCh, Verification,
     [[ THIS-Reference: Section 14.2.2: Page 202 ]]}
{CoatCh, Implementation, CoatChImpStd}
{CoatCh, Implementation, CoatChImpBoost}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C27: Neat coat algorithm

```
[CoatNeat : PrimAlg CoatCh ="Neat coat algorithm"]
{CoatNeat, Description,
   [[ Derivation of the monitored coat algorithm. It fuses adjacent loops. ]]}
solves (CoatNeat : Algorithm, TCProblem : Problem)
uses (CoatNeat : Algorithm, BooleanMatrix : DataStructure)
applies (CoatNeat : Algorithm, Coat : Technique)
applies (CoatNeat : Algorithm, M-Op : Technique)
applies (CoatNeat : Algorithm, ChMonitor : Technique)
applies (CoatNeat : Algorithm, FuseLoops : Technique)
{CoatNeat, Author, VP }
{CoatNeat, Date, [[ 2013-12-31 ]]}
{CoatNeat, Discussion,
     [[ THIS-Reference: Section 15.2: Page 208 ]]}
{CoatNeat, Complexity, CoatNeatTimeComp}
{CoatNeat, Specification,
     [[ THIS-Reference: Algorithm 15.2.1 (CoatNeat): Page 210 ]]}
{CoatNeat, Verification,
     [[ THIS-Reference: Section 15.2.2: Page 211 ]]}
{CoatNeat, Implementation, CoatNeatImpStd}
{CoatNeat, Implementation, CoatNeatImpBoost}
```

© University of Pretoria

Listing C28: Warren's algorithm

```
[Warren : PrimAlg ="Warren";"Warren";"Warren's TC algorithm"]
solves (Warren : Algorithm, TCProblem : Problem)
uses (Warren : Algorithm, BooleanMatrix : DataStructure)
applies (Warren : Algorithm, Grow : Technique)
applies (Warren : Algorithm, M-Op : Technique)
applies (Warren : Algorithm, TileLoops : Technique)
applies (Warren : Algorithm, Diagonal : Technique)
{Warren, Description,
   [[ A tiling algorithm applying the diagonal tiling strategy. ]]}
{Warren, Author, VP }
{Warren, Date, [[ 2014-01-02 ]]}
{Warren, Publication,
    "http://citeseerx.ist.psu.edu/showciting?cid=605208" }
{Warren, Publication, [[ THIS-Citation: Warren [251] ]]}
{Warren, Complexity, WarrenTimeComp }
{Warren, Specification,
   [[ THIS-Reference: Algorithm 17.1.1 (Warren) : Page 235 ]]}
{Warren, Verification, [[ THIS-Reference: Section 17.2: Page 236 ]]}
{Warren, Discussion, [[ THIS-Reference: Section 17.1 : Page 233 ]]}
{Warren, Implementation, WarrenImpStd }
{Warren, Implementation, WarrenImpBoost }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C29: Agrawal's blocked row algorithm

```
[BlockRow : TileSkeleton CompAlg
    ="BlockRow";;"Blocked Row algorithm to solve TC"]
{BlockRow, Description,
   [[ Rows are processed in blocks while applying loop tiling to minimise cache
   thrashing. The elements within a block are processed in column order ]]}
solves (BlockRow : Algorithm, TCProblem : Problem)
uses (BlockRow : Algorithm, BooleanMatrix : DataStructure)
applies (BlockRow : Algorithm, Grow : Technique)
applies (BlockRow : Algorithm, M-Op : Technique)
applies (BlockRow : Algorithm, LoopTiling : Technique)
applies (BlockRow : Algorithm, kRowDiagonal : Technique)
{BlockRow, Author, VP }
{BlockRow, Date, [[ 2014-01-26 ]]}
{BlockRow, Publication, [[ THIS-Citation: Agrawal and Jagadish [2] ]]}
{BlockRow, Publication, [[ THIS-Citation: Agrawal et al. [1] ]]}
{BlockRow, Complexity, BlockRowTimeComp }
{BlockRow, Verification,
   [[ THIS-Reference: Section 17.4: Page 243 ]]}
{BlockRow, Specification,
   [[ THIS-Reference: Algorithm 17.3.1 (BlockRow) : Page 240 ]]}
{BlockRow, Discussion,
   [[ THIS-Reference: Section 17.3 : Page 238 ]]}
{BlockRow, Implementation, BlockRowImpStd }
{BlockRow, Implementation, BlockRowImpBoost
```

© University of Pretoria

Listing C30: Agrawal's blocked column algorithm

```
[BlockCol : TileSkeleton CompAlg
     ="BlockCol";;"Blocked Column algorithm to solve TC"]
{BlockCol, Description,
   [[ Columns are processed in blocks while applying loop tiling to minimise
   cache thrashing. Each block is divided in three partiotions. A square middle
   partition overlapping the main diagonal of the matrix and two rectangular
   partitions repectively above and below this square partition. The elements
   within middle partition are processed in column order. The elements in the
   other two partitions are processed in row order. ]]}
solves (BlockCol : Algorithm, TCProblem : Problem)
uses (BlockCol : Algorithm, BooleanMatrix : DataStructure)
applies (BlockCol : Algorithm, Grow : Technique)
applies (BlockCol : Algorithm, M-Op : Technique)
applies (BlockCol : Algorithm, LoopTiling : Technique)
applies (BlockCol : Algorithm, kCol : Technique)
{BlockCol, Author, VP }
{BlockCol, Date, [[ 2014-04-21 ]]}
{BlockCol, Publication, [[ THIS-Citation: Agrawal and Jagadish [2] ]]}
{BlockCol, Publication, [[ THIS-Citation: Agrawal et al. [1] ]]}
{BlockCol, Complexity, BlockColTimeComp }
{BlockCol, Verification,
    [[ THIS-Reference: Section 17.6: Page 253 ]]}
{BlockCol, Specification,
    [[ THIS-Reference: Algorithm 17.5.3 (BlockCol) : Page 250 ]]}
{BlockCol, Discussion,
    [[ THIS-Reference: Section 17.5 : Page 247 ]]}
{BlockCol, Implementation, BlockColImpStd }
{BlockCol, Implementation, BlockColImpBoost
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Complexities

Listing C31: The complexity of Prosser's algorithm

```
[ProsserTimeComp : Complexity
    ="ProsserTimeComp";;"Complexity of Prosser's algorithm"
{ProsserTimeComp, Author, VP}
{ProsserTimeComp, Date, [[ 2013-07-30 ]]}
{ProsserTimeComp, Resource, [[ Time ]]}
{ProsserTimeComp, Specification, [[ Θ(n^4) ]]}
{ProsserTimeComp, Justification,
    [[ THIS-Reference: Section 12.4.4: Page 178 ]]}
```

Listing C32: The complexity of Warshall's algorithm

```
[WarshallTimeComp : Complexity]
{WarshallTimeComp, Author, VP }
{WarshallTimeComp, Date, [[ 2013-07-17 ]]}
{WarshallTimeComp, Resource, [[Time ]]}
{WarshallTimeComp, Specification, [[ Θ(n^3) ]]}
{WarshallTimeComp, Justification,
    [[ THIS-Reference: Section 12.6.5: Page 183 ]]}
```

Listing C33: The complexity of Martynyuk's algorithm

```
[MartynyukTimeComp : Complexity]
{MartynyukTimeComp, Author, VP }
{MartynyukTimeComp, Date, [[ 2013-07-22 ]]}
{MartynyukTimeComp, Resource, [[Time ]]}
{MartynyukTimeComp, Specification, [[ Θ(n^3 log n) ]]}
{MartynyukTimeComp, Justification,
    [[ THIS-Reference: Section 13.2.3: Page 195 ]]}
```

Listing C34: The complexity of Baker's algorithm

```
[BakerTimeComp : Complexity]
{BakerTimeComp, Author, VP }
{BakerTimeComp, Date, [[ 2013-07-22 ]]}
{BakerTimeComp, Resource, [[Time ]]}
{BakerTimeComp, Specification, [[ O(n^3 log n) ]]}
{BakerTimeComp, Justification,
    [[ THIS-Reference: Section 14.1.3: Page 200 ]]}
```

© University of Pretoria

Listing C35: The complexity of the fused coat algorithm

```
[CoatFuseComp : Complexity
    ="CoatFuseComp";"complexity fused coat";
      "The complexity of the fused coat algorithm"
{CoatFuseComp, Author, VP}
{CoatFuseComp, Date, [[ 2013−12−17 ]]}
{CoatFuseComp, Resource, [[ Time ]]}
{CoatFuseComp, Specification, [[ Θ(n^4) ]]}
{CoatFuseComp, Justification,
     [[ THIS−Reference: Section 15.1.2: Page 207 ]]}
```

Listing C36: The complexity of the monitored coat algorithm

```
[CoatChTimeComp : Complexity
    ="CoatChTimeComp";"complexity monitored coat worst";
      "The complexity of the monitored coat algorithm"
{CoatChTimeComp, Author, VP}
{CoatChTimeComp, Date, [[ 2013−09−13 ]]}
{CoatChTimeComp, Resource, [[ Time ]]}
{CoatChTimeComp, Specification, [[ O(n^4) ]]}
{CoatChTimeComp, Justification,
     [[ THIS−Reference: Section 14.2.3: Page 203 ]]}
```

Listing C37: The complexity of the neat coat algorithm

```
[CoatNeatTimeComp : Complexity
    ="CoatNeatTimeComp";"complexity neat coat worst";
      "The complexity of the Neat coat algorithm"
{CoatNeatTimeComp, Author, VP}
{CoatNeatTimeComp, Date, [[ 2013−12−17 ]]}
{CoatNeatTimeComp, Resource, [[ Time ]]}
{CoatNeatTimeComp, Specification, [[ O(n^4) ]]}
{CoatNeatTimeComp, Justification,
     [[ THIS−Reference: Section 15.2.3: Page 212 ]]}
```

Listing C38: The complexity of Warren's algorithm

```
[WarrenTimeComp : Complexity]
{WarrenTimeComp, Author, VP }
{WarrenTimeComp, Date, [[ 2014−01−02 ]]}
{WarrenTimeComp, Resource, [[Time ]]}
{WarrenTimeComp, Specification, [[ Θ(n^3) ]]}
{WarrenTimeComp, Justification,
     [[ THIS−Reference: Section 17.1.2: Page 235 ]]}
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C39: The complexities of Agrawal's blocked row algorithm

```
[BlockRowTimeComp : Complexity]
{BlockRowTimeComp, Author, VP }
{BlockRowTimeComp, Date, [[ 2014−01−26]]}
{BlockRowTimeComp, Resource, [[Time ]]}
{BlockRowTimeComp, Specification, [[ Ω(n^3) ]]}
{BlockRowTimeComp, Specification, [[ O(n^4) ]]}
{BlockRowTimeComp, Justification,
     [[ THIS−Reference: Section 17.3.2: Page 242 ]]}
```

Listing C40: The complexities of Agrawal's blocked column algorithm

```
[BlockColTimeComp : Complexity]
{BlockColTimeComp, Author, VP }
{BlockColTimeComp, Date, [[ 2014−04−21]]}
{BlockColTimeComp, Resource, [[Time ]]}
{BlockColTimeComp, Specification, [[ Ω(n^3) ]]}
{BlockColTimeComp, Specification, [[ O(n^4) ]]}
{BlockColTimeComp, Justification,
    [[ THIS−Reference: Section 17.5.4: Page 251 ]]}
```

© University of Pretoria

Archives

Listing C41: Input data suitable to use for several TC implementations

```
[TCTestData : SupportFile ="TC Test Data"]
{TCTestData, Author, VP }
{TCTestData, Date, [[ 2013-12-08 ]]}
{TCTestData, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{TCTestData, Size, [[ 1.4 MB ]]}
{TCTestData, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    InputData.zip" }
```

Listing C42: Implementations of Prosser's algorithm

```
/** Using a vector of char* */
[ProsserImpStd : SourceCode ="Prosser standard implementation"]
{ProsserImpStd, Author, VP }
{ProsserImpStd, Date, [[ 2013-09-01 ]]}
{ProsserImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{ProsserImpStd, Size, [[ 1.3 KB ]]}
{ProsserImpStd, Support, TCTestData}
{ProsserImpStd, Language, [[ C++ ]]}
{ProsserImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ProsserStd.zip" }

/** Using a vector of dynamic_bitset<> */
[ProsserImpBoost : SourceCode
    ="Prosser implementation with boost/dynamic_bitset"]
{ProsserImpBoost, Author, VP }
{ProsserImpBoost, Date, [[ 2013-09-01 ]]}
{ProsserImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{ProsserImpBoost, Size, [[ 1.3 KB]]}
{ProsserImpBoost, Support, TCTestData}
{ProsserImpBoost, Support, "http://www.boost.org/"}
{ProsserImpBoost, Language, [[ C++ ]]}
{ProsserImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ProsserBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C43: Short circuited implementations of Prosser's algorithm

```
/** Short Cricuited version using a vector of char* */
[ProsserShortImpStd : SourceCode
    ="Prosser standard implementation with short circuiting"]
{ProsserShortImpStd, Author, VP }
{ProsserShortImpStd, Date, [[ 2013−09−01 ]]}
{ProsserShortImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{ProsserShortImpStd, Size, [[ 1.3 KB ]]}
{ProsserShortImpStd, Support, TCTestData}
{ProsserShortImpStd, Language, [[ C++ ]]}
{ProsserShortImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ShortProsserStd.zip" }

/** Short Cricuited version using a vector of dynamic_bitset<> */
[ProsserShortImpBoost : SourceCode
    ="Prosser implementation with short circuiting
    using boost/dynamic_bitset"]
{ProsserShortImpBoost, Author, VP }
{ProsserShortImpBoost, Date, [[ 2015−01−08 ]]}
{ProsserShortImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{ProsserShortImpBoost, Size, [[ 1612 B]]}
{ProsserShortImpBoost, Support, TCTestData}
{ProsserShortImpBoost, Support, "http://www.boost.org/"}
{ProsserShortImpBoost, Language, [[ C++ ]]}
{ProsserShortImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ShortProsserBoost.zip" }
```

© University of Pretoria

Listing C44: Implementations of Warshall's algorithm

```
/** Using a vector of char* */
[WarshallImpStd : SourceCode
    ="Warshall standard implementation"]
{WarshallImpStd, Author, VP }
{WarshallImpStd, Date, [[ 2013−09−01 ]]}
{WarshallImpStd, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarshallImpStd, Size, [[ 967 B ]]}
{WarshallImpStd, Support, TCTestData}
{WarshallImpStd, Language, [[ C++ ]]}
{WarshallImpStd, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     WarshallStd.zip" }

/** Using a vector of dynamic_bitset<> */
[WarshallImpBoost : SourceCode
     ="Warshall implementation with boost/dynamic_bitset"]
{WarshallImpBoost, Author, VP }
{WarshallImpBoost, Date, [[ 2013−09−01 ]]}
{WarshallImpBoost, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarshallImpBoost, Size, [[ 931 B]]}
{WarshallImpBoost, Support, TCTestData}
{WarshallImpBoost, Support, "http://www.boost.org/"}
{WarshallImpBoost, Language, [[ C++ ]]}
{WarshallImpBoost, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     WarshallBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C45: Short circuited implementations of Warshall's algorithm

```
/** Using a vector of char* */
[WarshallShortImpStd : SourceCode
    ="Short circuited version of the standard implementation
      of Warshall's algorithm"]
{WarshallShortImpStd, Author, VP }
{WarshallShortImpStd, Date, [[ 2015-01-08 ]]}
{WarshallShortImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarshallShortImpStd, Size, [[ 1265 B ]]}
{WarshallShortImpStd, Support, TCTestData}
{WarshallShortImpStd, Language, [[ C++ ]]}
{WarshallShortImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ShortWarshallStd.zip" }

/** Using a vector of dynamic_bitset<> */
[WarshallShortImpBoost : SourceCode
    ="Short circuited version of the implementation of Warshall's algorithm
      using boost/dynamic_bitset"]
{WarshallShortImpBoost, Author, VP }
{WarshallShortImpBoost, Date, [[ 2015-01-08 ]]}
{WarshallShortImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarshallShortImpBoost, Size, [[ 1201 B]]}
{WarshallShortImpBoost, Support, TCTestData}
{WarshallShortImpBoost, Support, "http://www.boost.org/"}
{WarshallShortImpBoost, Language, [[ C++ ]]}
{WarshallShortImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ShortWarshallBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C46: Implementations of Martynyuk's algorithm

```
/** Using a vector of char* */
[MartynyukImpStd : SourceCode
     ="Martynyuk standard implementation"]
{MartynyukImpStd, Author, VP }
{MartynyukImpStd, Date, [[ 2013-09-01 ]]}
{MartynyukImpStd, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{MartynyukImpStd, Size, [[ 1009 B ]]}
{MartynyukImpStd, Support, TCTestData}
{MartynyukImpStd, Language, [[ C++ ]]}
{MartynyukImpStd, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     MartynyukStd.zip" }

/** Using a vector of dynamic_bitset<> */
[MartynyukImpBoost : SourceCode
     ="Martynyuk implementation with boost/dynamic_bitset"]
{MartynyukImpBoost, Author, VP }
{MartynyukImpBoost, Date, [[ 2013-09-01 ]]}
{MartynyukImpBoost, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{MartynyukImpBoost, Size, [[ 1004 B ]]}
{MartynyukImpBoost, Support, TCTestData}
{MartynyukImpBoost, Support, "http://www.boost.org/"}
{MartynyukImpBoost, Language, [[ C++ ]]}
{MartynyukImpBoost, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     MartynyukBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C47: Short circuited implementations of Martynyuk's algorithm

```
/** Using a vector of char* */
[MartynyukShortImpStd : SourceCode
    ="Short circuited version of the standard implementation of
      Martynyuk's algorthm" ]
{MartynyukShortImpStd, Author, VP }
{MartynyukShortImpStd, Date, [[ 2015-01-08 ]]}
{MartynyukShortImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{MartynyukShortImpStd, Size, [[ 1236 B ]]}
{MartynyukShortImpStd, Support, TCTestData}
{MartynyukShortImpStd, Language, [[ C++ ]]}
{MartynyukShortImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ShortMartynyukStd.zip" }

/** Using a vector of dynamic_bitset<> */
[MartynyukShortImpBoost : SourceCode
    ="Short circuited version of the implementation of
      Martynyuk's algorthm using boost/dynamic_bitset"]
{MartynyukShortImpBoost, Author, VP }
{MartynyukShortImpBoost, Date, [[ 2015-01-08 ]]}
{MartynyukShortImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{MartynyukShortImpBoost, Size, [[ 1263 B ]]}
{MartynyukShortImpBoost, Support, TCTestData}
{MartynyukShortImpBoost, Support, "http://www.boost.org/"}
{MartynyukShortImpBoost, Language, [[ C++ ]]}
{MartynyukShortImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    ShortMartynyukBoost.zip" }
```

© University of Pretoria

Listing C48: Implementations of Baker's algorithm

```
/** Using a vector of char* */
[BakerImpStd : SourceCode ="Baker standard implementation"]
{BakerImpStd, Author, VP }
{BakerImpStd, Date, [[ 2013−09−01 ]]}
{BakerImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BakerImpStd, Size, [[ 1010 B ]]}
{BakerImpStd, Support, TCTestData}
{BakerImpStd, Language, [[ C++ ]]}
{BakerImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    BakerStd.zip" }

/** Using a vector of dynamic_bitset<> */
[BakerImpBoost : SourceCode
    ="Baker implementation with boost/dynamic_bitset"]
{BakerImpBoost, Author, VP }
{BakerImpBoost, Date, [[ 2013−09−01 ]]}
{BakerImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BakerImpBoost, Size, [[ 993 B ]]}
{BakerImpBoost, Support, TCTestData}
{BakerImpBoost, Support, "http://www.boost.org/"}
{BakerImpBoost, Language, [[ C++ ]]}
{BakerImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    BakerBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C49: Short circuited implementations of Baker's algorithm

```
/** Using a vector of char* */
[BakerShortImpStd : SourceCode
    ="Short circuited version of the standard implementation of
        Baker's algorithm"]
{BakerShortImpStd, Author, VP }
{BakerShortImpStd, Date, [[ 2015-01-08 ]]}
{BakerShortImpStd, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BakerShortImpStd, Size, [[ 1213 B ]]}
{BakerShortImpStd, Support, TCTestData}
{BakerShortImpStd, Language, [[ C++ ]]}
{BakerShortImpStd, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     ShortBakerStd.zip" }

/** Using a vector of dynamic_bitset<> */
[BakerShortImpBoost : SourceCode
     ="Short circuited version of the implementation of
        Baker's algorithm using boost/dynamic_bitset"]
{BakerShortImpBoost, Author, VP }
{BakerShortImpBoost, Date, [[ 2015-01-08 ]]}
{BakerShortImpBoost, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BakerShortImpBoost, Size, [[ 1267 B ]]}
{BakerShortImpBoost, Support, TCTestData}
{BakerShortImpBoost, Support, "http://www.boost.org/"}
{BakerShortImpBoost, Language, [[ C++ ]]}
{BakerShortImpBoost, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     ShortBakerBoost.zip" }
```

© University of Pretoria

Listing C50: Implementations of the monitored coat algorithm

```
/** Using a vector of char* */
[CoatChImpStd : SourceCode
    ="Standard implementation of the monitored coat algorithm"]
{CoatChImpStd, Author, VP }
{CoatChImpStd, Date, [[ 2013-09-03 ]]}
{CoatChImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatChImpStd, Size, [[ 1.4 KB ]]}
{CoatChImpStd, Support, TCTestData}
{CoatChImpStd, Language, [[ C++ ]]}
{CoatChImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    MonCoatStd.zip" }

/** Using a vector of dynamic_bitset<> */
[CoatChImpBoost : SourceCode
    ="Implementation of the monitored coat algorithm
using boost/dynamic_bitset"]
{CoatChImpBoost, Author, VP }
{CoatChImpBoost, Date, [[ 2013-09-13 ]]}
{CoatChImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatChImpBoost, Size, [[ 1.4 KB ]]}
{CoatChImpBoost, Support, TCTestData}
{CoatChImpBoost, Support, "http://www.boost.org/"}
{CoatChImpBoost, Language, [[ C++ ]]}
{CoatChImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    MonCoatBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C51: Short circuited implementations of the monitored coat algorithm

```
/** Using a vector of char* */
[CoatChShortImpStd : SourceCode
      ="Short circuited version of the standard implementation of
          the monitored coat algorithm"]
{CoatChShortImpStd, Author, VP }
{CoatChShortImpStd, Date, [[ 2015-01-08 ]]}
{CoatChShortImpStd, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatChShortImpStd, Size, [[ 1663 KB ]]}
{CoatChShortImpStd, Support, TCTestData}
{CoatChShortImpStd, Language, [[ C++ ]]}
{CoatChShortImpStd, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortMonCoatStd.zip" }

/** Using a vector of dynamic_bitset<> */
[CoatChShortImpBoost : SourceCode
      ="Short circuited version of the implementation of the
          monitored coat algorithm using boost/dynamic_bitset"]
{CoatChShortImpBoost, Author, VP }
{CoatChShortImpBoost, Date, [[ 2015-01-08 ]]}
{CoatChShortImpBoost, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatChShortImpBoost, Size, [[ 1693 B ]]}
{CoatChShortImpBoost, Support, TCTestData}
{CoatChShortImpBoost, Support, "http://www.boost.org/"}
{CoatChShortImpBoost, Language, [[ C++ ]]}
{CoatChShortImpBoost, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortMonCoatBoost.zip" }
```

© University of Pretoria

Listing C52: Implementations of the fused coat algorithm

```
/** Using a vector of char* */
[CoatFuseImpStd : SourceCode
    ="Standard implementation of the fused coat algorithm"]
{CoatFuseImpStd, Author, VP }
{CoatFuseImpStd, Date, [[ 2013-12-17 ]]}
{CoatFuseImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatFuseImpStd, Size, [[ 1.4 KB ]]}
{CoatFuseImpStd, Support, TCTestData}
{CoatFuseImpStd, Language, [[ C++ ]]}
{CoatFuseImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    FusedCoatStd.zip" }

/** Using a vector of dynamic_bitset<> */
[CoatFuseImpBoost : SourceCode
    ="Implementation of the fused coat algorithm
using boost/dynamic_bitset"]
{CoatFuseImpBoost, Author, VP }
{CoatFuseImpBoost, Date, [[ 2013-12-17 ]]}
{CoatFuseImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatFuseImpBoost, Size, [[ 1.4 KB ]]}
{CoatFuseImpBoost, Support, TCTestData}
{CoatFuseImpBoost, Support, "http://www.boost.org/"}
{CoatFuseImpBoost, Language, [[ C++ ]]}
{CoatFuseImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    FusedCoatBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C53: Short circuited implementations of the fused coat algorithm

```
/** Using a vector of char* */
[CoatFuseShortImpStd : SourceCode
     ="Short circuited version of the standard implementation of
          the fused coat algorithm"]
{CoatFuseShortImpStd, Author, VP }
{CoatFuseShortImpStd, Date, [[ 2015-01-08 ]]}
{CoatFuseShortImpStd, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatFuseShortImpStd, Size, [[ 4697 B ]]}
{CoatFuseShortImpStd, Support, TCTestData}
{CoatFuseShortImpStd, Language, [[ C++ ]]}
{CoatFuseShortImpStd, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     ShortFusedCoatStd.zip" }

/** Using a vector of dynamic_bitset<> */
[CoatFuseShortImpBoost : SourceCode
     ="Short circuited version of the implementation of the
          fused coat algorithm using boost/dynamic_bitset"]
{CoatFuseShortImpBoost, Author, VP }
{CoatFuseShortImpBoost, Date, [[ 2015-01-08 ]]}
{CoatFuseShortImpBoost, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatFuseShortImpBoost, Size, [[ 4687 B ]]}
{CoatFuseShortImpBoost, Support, TCTestData}
{CoatFuseShortImpBoost, Support, "http://www.boost.org/"}
{CoatFuseShortImpBoost, Language, [[ C++ ]]}
{CoatFuseShortImpBoost, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     ShortFusedCoatBoost.zip" }
```

© University of Pretoria

Listing C54: Implementations of the neat coat algorithm

```
/** Using a vector of char* */
[CoatNeatImpStd : SourceCode
    ="Standard implementation of the neat coat algorithm"]
{CoatNeatImpStd, Author, VP }
{CoatNeatImpStd, Date, [[ 2013-12-31 ]]}
{CoatNeatImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatNeatImpStd, Size, [[ 1.4 KB ]]}
{CoatNeatImpStd, Support, TCTestData}
{CoatNeatImpStd, Language, [[ C++ ]]}
{CoatNeatImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    NeatCoatStd.zip" }

/** Using a vector of dynamic_bitset<> */
[CoatNeatImpBoost : SourceCode
    ="Implementation of the neat coat algorithm
using boost/dynamic_bitset"]
{CoatNeatImpBoost, Author, VP }
{CoatNeatImpBoost, Date, [[ 2013-12-31 ]]}
{CoatNeatImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatNeatImpBoost, Size, [[ 1.4 KB ]]}
{CoatNeatImpBoost, Support, TCTestData}
{CoatNeatImpBoost, Support, "http://www.boost.org/"}
{CoatNeatImpBoost, Language, [[ C++ ]]}
{CoatNeatImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    NeatCoatBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C55: Short circuited implementations of the neat coat algorithm

```
/** Using a vector of char* */
[CoatNeatShortImpStd : SourceCode
      ="Short circuited version of the standard implementation of
          the neat coat algorithm"]
{CoatNeatShortImpStd, Author, VP }
{CoatNeatShortImpStd, Date, [[ 2015-01-08 ]]}
{CoatNeatShortImpStd, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatNeatShortImpStd, Size, [[ 1687 B ]]}
{CoatNeatShortImpStd, Support, TCTestData}
{CoatNeatShortImpStd, Language, [[ C++ ]]}
{CoatNeatShortImpStd, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortNeatCoatStd.zip" }

/** Using a vector of dynamic_bitset<> */
[CoatNeatShortImpBoost : SourceCode
      ="Short circuited version of the implementation of
          the neat coat algorithm using boost/dynamic_bitset"]
{CoatNeatShortImpBoost, Author, VP }
{CoatNeatShortImpBoost, Date, [[ 2015-01-08 ]]}
{CoatNeatShortImpBoost, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{CoatNeatShortImpBoost, Size, [[ 4842 B]]}
{CoatNeatShortImpBoost, Support, TCTestData}
{CoatNeatShortImpBoost, Support, "http://www.boost.org/"}
{CoatNeatShortImpBoost, Language, [[ C++ ]]}
{CoatNeatShortImpBoost, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortNeatCoatBoost.zip" }
```

© University of Pretoria

Listing C56: Implementations of Warren's algorithm

```
/** Using a vector of char* */
[WarrenImpStd : SourceCode
    ="Standard implementation of Warren's algorithm"]
{WarrenImpStd, Author, VP }
{WarrenImpStd, Date, [[ 2014−01−08 ]]}
{WarrenImpStd, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarrenImpStd, Size, [[ 2014 B ]]}
{WarrenImpStd, Support, TCTestData}
{WarrenImpStd, Language, [[ C++ ]]}
{WarrenImpStd, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    WarrenStd.zip" }

/** Using a vector of dynamic_bitset<> */
[WarrenImpBoost : SourceCode
    ="Implementation of Warren's algorithm
using boost/dynamic_bitset"]
{WarrenImpBoost, Author, VP }
{WarrenImpBoost, Date, [[ 2014−01−08 ]]}
{WarrenImpBoost, Discussion,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarrenImpBoost, Size, [[ 970 B ]]}
{WarrenImpBoost, Support, TCTestData}
{WarrenImpBoost, Support, "http://www.boost.org/"}
{WarrenImpBoost, Language, [[ C++ ]]}
{WarrenImpBoost, File,
    "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
    WarrenBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C57: Short circuited implementations of Warren's algorithm

```
/** Using a vector of char* */
[WarrenShortImpStd : SourceCode
     ="Short circuited version of the standard implementation
        of Warren's algorithm"]
{WarrenShortImpStd, Author, VP }
{WarrenShortImpStd, Date, [[ 2015-01-08 ]]}
{WarrenShortImpStd, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarrenShortImpStd, Size, [[ 1260 B ]]}
{WarrenShortImpStd, Support, TCTestData}
{WarrenShortImpStd, Language, [[ C++ ]]}
{WarrenShortImpStd, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     ShortWarrenStd.zip" }

/** Using a vector of dynamic_bitset<> */
[WarrenShortImpBoost : SourceCode
     ="Short circuited version of the implementation of Warren's algorithm
        using boost/dynamic_bitset"]
{WarrenShortImpBoost, Author, VP }
{WarrenShortImpBoost, Date, [[ 2015-01-08 ]]}
{WarrenShortImpBoost, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{WarrenShortImpBoost, Size, [[ 1227 B ]]}
{WarrenShortImpBoost, Support, TCTestData}
{WarrenShortImpBoost, Support, "http://www.boost.org/"}
{WarrenShortImpBoost, Language, [[ C++ ]]}
{WarrenShortImpBoost, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     ShortWarrenBoost.zip" }
```

© University of Pretoria

Listing C58: Implementations of Agrawal's block row algorithm

```
/** Using a vector of char* */
[BlockRowImpStd : SourceCode
     ="Standard implementation of Agrawal's blocked row algorithm"]
{BlockRowImpStd, Author, VP }
{BlockRowImpStd, Date, [[ 2014-01-26 ]]}
{BlockRowImpStd, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockRowImpStd, Size, [[ 1256 B ]]}
{BlockRowImpStd, Support, TCTestData}
{BlockRowImpStd, Language, [[ C++ ]]}
{BlockRowImpStd, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     BlockRowStd.zip" }

/** Using a vector of dynamic_bitset<> */
[BlockRowImpBoost : SourceCode
     ="Implementation of Agrawal's blocked row algorithm
        using boost/dynamic_bitset"]
{BlockRowImpBoost, Author, VP }
{BlockRowImpBoost, Date, [[ 2014-01-26 ]]}
{BlockRowImpBoost, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockRowImpBoost, Size, [[ 1331 B ]]}
{BlockRowImpBoost, Support, TCTestData}
{BlockRowImpBoost, Support, "http://www.boost.org/"}
{BlockRowImpBoost, Language, [[ C++ ]]}
{BlockRowImpBoost, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     BlockRowBoost.zip" }
```

© University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C59: Short circuited implementations of Agrawal's block row algorithm

```
/** Using a vector of char* */
[BlockRowShortImpStd : SourceCode
      ="Short circuited version of the standard implementation of
          Agrawal's blocked row algorithm"]
{BlockRowShortImpStd, Author, VP }
{BlockRowShortImpStd, Date, [[ 2014-10-22 ]]}
{BlockRowShortImpStd, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockRowShortImpStd, Size, [[ 1581 B ]]}
{BlockRowShortImpStd, Support, TCTestData}
{BlockRowShortImpStd, Language, [[ C++ ]]}
{BlockRowShortImpStd, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortBlockRowStd.zip" }

/** Using a vector of dynamic_bitset<> */
[BlockRowShortImpBoost : SourceCode
      ="Short circuited version of the implementation of
          Agrawal's blocked row algorithm
using boost/dynamic_bitset"]
{BlockRowShortImpBoost, Author, VP }
{BlockRowShortImpBoost, Date, [[ 2015-01-08 ]]}
{BlockRowShortImpBoost, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockRowShortImpBoost, Size, [[ 1615 B ]]}
{BlockRowShortImpBoost, Support, TCTestData}
{BlockRowShortImpBoost, Support, "http://www.boost.org/"}
{BlockRowShortImpBoost, Language, [[ C++ ]]}
{BlockRowShortImpBoost, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortBlockRowBoost.zip" }
```

© University of Pretoria

Listing C60: Implementations of Agrawal's block column algorithm

```
/** Using a vector of char* */
[BlockColImpStd : SourceCode
     ="Standard implementation of Agrawal's blocked column algorithm"]
{BlockColImpStd, Author, VP }
{BlockColImpStd, Date, [[ 2014-04-21 ]]}
{BlockColImpStd, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockColImpStd, Size, [[ 1303 B ]]}
{BlockColImpStd, Support, TCTestData}
{BlockColImpStd, Language, [[ C++ ]]}
{BlockColImpStd, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     BlockColStd.zip" }

/** Using a vector of dynamic_bitset<> */
[BlockColImpBoost : SourceCode
     ="Implementation of Agrawal's blocked column algorithm
        using boost/dynamic_bitset"]
{BlockColImpBoost, Author, VP }
{BlockColImpBoost, Date, [[ 2014-04-21 ]]}
{BlockColImpBoost, Discussion,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockColImpBoost, Size, [[ 1458 B ]]}
{BlockColImpBoost, Support, TCTestData}
{BlockColImpBoost, Support, "http://www.boost.org/"}
{BlockColImpBoost, Language, [[ C++ ]]}
{BlockColImpBoost, File,
     "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
     BlockColBoost.zip" }
```

© University of Pretoria

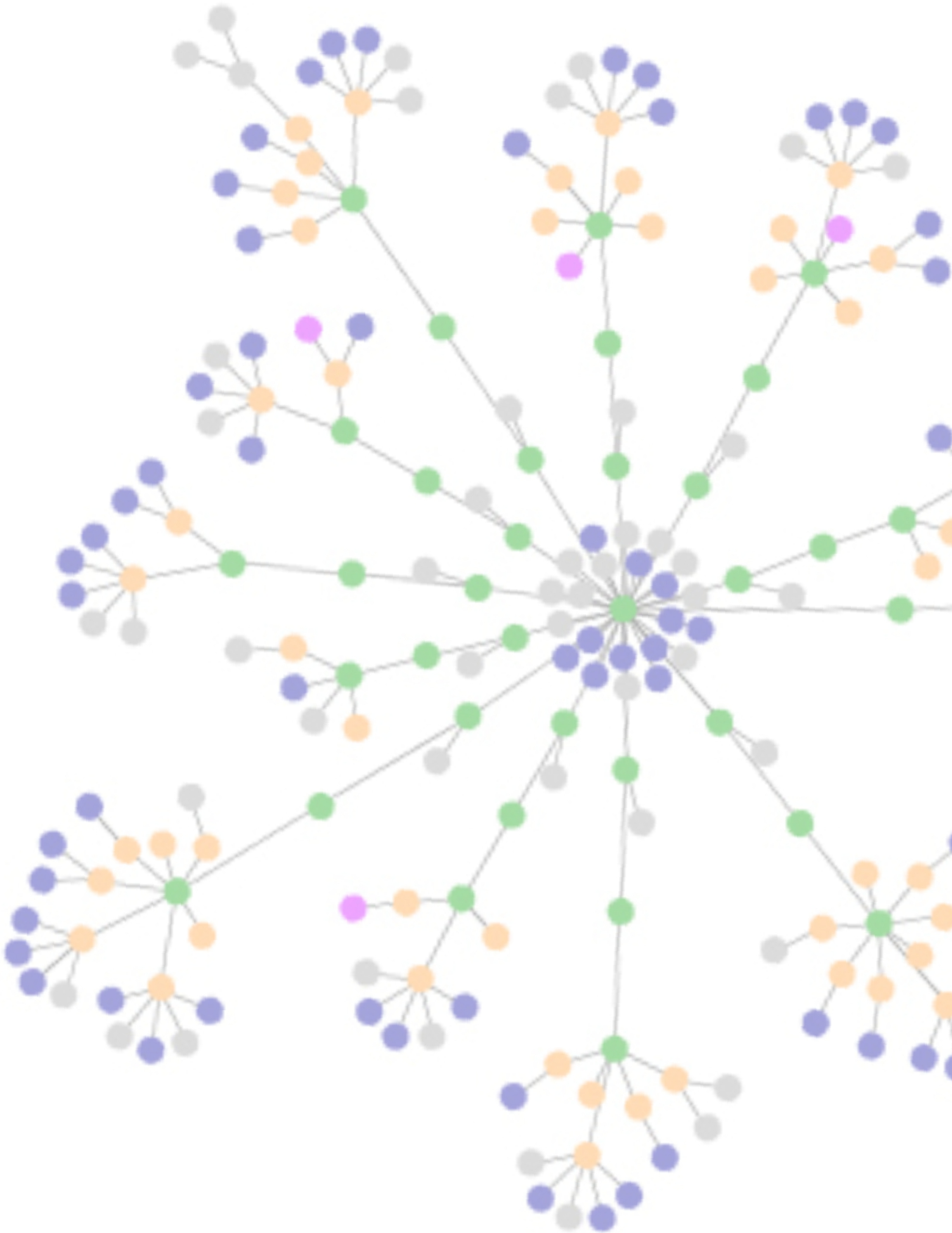
UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Listing C61: Short circuited implementations of Agrawal's block column algorithm

```
/** Using a vector of char* */
[BlockColShortImpStd : SourceCode
      ="Short circuited version of the standard implementation of
          Agrawal's blocked column algorithm"]
{BlockColShortImpStd, Author, VP }
{BlockColShortImpStd, Date, [[ 2015−01−08 ]]}
{BlockColShortImpStd, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockColShortImpStd, Size, [[ 1668 B ]]}
{BlockColShortImpStd, Support, TCTestData}
{BlockColShortImpStd, Language, [[ C++ ]]}
{BlockColShortImpStd, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortBlockColStd.zip" }

/** Using a vector of dynamic_bitset<> */
[BlockColShortImpBoost : SourceCode
      ="Short circuited version of the implementation of Agrawal's
          blocked column algorithm using boost/dynamic_bitset"]
{BlockColShortImpBoost, Author, VP }
{BlockColShortImpBoost, Date, [[ 2015−01−08 ]]}
{BlockColShortImpBoost, Discussion,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/AptiAlgoritmi.html" }
{BlockColShortImpBoost, Size, [[ 1800 B ]]}
{BlockColShortImpBoost, Support, TCTestData}
{BlockColShortImpBoost, Support, "http://www.boost.org/"}
{BlockColShortImpBoost, Language, [[ C++ ]]}
{BlockColShortImpBoost, File,
      "http://www.cs.up.ac.za/cs/vpieterse/AptiAlgo/Archives/
      ShortBlockColBoost.zip" }
```

© University of Pretoria



© University of Pretoria