



UNIVERSIDADE DA CORUÑA
DEPARTAMENTO DE COMPUTACIÓN

Formal Extension of the Relational Model for the Management of Spatial and Spatio-temporal Data

Tese Doutoral

Doutorando: José Ramón Ríos Viqueira
Director: Dr. Nikos A. Lorentzos
Titora: Dra. Nieves Rodríguez Brisaboa

A Coruña, Xullo de 2003

Ph. D. Thesis supervised by
Tese doutoral dirixida por

Dr. Nikos A. Lorentzos
Informatics Laboratory
Science Department
Agricultural University of Athens
Iera Odos 75, GR 118 55 Athens Greece
Telf: +(01) 529 4175
Fax: +(01) 529 4199
lorentzos@aua.gr

Tutor
Titora

Dra. Nieves Rodríguez Brisaboa
Departamento de Computación
Facultade de Informática
Universidade da Coruña
15071 A Coruña (España)
Telf. +34 981 16 70 00 Ext. 1243
Fax: +34 981 16 71 60
brisaboa@udc.es

Acknowledgements

I would like to thank my supervisor Dr. Nikos A. Lorentzos for his support, dedication and patience during all these years of hard work. I do appreciate very much his high degree of professionalism but, beyond that, I am especially thankful for his being such human.

I would also like to thank my tutor in Spain Dr. Nieves Rodríguez Brisaboa, for her support and advice in both research and, more generally, in life.

I am grateful to my fellows in the Laboratory of Databases at the Department of Computer Science of the University of A Coruña, Fariña, Miguel Luaces, Ángeles, Tony, Fran, José Paramá, Eva, Miguel Penabad and mon. They have made my life much easy while working at the University A Coruña. I am also much grateful to all the staff and fellows in the Informatics Laboratory of the Agricultural University of Athens, Professor Alex Sideridis, Director of the Laboratory, Thodoros, Costas, Ntina, Spyros, Costas Dondis, Yiannis, Marios, Bader, Fani, Christina, Dimitris and Thanasis, for their help and kindness during my stay in Athens.

Many thanks also go to my family, my parents Daniel and Clarisa, to my sister Cristina, to my brother Jorge (alive in our heart), to my grandmother Balbina and to my aunt M^a Carmen. Thank you for your care and your patience.

Thank you Belen for your love and support in good and bad moments. Thank you for your patience and understanding during my long periods of absence.

Thanks to all my friends at Kypseli, Athens, S & K Omadara, Kostas, Stefanos, Vassilis, Andonis, Eirini, Panayiotis, Marios, Andreas, all those with whom I had the pleasure to enjoy so many good moments. Many thanks also go to Mr Panayiotis and Mrs Fani. I shall not forget their hospitality and those delicious Sunday lunches. Thanks also to Victor for those unforgettable games of tennis.

I would finally like to thank all the organizations that supported the work of this thesis:

The European Union that funded my work via project “*CHOROCHRONOS: A Research Network for Spatiotemporal Database Systems*”, A TMR Research Network Project N^o ERB FMRX-CT96-0056, during the period October 1998 – July 2000.

The University of A Coruña, which funded my research during the periods January 2001 – October 2001 and January 2002 – October 2002.

Agradecementos

Gustaríame agradecer ó meu director de tese Dr. Nikos A. Lorentzos o seu apoio, dedicación e paciencia durante todos estes anos de traballo duro. É realmente de apreciar o seu alto grado de profesionalidade, pero sobre todo, estou altamente agradecido pola súa grande cualidade humana.

Tamén me gustaría agradecer á miña titora en España Dra. Nieves Rodríguez Brisaboa o seu apoio e bos consellos, tanto no traballo como na vida en xeral.

Estoulles altamente agradecido ós meus compañeiros do Laboratorio de Bases de Datos do Departamento de Computación da Universidade da Coruña, Fariña, Miguel Luaces, Ángeles, Tony, Fran, José Paramá, Eva, Miguel Penabad e mon. Eles conseguiron que a miña vida no traballo na Universidade da Coruña fose moito máis sinxela. Tamén lles estou agradecido ó persoal e compañeiros do Informatics Laboratory da Agricultural University of Athens, Profesor Alex Sideridis, Director do laboratorio, Thodoros, Costas, Ntina, Spyros, Costas Dondis, Yiannis, Marios, Bader, Fani, Christina, Dimitris e Thanasis, pola súa axuda e amabilidade durante as miñas estancias en Atenas.

Moitas gracias tamén á miña familia, ós meus pais, Daniel e Clarisa, á miña irmá Cristina, ó meu irmán Jorge (vivo no noso corazón), á miña avoa Balbina e á miña tía M^a Carmen. Gracias polos vosos coidados e paciencia.

Gracias Belén polo teu amor e apoio nos bos e malos momentos. Gracias pola túa paciencia e comprensión durante os meus largos períodos de ausencia.

Gracias a todos os meu amigos de Kypseli, Atenas, S & K Omadara, Kostas, Stefanos, Vassilis, Andonis, Eirini, Panayiotis, Marios, Andreas, todos aqueles cos que tiven o pracer de desfrutar tantos bos momentos. Moitas gracias tamén para o Sr. Panayiotis e a Sra. Fani. Nunca esqueceré a súa hospitalidade e aquelas deliciosas comidas de domingo. Gracias tamén a Víctor por aqueles inesquecibles partidos de tenis.

Por último, gustaríame agradecer a todas as organizacións que apoiaron o traballo desta tese:

Á Union Europea que financiou o meu traballo a través do proxecto “*CHOROCHRONOS: A Research Network for Spatiotemporal Database Systems*”, A TMR Research Network Project N^o ERB FMRX-CT96-0056, durante o período Outubro 1998 – Xullo 2000.

Á Universidade da Coruña, que financiou o meu traballo investigador durante os períodos Xaneiro 2001 – Outubro 2001 e Xaneiro 2002 – Outubro 2002.

A Belén e Jorge

Contents

1. INTRODUCTION	1
1.1 BACKGROUND.....	1
1.2 CONTRIBUTIONS OF THIS THESIS.....	4
1.3 OUTLINE OF THESIS.....	5
2. PREVIOUS WORK AND MOTIVATION	7
2.1 INTRODUCTION.....	7
2.2 PREVIOUS RESEARCH ON SPATIAL APPROACHES.....	8
2.2.1 ROSE Algebra.....	8
2.2.2 Tomlin's Map Algebra (TMA).....	10
2.2.3 Erwig and Schneider's Spatial Partition Model (ESSPM).....	11
2.2.4 Hadzilacos and Tryfona's GIS Approach (HTGIS).....	13
2.2.5 ESRI ArcInfo 8.....	14
2.2.6 Intergraph Geomedia 5.....	15
2.2.7 MapInfo Professional 7.....	15
2.2.8 Bentley Microstation Geographics 7.2.....	15
2.2.9 Grid Based Commercial GIS Tools (GRIDGIS).....	16
2.2.10 Geosabrina.....	17
2.2.11 Egenhofer's Spatial SQL (ESSQL).....	18
2.2.12 Pictorial SQL (PSQL).....	18
2.2.13 Scholl and Voisard's Relational Approach (SVRA).....	19
2.2.14 Gargano, Nardelli and Talamo's Relational Model (GNTRM).....	19
2.2.15 GeoSAL.....	20
2.2.16 Geo Relational Algebra (GRAL).....	21
2.2.17 QLG.....	21
2.2.18 Dedale.....	22
2.2.19 CALG.....	23
2.2.20 van Roessel's Conceptual Model (RCM).....	23
2.2.21 Scholl and Voisard's Thematic Map Model (SVTMM).....	24
2.2.22 ISO SQL Multimedia Standard: Spatial (ISOSQLMM).....	24
2.2.23 OpenGis Simple Features Specification for SQL (OGISSQL).....	26
2.2.24 Oracle8i Spatial Cartridge (ORACLE).....	26
2.2.25 IBM Informix Spatial DataBlade (INFORMIX).....	27
2.2.26 IBM DB2 Spatial Extender (DB2).....	27
2.2.27 PostgreSQL.....	27
2.2.28 GEO++.....	28
2.2.29 GEUS.....	28
2.2.30 Object-Oriented Geographic Data Model (OOGDM).....	28
2.3 PREVIOUS RESEARCH IN SPATIO-TEMPORAL APPROACHES.....	30
2.3.1 Güting et al Spatio-temporal Model (G+STM).....	30
2.3.2 Moreira, Ribeiro and Abdessalem's Spatio-temporal Model (MRASTM).....	31
2.3.3 Worboys's Spatio-temporal Model (WSTM).....	32
2.3.4 Erwig and Schneider's Spatio-temporal Partition Model (ESSTPM).....	32
2.3.5 d'Onofrio and Pourabbas's Temporal Map Model (OPTMM).....	33
2.3.6 Tryfona and Hadzilacos's Temporal GIS Approach (THTGIS).....	33

2.3.7	<i>Kemp and Kowalczyk's Temporal GIS Approach (KKTGIS)</i>	33
2.3.8	<i>STSQL</i>	34
2.3.9	<i>SQLST</i>	35
2.3.10	<i>Dedale</i>	36
2.3.11	<i>Yeh and de Cambray's Spatio-temporal Model (YCSTM)</i>	36
2.3.12	<i>Informix Geodetic DataBlade (GEODETTIC)</i>	36
2.3.13	<i>ORParaDB</i>	37
2.3.14	<i>Temporal Object-Oriented Geographic Data Model (TOOGDM)</i>	37
2.3.15	<i>Tripod</i>	38
2.3.16	<i>MOST</i>	38
2.4	CLASSIFICATION OF SPATIAL APPROACHES	39
2.4.1	<i>Discrete versus Continuous Change in Space</i>	39
2.4.2	<i>Raster-based versus Vector-based Representation of Space</i>	40
2.4.3	<i>GIS-centric versus Database-centric Spatial Data Management</i>	41
2.5	REMARKS ON SPATIAL APPROACHES	42
2.5.1	<i>Informal Data Types and Operations for Space</i>	42
2.5.2	<i>Spatial Data Types Deviating from Human Perception</i>	46
2.5.3	<i>Spatial Objects of Practical Interest Treated as Invalid Data</i>	47
2.5.4	<i>Lack of Spatial Data Validation Mechanisms</i>	48
2.5.5	<i>Need to Support Non-connected Spatial Objects as Valid Data</i>	48
2.5.6	<i>Need to Support Complex Spatial Data Structures</i>	48
2.5.7	<i>Limitations on Spatial Data Structures</i>	49
2.5.8	<i>Data Loss in Spatial Operations</i>	49
2.5.9	<i>Limited Functionality of Spatial Operations</i>	50
2.5.10	<i>Definition of Too Many Primitive Operations</i>	51
2.5.11	<i>Dimension Dependent Spatial Data Types and Operations</i>	51
2.5.12	<i>Availability of Implementation</i>	52
2.6	CLASSIFICATION OF SPATIO-TEMPORAL APPROACHES	52
2.6.1	<i>Discrete versus Continuous Change with Respect to Time</i>	52
2.6.2	<i>Valid Time versus Transaction Time Modelling</i>	52
2.6.3	<i>Tuple versus Attribute Time Recording</i>	53
2.7	REMARKS ON SPATIO-TEMPORAL APPROACHES	54
2.7.1	<i>Informal Data Types and Operations for Time</i>	54
2.7.2	<i>Time Data Types Deviating from Human Perception</i>	54
2.7.3	<i>No Support of Various Granularities of Time</i>	54
2.7.4	<i>Need to Support Spatio-temporal Data Types</i>	55
2.7.5	<i>Need to Support Complex Data Structures</i>	55
2.7.6	<i>Non-Generic Support of Temporal Data</i>	55
2.7.7	<i>Limitations on Data Structures for Time</i>	55
2.7.8	<i>Redefinition of Functionality of Conventional Operations</i>	56
2.7.9	<i>No Support of Evolution of Spatial Operations with Respect to Time</i>	56
2.7.10	<i>Definition of Too Many Primitive Operations</i>	56
2.7.11	<i>Availability of Implementation</i>	57
2.8	THESIS OBJECTIVES	57
2.9	CONCLUSIONS	58
3.	QUANTA AND DATA TYPES FOR SPACE	59
3.1	INTRODUCTION	59
3.2	SPATIAL QUANTA	59
3.3	DATA TYPES FOR SPACE	63

3.4 PREDICATES.....	65
3.5 FUNCTIONS.....	69
3.6 OPERATIONS.....	71
3.7 CONCLUSIONS.....	76
4. FORMALISM FOR SPATIAL DATA MANAGEMENT	77
4.1 INTRODUCTION	77
4.2 DATA STRUCTURES.....	77
4.3 RELATIONAL ALGEBRA OPERATIONS.....	78
4.3.1 <i>Conventional Operations</i>	80
4.3.2 <i>Additional Basic Operations</i>	82
4.3.3 <i>Quantum Operations</i>	85
4.3.4 <i>Pair-Wise Operations</i>	89
4.3.5 <i>Overlay Operations</i>	91
4.3.6 <i>Other Operations of Spatial Interest</i>	92
4.3.7 <i>Enhancement of the Functionality of Operations</i>	96
4.4 CONCLUSIONS.....	100
5. TEMPORAL AND SPATIO-TEMPORAL DATA MANAGEMENT .	101
5.1 INTRODUCTION	101
5.2 QUANTA AND DATA TYPES FOR TIME	102
5.3 PREDICATES AND FUNCTIONS FOR TIME.....	103
5.4 DATA STRUCTURES AND OPERATIONS FOR TIME	104
5.5 APPLICATION OF OPERATIONS TO TEMPORAL DATA.....	108
5.5.1 <i>Application of Quantum Operations to Temporal Data</i>	108
5.5.2 <i>Application of Pair-Wise Operations to Temporal Data</i>	110
5.5.3 <i>Application of Overlay Operations to Temporal Data</i>	112
5.5.4 <i>Application of Other Operations to Temporal Data</i>	113
5.6 EVOLUTION OF DATA WITH RESPECT TO TIME.....	115
5.7 APPLICATION TO SPATIO-TEMPORAL DATA.....	117
5.7.1 <i>Spatio-temporal Unfold and Fold Operations</i>	120
5.7.2 <i>Spatio-temporal Quantum Operations</i>	123
5.7.3 <i>Spatio-temporal Pair-Wise Operations</i>	124
5.7.4 <i>Spatio-temporal Overlay Operations</i>	129
5.7.5 <i>Other Interesting Spatio-temporal Operations</i>	130
5.8 CONCLUSIONS.....	133
6. SQL EXTENSION	135
6.1 INTRODUCTION	135
6.2 QUERY SPECIFICATION.....	136
6.3 QUERY EXPRESSION.....	139
6.4 NON-JOIN QUERY EXPRESSION.....	141
6.5 JOINED TABLE.....	147
6.6 UNARY QUERY EXPRESSION	149
6.7 CONCLUSIONS.....	151

7. COMPARISON WITH OTHER APPROACHES.....	153
7.1 INTRODUCTION	153
7.2 CLASSIFICATION AND EVALUATION OF SPATIAL APPROACHES.....	154
7.2.1 ROSE Algebra.....	154
7.2.2 Tomlin's Map Algebra (TMA).....	159
7.2.3 Erwig and Schneider's Spatial Partition Model (ESSPM).....	159
7.2.4 Hadzilacos and Tryfona's GIS Model (HTGIS)	160
7.2.5 ESRI ArcInfo 8.....	161
7.2.6 Intergraph Geomedia 5	162
7.2.7 MapInfo Professional 7	162
7.2.8 Bentley Microstation Geographics 7.2.....	162
7.2.9 Grid Based Commercial GIS Tools (GRIDGIS).....	163
7.2.10 Geosabrina.....	163
7.2.11 Egenhofer's Spatial SQL (ESSQL).....	164
7.2.12 Pictorial SQL (PSQL).....	164
7.2.13 Scholl and Voisard's Relational Approach (SVRA).....	165
7.2.14 Gargano, Nardelli and Talamo's Relational Model (GNTRM).....	165
7.2.15 GeoSAL	166
7.2.16 Geo Relational Algebra (GRAL)	166
7.2.17 QLG.....	167
7.2.18 Dedale	167
7.2.19 CALG.....	168
7.2.20 van Roessel's Conceptual Model (RCM)	168
7.2.21 Scholl and Voisard's Thematic Map Model (SVTMM)	169
7.2.22 ISO SQL Multimedia Standard: Spatial (ISOSQLMM).....	169
7.2.23 OpenGis Simple Features Specification for SQL (OGISSQL).....	170
7.2.24 Oracle8i Spatial Cartridge (ORACLE).....	170
7.2.25 IBM Informix Spatial DataBlade (INFORMIX).....	171
7.2.26 IBM DB2 Spatial Extender (DB2).....	171
7.2.27 PostgreSQL.....	171
7.2.28 GEO++.....	172
7.2.29 GEUS.....	172
7.2.30 Object-Oriented Geographic Data Model (OOGDM)	173
7.3 CLASSIFICATION AND EVALUATION OF SPATIO-TEMPORAL APPROACHES. 173	
7.3.1 Güting et al Spatio-temporal Model (G+STM)	177
7.3.2 Moreira, Ribeiro and Abdessalem's Spatio-temporal Model (MRASTM).....	177
7.3.3 Worboys's Spatio-temporal Model (WSTM).....	178
7.3.4 Erwig and Schneider's Spatio-temporal Partition Model (ESSTPM).....	179
7.3.5 d'Onofrio and Pourabbas's Temporal Map Model (OPTMM)	179
7.3.6 Tryfona and Hadzilacos's Temporal GIS Approach (THTGIS).....	180
7.3.7 Kemp and Kowalczyk's Temporal GIS Approach (KKTGIS)	180
7.3.8 STSQL.....	181
7.3.9 SQLST.....	181
7.3.10 Dedale	182
7.3.11 Yeh and de Cambray's Spatio-temporal Model (YCSTM).....	183
7.3.12 Informix Geodetic DataBlade (GEODETIC).....	183
7.3.13 ORParaDB.....	184
7.3.14 Temporal Object-Oriented Geographic Data Model (TOOGDM)	185
7.3.15 Tripod.....	185
7.3.16 MOST.....	186
7.4 CLASSIFICATION AND EVALUATION OF THE PROPOSED MODEL	186

7.4.1	<i>Classification and Evaluation with Respect to Spatial Properties</i>	186
7.4.2	<i>Classification and Evaluation with Respect to Spatio-temporal Properties</i>	188
7.5	COMPARISON WITH FUNCTIONALITY OF OTHER APPROACHES	190
7.5.1	<i>Predicates and Functions</i>	190
7.5.2	<i>Relational Algebra Operations</i>	190
7.6	ADDITIONAL CHARACTERISTICS.....	198
7.7	SIMILARITIES WITH OTHER APPROACHES	199
7.8	CONCLUSIONS.....	200
8.	SUMMARY AND FUTURE WORK	201
8.1	INTRODUCTION	201
8.2	SUMMARY	201
8.3	FUTURE WORK	203
8.3.1	<i>Definition of Predicates and Functions</i>	203
8.3.2	<i>Investigation of Efficient Storage Structures</i>	204
8.3.3	<i>Investigation of Optimisation Techniques</i>	204
8.3.4	<i>Implementation</i>	204
8.3.5	<i>Modelling Continuous Change in Space and in Time</i>	205
	REFERENCES	207
	APPENDIX A. IMPORTANT TAUTOLOGIES	229
	APPENDIX B. BNF FOR SQL EXTENSION	231
B.1	GENERAL REMARKS.....	231
B.2	QUERY EXPRESSIONS.....	231
B.3	LITERALS.....	234
B.4	SEARCH CONDITIONS.....	235
B.5	VALUE EXPRESSIONS	237
	APPENDIX C. IMPLEMENTATION	241
C.1	INTRODUCTION	241
C.2	ALGORITHMS	242

CHAPTER 1

INTRODUCTION

1.1 Background

A lot of research has been undertaken in recent years for the management of spatial data related to applications like cartography and cadastral systems. Geographic Information Systems (GIS), in particular, are fully dedicated to this problem. As should be expected, however, initial approaches exhausted their efforts in the precise geometric representation of spatial data and in the implementation of operations between spatial objects. Subsequently, only primitive effort was made on the association of spatial data with conventional data such as numbers, indicating various measurements (height, depth, etc) names (of cities, rivers, mountains etc), which make spatial data really meaningful. As a consequence, the management of geographic data had to be split into two distinct types of processing, one for the spatial data itself and another for the attributes of conventional data and their association with spatial data. Effort to define a formal and expressive query language for the easy formulation of queries was almost missing and, therefore, too much programming was required. Finally, even the processing of spatial data lacked an underlying formalism.

Given that efficient processing of conventional data can only be achieved from within a Database Management System (DBMS), a new research effort was next undertaken in the area of *spatial databases* [SCR99, RSV02], that covered various sectors such as the definition of data models and query languages, the design of efficient physical data structures and access methods, the investigation on query processing and optimisation techniques, visual interfaces, etc. As a result of these efforts, the last generation commercial and open-source DBMS include extensions of their models that enable the storage and management of spatial data. At the same time, the last generation GIS enable the storage of geographic data in spatially extended DBMS.

Another area, which attracted the interest of researchers, was that of *temporal* databases. Work in it is concerned with the management of the changes of

data with respect to time. Research in this area evolved in parallel with that in spatial data management. Finally, a combination of these two efforts gave rise to a new area, *spatio-temporal databases*, which is concerned with the management of the changes of spatial data with respect to time.

Research has also been undertaken in the modelling of spatial and of spatio-temporal data. However, the problems that appear are many and difficult to overcome. They are outlined below.

One first problem is that operations between pieces of spatial data have much individuality. If discussion is restricted to 2-dimensional spatial data, a typical example of this individuality is that the *spatial intersection* of two surfaces can give one out of many distinct results: It can be one surface (which can be seen as an element) but it may also be many surfaces (which can be seen as a set). It may also be the empty set, but clearly, it does not make too much sense to consider an *empty* surface. Many other results are also possible, such as one or more lines and one or more points. One case of particular interest is obtaining a spatial object, which is neither a surface nor a line but a combination of the two. In the most general case, the result is a set of surfaces, lines, points and spatial objects that can be seen as combinations of surfaces and lines. Similar observations can also be made for other spatial operations such as *spatial union* and *spatial difference*. The problem becomes even more complicated if such operations are considered between spatial objects of intuitively distinct spatial type, such as between a surface and a line, a surface and a point and so on. Although such operations do make sense in the real world, they are still seen as operations between *distinct* types of data. As a consequence, this problem has not yet been attacked satisfactorily even in GIS-based approaches, which are fully dedicated to the management of such data. The complexity of this problem has penalised all researchers and has been explicitly recognised in [SV92]. In the simplest case, *points*, *lines* and *surfaces*, which are the *types* adopted in daily practice, can be considered, but the result of any of the above operations is, in the general case, a set. Inversely, defining *sets of spatial objects* as data types deviates from daily practice.

A second problem is determining the most appropriate data structures to represent spatial data. The use of *layers*, borrowed from GIS-based approaches, seems convenient on a first glance, but the problem is how to associate spatial data with conventional data in an efficient way. Recording on the other hand, spatial data in conventional data structures gives rise to a new series of problems, which of these structures are the best. The immediate indication is that a non-nested relational structure does not suffice for this purpose. Hence, nested relational or object-oriented or combinations of the two had to be considered, even combinations between conventional relational

structures and layers. Definitely, however, such models lack the simplicity of a non-nested relational model.

Another problem, fully associated with the previous two, is the definition of appropriate operations between data structures. Considering only layers, inherits the problems discussed earlier. Considering only conventional relational structures gives rise to the question what is a proper set of operations, since operations between data structures seem to be different than those between spatial objects. Adopting on the other hand, two types of operations, one between such structures and another between spatial data, the question is how this can be achieved and whether the model defined this way lacks simplicity.

As should be obvious, in spatio-temporal databases the addition of one more dimension, *time*, increases further the complexity of the modelling problem, given that the modelling of only temporal data is by itself a major research problem, in which many diverse approaches have been proposed. This can be witnessed by the fact that many survey papers on temporal databases have appeared [SS88, AS93, K193, TT96, MS91].

Due to the above, many diverse spatial and spatio-temporal data modelling approaches have also been proposed, which differ substantially in the data types considered (simple or complex data types), the representation of spatial data (nested relational structures, object-oriented, complex object, object relational, combination of relations with layers, constraint-based relations) and the operations on them, including many sorted algebras. Given also the peculiarities of spatial data, many of the approaches restrict to only informal presentations either of the data types or of the functionality of the operations.

The present thesis proposes a solution that overcomes all the problems outlined above for the modelling of spatial and of spatio-temporal data for applications like cartography and cadastral systems. Part of the research was completed during a four-years (01/08/1996 - 31/07/2000) *Training and Mobility Research* project, '*CHOROCHRONOS: A Research Network for Spatiotemporal Database Systems*', that was funded by the European Union [FGG+99]. In this thesis three simple spatial data types are defined, matching fully the human perception of *point*, *line* and *surface*. The simple non-nested data structures of the pure relational data model are considered. The set of relational operations achieves almost all of the functionality proposed in other models. All the operations are defined in terms of a small set of kernel operations. All types of data, spatial, spatio-temporal and conventional are manipulated uniformly by this set of operations. An extension of SQL is also proposed for the management of spatial and of spatio-temporal data, based on a previous extension, IXSQL [LM97], for the management of temporal and,

more generally, *interval* data. Contributions of this thesis and an outline of the topics covered in the subsequent chapters are outlined below.

1.2 Contributions of this Thesis

The pieces of originality of the work in this thesis can be outlined as follows:

1. A discrete 2-dimensional (2-d) space is considered and formalism for 2-d *spatial quanta* is developed. Based on them, three spatial data types are defined, *point*, *line* and *surface*. An element of any of these types is a connected, closed subset of \mathbb{R}^2 . The empty set is not a valid spatial type. The definition matches human perception. Although the formalism considers a 2-d space, its generalization to either a 3-d or an n-d, $n \geq 3$, space is straightforward. Although the formalism is closer to raster-based approaches, it is argued that it is not restricted by the physical representation of spatial data. To the best of this author's knowledge, spatial quanta have not been considered in any spatial data modelling approach.
2. An algebra is defined for the management of spatial data, recorded in non-nested relational structures. The algebra inherits two fundamental characteristics: Firstly, it consists of only a limited number of *kernel* operations, the known operations of the relational model and two more, originally defined for the management of temporal data [LM97], *Unfold* and *Fold*. The remainder operations can be defined in terms of the kernel operations. Secondly, the management of spatial data actually reduces to the management of relations. Subsequently, a map matches the geometric representation of the contents of one or more relations. Although the definition of the operations considers the management of 2-d spatial objects, their extension, so as to apply to n-d spatial objects, $n \geq 3$, is again straightforward. It is also shown that operations of general acceptance, which are defined in other approaches, can be expressed in terms of the operations defined in this thesis. Due to previous research work [LM97], all the operations are closed, in that they can be applied to relations that contain any type of data. Consequently, all types of data can be handled uniformly by a unique set of relational algebra operations. As a direct consequence, the management of spatio-temporal data is straightforward, in that it can be seen only as one of the application areas of the proposed model. *Valid time* is considered at the temporal dimension, i.e. the time at which data is assumed to be

true in the real world. To the best of this author's knowledge, no other data modelling approach satisfies such properties.

3. An earlier SQL extension [LM97], for the management of temporal and interval data, has been extended further in a way that enables, in addition, the management of spatial and of spatio-temporal data.
4. All the known spatial and spatio-temporal approaches, and the model proposed in this thesis as well, are evaluated with respect to a number of properties. The evaluation has shown that the proposed model, though simple, overcomes limitations identified in other, complicated in general, approaches.

Research results of this piece of work have been reported in [LTV99, LVT99, LVT00, Vi00, VL01, VLT01, VL03a, LVT03].

1.3 Outline of Thesis

The remainder chapters of this thesis are organized as follows:

Previous research work on the management of spatial and of spatio-temporal data is reviewed in Chapter 2. Based on a number of limitations encountered in various approaches, a number of properties are identified, whose satisfaction by a relevant data model is desirable. In this way, the detailed objectives of this thesis are specified.

In Chapter 3 a discrete 2-d space is considered and a set of *spatial quanta* are formalized, *quantum point*, *pure quantum line*, *pure quantum surface*, *quantum line* and *quantum surface*. Based on quanta, a set of spatial data types is also defined, POINT, PLINE (*pure line*), PSURFACE (*pure surface*), LINE and SURFACE. Each of them consists of the union of relevant spatial quanta. An element of any of these types is a *spatial object*. One spatial object of particular interest is *hybrid surface*, composed of pure lines and pure surfaces. Operations between spatial objects are also defined. Finally, some spatial predicates and functions are defined.

In Chapter 4 non-nested relations with attributes of a spatial data type are considered. A relational algebra is defined for the management of relations containing spatial data. The kernel of the algebra consists of the known set of relational algebra operations plus two more, *Unfold* and *Fold*. Based of them, a series of practically useful operations is defined. It is shown that operations on relations reduce to operations on spatial data.

The management of temporal and of spatio-temporal data are addressed in Chapter 5. Initially, an alternative formalism for a temporal data model

defined in [Lo88, LM97] is provided, based on *time quanta*. Discrete time is considered and two generic time data types are defined, INSTANT and PERIOD. Some predicates and functions for time are also defined. Again, non-nested relations are considered for the representation of temporal data. The application of operations *Unfold* and *Fold* to relations, on a time attribute, is also defined. It is shown that it makes sense to apply to temporal relations operations, which were originally defined specially for the management of spatial data. In some cases it is shown that such an application has practical interest. Next, non-nested relational structures are considered for the representation of spatio-temporal data. It is shown that the management of such data does not require the definition of new operations. Formalism is also developed for the *evolution of data with respect to time*. Finally, relational algebra expressions are given that enable obtaining the *evolution of spatial data with respect to time*.

In Chapter 6 an SQL extension is defined for the management of spatial and of spatio-temporal data. The SQL syntax is a further extension of previous research work [LM97], aiming at the modelling of *interval* data. Of particular interest is the extension of a <query expression> by a <unary query expression> that enables incorporating in SQL a set of unary relational algebra operations defined in Chapter 4. The functionality of the extension is demonstrated by appropriate examples.

In Chapter 7 the spatial and spatio-temporal approaches described in Chapter 2 are evaluated with respect to the properties, which a relevant data model should satisfy. The model formalized in this thesis is also evaluated with respect to the same properties. This way, it is shown that it overcomes all the limitations identified in other approaches. It is also shown that operations defined in other approaches are not only expressible in terms of the operations defined in this thesis but a more general functionality is also achieved. Additional characteristics of the proposed model and similarities with other approaches are finally discussed

The findings of the research undertaken in this thesis and issues of further research are summarised in Chapter 8.

Finally, three appendices complete the thesis. Appendix A consists of some properties of the operations defined in Chapter 4, which can be used in an optimised implementation. Appendix B provides the syntax of the SQL extension. Finally, Appendix C provides pseudo code for an implementation of operations *Unfold* and *Fold* and of a predicate, *conductive*. Given that the definition of the remainder relational algebra operations has been based on them, it is shown that the model proposed in this thesis can be implemented.

CHAPTER 2

PREVIOUS WORK AND MOTIVATION

2.1 Introduction

A review of previous research work and of the functionality of commercial products for the management of spatial and spatio-temporal data is undertaken in this chapter. Comments on the various approaches are addressed, and the motivation for this thesis is outlined. For ease of presentation, various approaches are classified roughly with respect to their underlying data model as follows:

- *Spatial (Spatio-temporal) Object Models and Map (Temporal Map) Models*: The former provide formalisms for spatial (spatio-temporal) data types and operations on them. The latter provide formalisms for maps (evolution of a map with respect to time).
- *GIS-centric approaches*: They provide specialized data structures to support *Maps (Temporal Maps)*. These structures are usually based on conventional data structures such as relations.
- *Relational Approaches*: Spatial, temporal and spatio-temporal data are recorded in the attributes of relations. Relational algebra or SQL are extended by new predicates, functions and relational operations.
- *Nested Relational Models*: They support either set-valued or relation-valued attributes, i.e. they do not satisfy First Normal Form. Spatial, temporal, and spatio-temporal predicates and functions can be applied to set-valued attributes.
- *Nested Constraint-Based Models*: A spatial or spatio-temporal object is conceptually represented by a (possibly infinite) relation whose attributes are interpreted as the dimensions of an n-d space (time can be one of these dimensions). At a lower level of abstraction, such an infinite relation is represented by a finite set of constraints [KKR95].
- *Complex Object Models*: Their data structures are even more complex than those of nested models [AB95]. Generally, they are defined

recursively. Spatial operations are applied to specially defined spatial data structures.

- *Object Relational-Based Models*: They support user-defined data types. One such type includes a possibly complex data structure and a set of methods. It can be incorporated in the model in two distinct ways: (i) As the scheme of a table: A table of a user-defined type is the analogue of a class in an object-oriented model. Tuples are instances of a class. (ii) As any other conventional data type, in the definition of the scheme of a table. Note that the object relational model inherits the characteristics of the pure relational model and, at the same time, it incorporates object-oriented capabilities. Spatial types are used defined data types.
- *Object-Oriented Models*: Their data structures and methods are combined in the definition of classes. A hierarchy of classes is provided as a general tool for the design of spatial and spatio-temporal applications.

The remainder of this chapter is outlined as follows: In Section 2.2 (2.3) spatial (spatio-temporal) approaches are described, to the degree necessary, in the above classification order. In Section 2.4 spatial models are classified with respect to general characteristics. Comments on the various spatial approaches are made in Section 2.5. Similarly, spatio-temporal models are classified with respect to general characteristics in Section 2.6 and comments on the various approaches are made in Section 2.7. Based on the above remarks, the objectives of this thesis are outlined in Section 2.8. Conclusions are finally drawn in the last section.

2.2 Previous Research on Spatial Approaches

Spatial data modelling issues have inevitably been addressed in almost all the research areas of spatial databases, although it is not the primary research objective. However, such issues are discussed in this section, for reasons of completeness. Given that there are similarities between distinct research efforts, only the most representative approaches are presented to the necessary detail.

2.2.1 ROSE Algebra

The ROSE (RObust Spatial Extension) is a *spatial object model* for 2-d spatial data [GS93, GS95]. The definition of a set of spatial data types is based on a finite spatial structure called *realm*. An element of data type

POINTS is a finite and possibly empty set of vector points (g_1, g_2, g_3 in Figure 2.1(a)). An element of data type LINES is a finite and possibly empty set of *polylines*, (g_4, g_5 and g_6 in Figure 2.1(b)). An element of data type REGIONS is a finite and possibly empty set of polygons and it may have holes (g_7 and g_8 in Figure 2.1(c)). AREA is a REGIONS type with an additional restriction, that the intersection of two areas, which are recorded in the same attribute, may not be a surface. A long set of primitive operations is formalized. Given two values of some spatial type D, operation *Union* (*Minus*) yields that part of the spatial union (spatial difference) of the input values that is also of type D. Operation *Intersection* yields one of the following results: (i) Points which are common to two elements of type POINTS, (ii) *isolated* points in the spatial intersection of two elements of type LINES (only under certain conditions), (iii) Surface components of the spatial intersection of two elements of type REGIONS, (iv) line segments of an element of type LINES that are inside but not on the boundary of an element of type REGIONS.

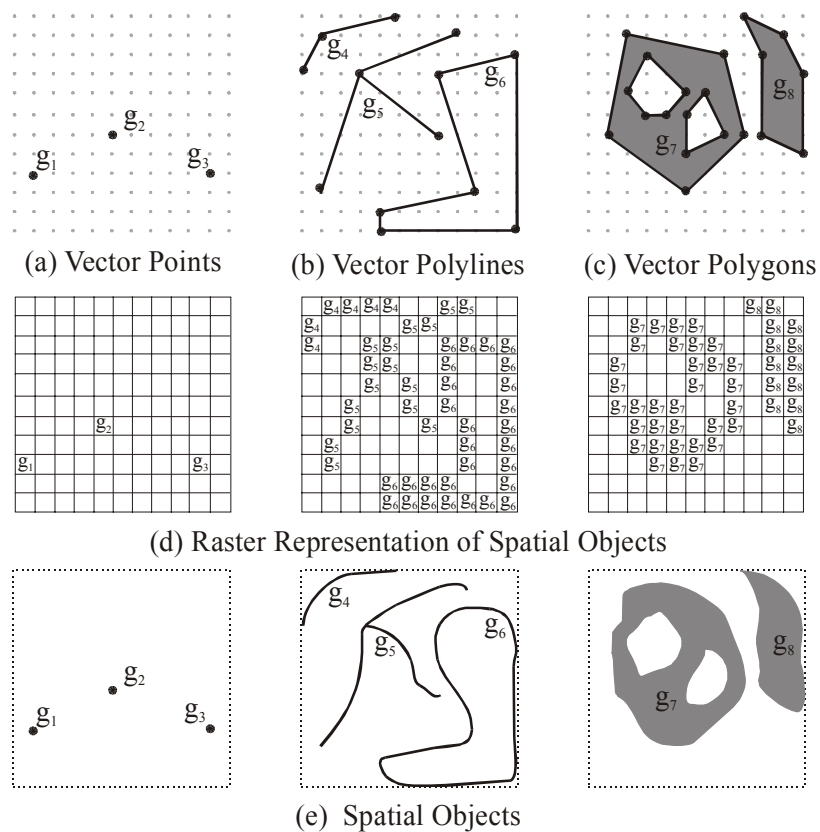


Figure 2.1: Representation of spatial objects in various approaches.

Operation *Common_border* gives one of the following results: (i) Line segments in the spatial intersection of two elements of type LINES, (ii) line segments in the spatial intersection of the boundaries of two elements of REGIONS type, (iii) line segments in the spatial intersection of an element of LINES type and the boundary of an element of type REGIONS. Operation *Vertices* gives the set of points that are vertices of segments in an element of either LINES or REGIONS data type. Operation *Contour* yields the boundary (LINES data type) of an element of type REGIONS. Additional functionality enables testing topological relationships, calculating distances, lengths, areas and perimeters, constructing the element of type REGIONS that is enclosed by an element of type LINES and obtaining the number of connected components of a spatial object. Assuming the existence of an underlying conventional data model, the following operations manipulate data structures that combine spatial and conventional data. Operation *Decompose* decomposes spatial objects into their connected components. Operation *Fusion* computes the spatial union of all the spatial objects that share identical conventional values. Finally, operation *Overlay* computes the spatial intersection of every element of type AREA in one data structure with every element of type AREA in another.

2.2.2 Tomlin's Map Algebra (TMA)

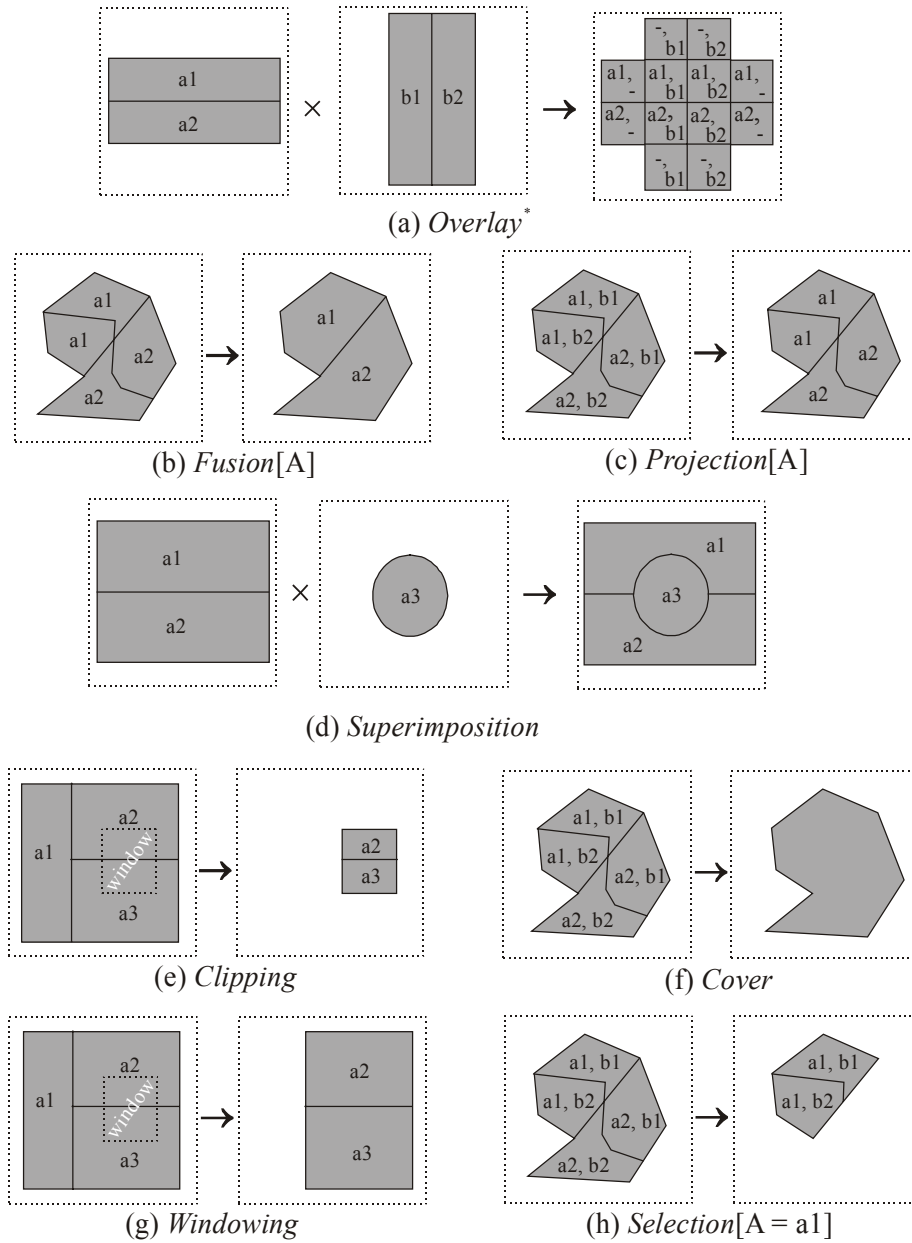
A *map model* and a relevant *map algebra*, at a conceptual level of abstraction, is described in [To90, To91, To94, To97]. The main objective is to propose a classification of the operations required for the management of maps of any kind. The set of operations that is proposed in each class does not intend to be complete. Vector and raster representations can be used at the implementation level. In this approach, a map is considered, in an informal way, as a spatial representation of a given area. A map is composed of *zones*. Each *zone* is composed of a set of *locations* (i.e. of points in the underlying 2-d space) and has an associated numeric value. All the locations of a given map that have the same associated value are automatically *merged* into a single zone. For ease of subsequent presentation, the author of this thesis introduces some notation: Characters m and p , possibly subscripted, denote maps and locations, respectively. The value associated with a location p in a map m is denoted as $m(p)$. $Z_{p,m}$ denotes the set of locations in the same zone of a location p in a map m . $N_{p,m}$ denotes the set of locations in the neighbourhood of a location p in a map m . A neighbourhood can be defined in a variety of ways. As an example, a location p_1 is in the neighbourhood of location p_2 in a map m iff p_1 and p_2 are in the same zone. As another example, a location p_1 is in the neighbourhood of location p_2 iff the distance of p_1 from p_2 is less than a given threshold. The map operations can be classified into four types:

1. *Local Operations*: Given a set of input maps m_i , the value $m(p)$ in the output map m is obtained as a function of $m_i(p)$, for each location p . A worth mentioning example is *LocalCombination*, where $m(p)$ is obtained as the combination of all $m_i(p)$ (it matches the *Overlay* operation of other approaches).
2. *Zonal Operations*: Given two input maps m_1 and m_2 , the value $m(p)$ of the output map m is obtained as a function of either one or two of the following parameters: (i) the value $m_2(p)$ and (ii) the set $\{m_1(p_1), \dots, m_1(p_n)\}$, where each p_i is in Z_{p,m_2} . For example, *ZonalAverage* computes the average of $\{m_1(p_1), \dots, m_1(p_n)\}$, and *ZonalRanking* computes the number of elements in $\{m_1(p_1), \dots, m_1(p_n)\}$ that satisfy $m_1(p_i) < m_2(p)$.
3. *Focal Operations*: Given an input map m_1 , the value $m(p)$ of the output map m is obtained as a function of the values in $\{m_1(p_1), \dots, m_1(p_n)\}$, where each p_i is in N_{p,m_1} .
4. *Incremental Operations*: They extend the set of Focal operations. They take into account the type of the zone at each location (point, line or surface).

The set of operations is a potential standard [Ogis00, Ogis01b] and serves as a basis of commercial tools (Subsection 2.2.9) for the manipulation of maps that represent properties that change continuously in space. Further research [KP90, EB95] is dedicated to the design of graphical user interfaces.

2.2.3 Erwig and Schneider's Spatial Partition Model (ESSPM)

In [ES97, ES00] a conceptual model (*map model*) is presented for a special type of maps, namely *spatial partitions*. *Spatial partitions* and primitive operations on them are formalized. Informally, if $A = A_1 \times A_2 \times \dots \times A_n$, $n > 0$, is a Cartesian product of conventional data types, then a *spatial mapping of type A* is defined as a total mapping $SM: R^2 \rightarrow A \cup Power(A)$, where $Power(A)$ denotes the powerset of A . Each subset of R^2 , whose points are mapped to the same value, is called *block*. If a block is mapped to a single value it is a *region*, otherwise it is a *border*. A *spatial partition of type A* is defined as a *spatial mapping of type A* that satisfies the following two properties: (i) Each region r is a regular open set, i.e. $r = interior(closure(r))$. (ii) Each border is mapped to the set of values of its adjacent regions. Three primitive operations are defined on spatial partitions. Informally, *Intersection* computes the overlay of two spatial partitions (Figure 2.2(a)). If f is a function ($f: A \rightarrow B$) then *Relabel* enables applying f to each conventional value of the attribute of a spatial partition of type A in order to obtain a spatial partition of type B .



* Overlay is proposed only for maps of the same cover in [SV89].

Figure 2.2: Representative operations on maps proposed in [SV89].

Note that regions in the result partition with the same B value are automatically merged (Figure 2.2(b)). *Refine* assigns a different identification number to each surface (connected component of a region) in a partition. A

representative functionality for map management, originally proposed in [SV89] (Figure 2.2), can be achieved by these last three operations. *Categorical coverage*, presented in [VE93, FVM97], resembles that of *spatial partition*. Note however that categorical coverages are not formalized in [VE93, FVM97] and operations between them are not provided.

2.2.4 Hadzilacos and Tryfona's GIS Approach (HTGIS)

This piece of research [HT96, DHT94, HT92] combines the map model and the relational model, into a single *GIS* approach for the management of spatial and conventional data.

The *Map Model* and a relevant map algebra for 2-d spatial data are described in [DHT94]. The approach formalizes the concept of *layer*, and proposes a primitive set of operations on layers. Only few of the operations are formalized. A POINT data type consists of R^2 points. An element of type ARC is a simple polyline (g_4 in Figure 2.1(b)). An element of type REGION is a connected subset of R^2 that is defined by a polygon and it may have holes (g_7 and g_8 in Figure 2.1(c)). Any combination of these spatial data types is also a spatial data type. A set of primitive spatial predicates and functions is given. It includes functions *union*, *difference* and *intersection* whose result is, in the general case, a set of spatial objects of any spatial data type. A *layer* L is defined as a mapping from a set of spatial values G to the Cartesian product of a set of conventional attributes ($L: G \rightarrow C_1 \times C_2 \times \dots \times C_n$). Four primitive operations are defined on layers. *Attribute derivation (Spatial computation)* enables the application of conventional (spatial) functions and predicates. Conventional attributes are not maintained in the result of a spatial computation. *Reclassification* merges into one all the range tuples of a layer, which satisfy the following two properties: (i) They have identical values in a given attribute and (ii) They concern adjacent spatial objects (Figures 2.2(b)). This operation can be applied only to layers of type ARC or REGION. Finally, operation *Overlay* can be applied to two layers of any type.

The layer algebra enables the user to create derived *virtual layers* from others already existing. *Object classes* combines spatial data, stored in layers, with conventional data, recorded in relations, into a single data structure. *Constraints* [HT92] are used in the definition of object classes and enable specifying which spatial features of each layer and which tuples of each relation compose each spatial object. Queries can thus be expressed by defining object classes.

2.2.5 ESRI ArcInfo 8

ArcInfo is a commercial *GIS* developed by ESRI (Environmental Systems Research Institute) [Esri99, Esri00, Ze99, Ma99, Tu00, MSW99, Esri02a, Esri02b]. Its spatial data model, *Geodatabase*, is organized in *geographic datasets* [Ma99]. Geographic data sets are either of the following: (i) *Feature datasets*, used to model maps containing 2-d spatial objects. (ii) *Raster datasets*, used to model maps representing properties that change continuously in space. (iii) *TIN datasets*, used to model *elevation* data. Within the objectives of this thesis, this subsection restricts to only feature datasets and tools provided for their manipulation. However, tools for the manipulation of raster datasets are described in Subsection 2.2.9.

A *feature dataset* is a collection of *object classes* (relations) and *feature classes* (relations with one spatial data type called *shape*). Both of them are implemented as tables in a DBMS. The ArcSDE module [Esri02a] acts as an interface between ArcInfo and commercial DBMSs and enables taking advantage of the spatial capabilities of DBMSs [Ora00, Inf01, Ibm01]. An element of a POINT data type is a vector point (Figure 2.1(a)). An element of MULTIPOINT data type is a collection of POINT elements. An element of POLYLINE type is a collection of simple vector polylines (g_6 in Figure 2.1(b)). An element of type POLYGON is a disjoint collection of polygons (Figure 2.1(c)). The empty set is a valid spatial object. Predicates defined between these data types enable testing topological relationships. The user interface of ArcInfo has three components, ArcMap, ArcCatalog and ArcToolbox. From these, ArcMap provides tools for the visualization and exploration of feature classes in maps [MSW99]. ArcToolbox contains over 120 advanced geoprocessing tools for the manipulation of both feature and object classes [Tu00]. Functionality relevant to the present work is outlined as follows: *Selection tools* enable using a restricted version of SQL with spatial predicates. Operation *Dissolve* merges adjacent polygons or polylines in a feature class with the same value in a given conventional attribute. Similarly, operation *DissolveRegions* combines polygons, either adjacent or not with the same value in a conventional attribute. Given a feature class of any data type A and a feature class B of type POLYGON, three operations enable computing the overlay of feature classes. The result type is that of A. Operation *Intersection* yields pieces of spatial objects in A lying inside some polygon B. Operation *Identity* yields the result of *Intersection* plus the pieces of spatial objects in A that lie outside all the polygons of B. Finally, operation *Union*, applied to two POLYGON feature classes, yields the result of operation *Identity* plus the pieces of spatial objects in B that lie outside the polygons of A (Figure 2.2(a)). Operation *Erase* yields a new feature class with the pieces of spatial objects of A outside all the polygons of B. Operation

Update yields the *Superimposition* (Figure 2.2(d)) of two feature classes whose scheme is compatible. Other functionalities are *Buffer*, *Clipping* (Figure 2.2(e)) and *Cover* (Figure 2.2(f)). A final one yields the *Thiessen polygons* that are associated with a set of points.

2.2.6 Intergraph Geomedia 5

Geomedia [LH98, Int02] is a commercial *GIS* developed by Intergraph Corporation. Its characteristics are close to those of ESRI ArcInfo, described in the previous section, therefore, only the main differences are discussed here. Besides data types POINT (MULTIPOINT in ArcInfo), LINE (POLYLINE in ArcInfo) and AREA (POLYGON in ArcInfo), an additional data type COMPOUND consists of spatial objects of any of the previous data types. Contrary to ArcInfo, the empty set is not a valid spatial object in Geomedia. Similarly with ArcInfo, more than one attribute of a spatial data type is allowed in a feature class. However, only one of them is the *primary geometry*. Contrary to ArcInfo, a conventional relation is also considered as a *feature class*. Regarding spatial data management, the set of tools provided in Geomedia is smaller than that of ArcInfo. Operations with a functionality similar to that of operations *Dissolve* and *DissolveRegions* of ArcInfo apply to feature classes that contain spatial objects of any spatial data type. Operation *Spatial intersection* of Geomedia can be applied between two feature classes of any spatial data type. Operation *Spatial intersection* of two feature classes of type AREA yields, in the general case, points, lines and areas. The *Spatial difference* operation does not allow subtracting lines from surfaces and points from lines.

2.2.7 MapInfo Professional 7

MapInfo Professional [Mapi01, Mapi02] is a commercial *GIS* developed by MapInfo Corporation. Its characteristics are close to those of ESRI ArcInfo, (Subsection 2.2.5). However, only one data type ST_SPATIAL is supported. Operations *Dissolve*, *Erase*, *Update* and the various types of *Overlay* are not supported.

2.2.8 Bentley Microstation Geographics 7.2

Microstation Geographics [Bent01] is a commercial *GIS* developed by Bentley Systems. It provides the user with a data structure, called *Feature*, which enables the modelling of maps that contain 2-d spatial objects. A *Feature* is implemented as a relation in a conventional DBMS. Two of the

attributes of a relation (*mblink* and *mapid*) are used to store pointers to spatial objects stored in CAD files. Note that the same spatial object may be addressed from to distinct *Features*. Valid spatial objects in a CAD file include points, polylines, polygons and non-empty collections of them (Figures 2.1(a-c)).

Regarding spatial data management, the SQL of the underlying DBMS can be used to query *Features*. When more than one feature is included in a FROM clause, one of them is treated as the master table and the others as look up tables. In addition, specialized tools support an enhanced functionality such us *Spatial selection*, *Overlay* and management of networks. Another commercial *GIS* with a similar functionality is provided by Intergraph [Int95].

2.2.9 Grid Based Commercial GIS Tools (GRIDGIS)

In many commercial *GIS*, a data structure, raster *grid*, is provided for the modelling of maps that represent properties that change continuously in space. Examples of such systems are ArcGIS Spatial Analyst (grid extension for ArcInfo) [Esri01, MJ01], MFWorks [Keig02] (also provided as a grid extension of Geomedia), GRASS (Geographic Resources Analysis Support System) [Gras02], IDRISI [Lo00] and MAPCALC [BK01, RHS01]. Characteristics that are common to them are described in this subsection.

Informally, a 2-d raster *grid* is a partition of a given rectangular area into a matrix of squares called *cells* or *pixels*. The number of rows and columns in the matrix determines the resolution of the grid. Each cell is associated with a numeric value representing a given property at that location. Each set of cells with identical numeric value is called a *zone* or *category*. Grids provide efficient discrete representations for physical phenomena that change continuously in space like temperature, atmospheric pressure, elevation, etc. Some examples of operations between grids are shown in Figure 2.3. Operation *Clump* reassigns distinct identification numbers to connected zones (Figure 2.3(a)). Operation *Classify* changes the numeric values of some zones of a grid according to given rules Figure 2.3(b). Operation *Interpolation* produces a continuous surface from a set of discrete values by applying some interpolation algorithm (Figure 2.3(c)). *Zonal* operations apply a given aggregate function (average, for example) to the values of a grid, for each zone in another grid (Figure 2.3(d)). Operation *Compute* applies an algebraic pixel-wise operation between two grids Figure 2.3(e). Operation *Combine* assigns distinct numbers to each distinct combination of values for the same pixel in two distinct grids (Figure 2.3(f)). Operation *Drain* computes the path that water would follow from a given set of sources in a grid through a surface of elevations in another grid. Operation *Radiate* returns the pixels that are

visible from a given set of *source points* in a given grid of *elevations* (Figure 2.3(h)). Operation *Spread* computes the minimum cost to go from each non-zero pixel in a grid to its closest non-zero pixel in the same grid. The cost of crossing each pixel is given in another grid (Figure 2.3(i)).

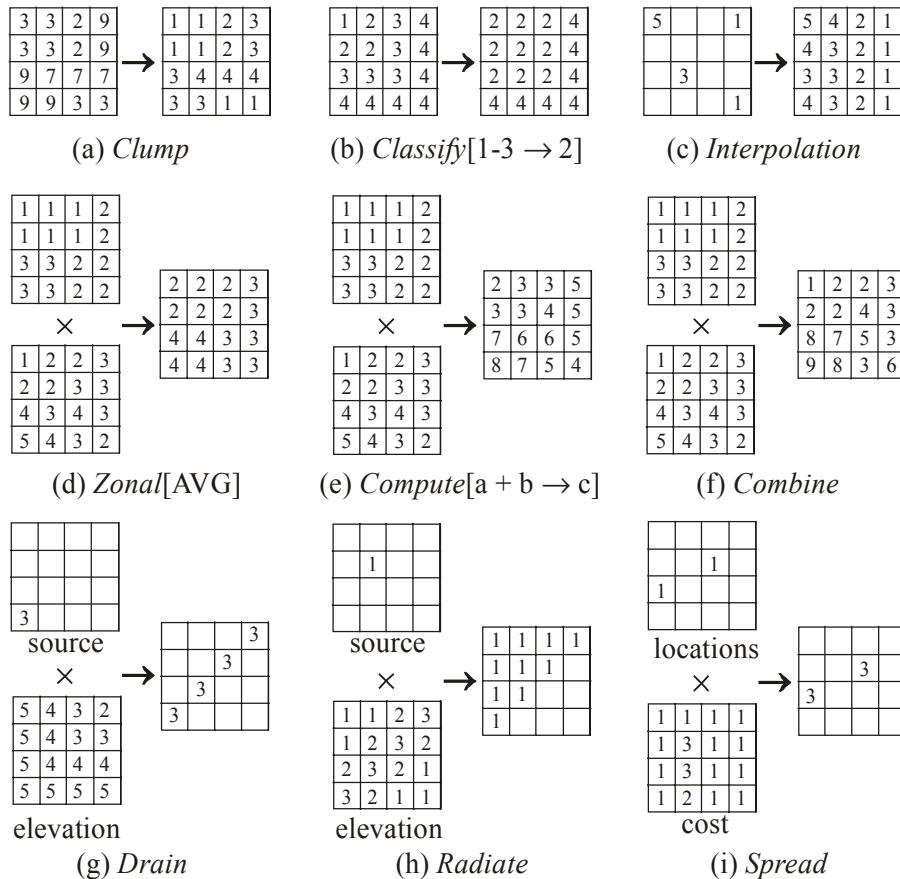


Figure 2.3: Examples of operations on raster grids.

2.2.10 Geosabrina

Geosabrina is a spatial DBMS implemented as an extension of the commercial relational DBMS Sabrina, and aims at the management of 2-d spatial data [LPV93, CVL+94]. An extension of it, for the management of 3-d spatial data, is proposed in [YC94a]. The main objective of the approach is the description of a spatial DBMS rather than the formalization of a spatial data model. Only one spatial data type, GEOMETRY, is supported.

Informally, an element of type GEOMETRY is a set of spatial objects that are either points or polylines or polygons, possibly with holes (Figures 2.1(a-c)). Any number of attributes of type GEOMETRY can be incorporated in the definition of a relation. Both predicates and functions can be incorporated in SQL queries for the manipulation of spatial objects. Amongst the functions, worth mentioning are those that enable the computation of *spatial union*, *spatial difference* and *spatial intersection* of two spatial objects. An aggregate function, called *sum*, yields the spatial union of a set of spatial objects.

2.2.11 Egenhofer's Spatial SQL (ESSQL)

A *relational* SQL extension for the management of 3-d spatial data is described in [Eg94]. The main objective of the approach is the identification of general requirements for a spatial SQL, with particular interest in spatial data presentation rather than the formalization of a set of data types and operations. Four spatial data types, described informally, enable the representation of points, lines, surfaces and solids. An additional data type, SPATIAL, consists of all the previous data types. The representation of spatial values follows a vector-based approach (Figures 2.1(a-c)). Any number of attributes of a spatial data type can be included in the definition of relations. The author of the paper argues that spatial data manipulation can be achieved by the incorporation of spatial predicates and functions in SQL queries. Although definitions of predicates are given in [Eg89, EF91] no spatial functions are defined. Two of them, that are discussed informally, are *Complementation* and *Boundary*. In this approach, the *boundary* of a line is a set of points and the boundary of a point is the empty set.

2.2.12 Pictorial SQL (PSQL)

An SQL extension for the management of 2-d spatial data is described in [RFS88]. However, the main objective of the approach is the efficient management of spatial data by the use of R^+ -trees rather than the formalization of spatial data types and operations. A difference from the previous approach is that both vector and raster spatial representations are allowed internally. Example spatial data types that can be supported are points, lines and surfaces. It is not clear whether the empty set is a valid spatial object. Regarding *spatial intersection*, an example of a function is given, which yields the intersection points of two lines.

2.2.13 Scholl and Voisard's Relational Approach (SVRA)

The experience on the implementation of a *relational* (actually relational-like) extension for the management of 2-d spatial data, on top of the object-oriented DBMS O2, is described in [SV92]. The approach does not aim at defining a complete set of spatial data types, predicates and functions, but just at reporting on the peculiarities identified during the implementation phase. An element of type POINTS is defined as a finite set of points in R^2 . A line is defined in terms of line segments. Each line segment consists of an infinite number of R^2 points (Figure 2.1(b)). The authors state that whether the end points of a line belong to the line or not is beyond their objective. An element of data type LINES is defined as a finite set of lines. An element of type REGIONS is defined as a subset of R^2 composed of a finite set of polygons without holes (g_8 in Figure 2.1(c)). As in the case of lines, whether the boundary of a polygon belongs or not to a region is left open by the authors. Some spatial predicates are defined. Four functions enable computing the *spatial intersection* of two spatial objects. Function *inside* returns the pieces of a LINES element which are inside a REGIONS element. Function *rintersection* returns the REGIONS element contained in the spatial intersection of two REGIONS elements. Similarly, functions *lintersection* and *pintersection* return the LINES elements and POINTS elements, respectively, which are contained in the spatial intersection of two LINES elements. Function *border* returns the LINES element that forms the boundary of a REGIONS element.

2.2.14 Gargano, Nardelli and Talamo's Relational Model (GNTRM)

A formalism of a *relational* model for the management of 2-d spatial data is given in [GNT91a, GNT91b]. The extension of the conventional relational model is minimal, since only one generic data type and three new relational algebra operations are defined. For an informal description, if S is the set of all raster cells (pixels) in a grid, the possible elements of data type GEOMETRY(S) are defined as sets of sets of elements in S . Thus, only surfaces can directly be represented in the model, whereas points and lines have to be approximated by surfaces (Figure 2.1(d)). Both the empty set and non-connected surfaces are valid elements of GEOMETRY(S). Relational algebra is extended by three new primitive operations. Operation *G-Compose* merges the spatial objects recorded in some attribute of a relation R , provided that they are in tuples whose value in some other attribute of R is the same. Inversely, operation *G-Decompose* decomposes each spatial object to so many tuples as the number of cells it consists of. Finally, the spatial functionality of

operation *G-Fusion* is identical to that of *G-Compose*, but it enables, at the same time, the application of aggregate functions to non-spatial attributes.

2.2.15 GeoSAL

A *many sorted algebra*, used in a prototype of system designed for spatial data analysis, is described in [SH91, HSH92, HS93]. Spatial data types and relations enable the management of points, lines, polygons and raster grids. An element of data type POINT is defined as a pair (x, y) of real numbers. Data type LINE consists of simple vector polylines (g_4 in Figure 2.1(b)). Data type POLYGON consists of vector polygons without holes (g_8 in Figure 2.1(c)). Subtypes of polygon include REG_POLYGON, SQUARE and XY_SQUARE, which represent, respectively, regular polygons, squares and orthogonal squares. Any number of the previous spatial types can be declared at the definition of the scheme of a relation. A raster grid is defined as a relation whose scheme includes one attribute of a SQUARE data type. The many sorted algebra enables defining a long set of operations between combinations of (i) conventional values, (ii) spatial objects, (iii) sets of conventional values and spatial objects and (iv) relations. Operations *Union*, *Difference* and *Intersection* yields, respectively, the spatial union, difference and intersection, of two spatial objects of the same data type, either LINE or POLYGON. The result of the *Union* of two lines (polygons) is a set of lines (polygons). The same observation also applies to *Difference*. The *Intersection* of two polygons is a set of polygons. However, the *Intersection* of two lines is exclusively either a set of lines or a set of points, depending on whether the lines are partly collinear. Operations *Union* and *Intersection* can also be used as aggregate functions. Note that, although the results of the previous operations are sets, an implicit *Unnest* operation always yields a flat data structure. Operation *Boundary* yields the boundary line of an element of a POLYGON data type. Further functionality for the manipulation of spatial objects includes the calculation of the *length* of lines, of the *area* of surfaces, of the *shortest Euclidian distance* of two spatial objects, of the *gravity centre* of a spatial object, of the *buffer* area of a spatial object, of the *split* of a polygon with respect to a line, of the split of a line with respect to a point and of the *voronoi diagram* of a set of points. Specific functionality for the manipulation of raster grids includes operations *Visible* (resembles operation *Radiate* in Figure 2.1.(h)) and *Diffuses* (resembles operation *Spread* in Figure 2.1.(i)).

2.2.16 Geo Relational Algebra (GRAL)

Another *many sorted algebra* for the management of 2-d spatial objects is defined in [Gu88, Gu89]. Three spatial data types are supported. An element of type POINT is a set of just one element of R^2 . An element of type LINE is an infinite subset of R^2 defined by a simple polyline (g_4 in Figure 2.1(b)). A value of type PGN is an infinite subset of R^2 defined by a polygon without holes (g_8 in Figure 2.1(c)). Data type AREA is defined as PGN, with an additional restriction, that the intersection of two polygons that are recorded in the same column may not be another polygon. The approach has similarities with that of the previous subsection. Operation *Intersection* enables obtaining part of the result of a spatial intersection of pairs of spatial objects recorded in two relations. The following cases are identified: (i) If both relations contain only lines then the result relation contains only isolated points. (ii) If both relations contain only polygons then the result relation contains only polygons. (iii) If the first relation contains only lines and the second only polygons then the result relation contains only lines. (iv) If both relations contain only areas then the operation is called *Overlay* and the result contains only areas. Implementation issues are discussed in [Gu89].

2.2.17 QLG

A *nested relational* model and query language for the management of 2-d spatial data is proposed in [CZ96, CW96, CN97]. A set of operators is defined, which is intended to be general and independent of any particular nested data model. A mechanism for the incorporation of new functions is also provided. Spatial objects are defined as closed, non-empty and possibly infinite subsets of R^2 . Data type POINT consists of R^2 points. Elements of data type S_LINE are defined as simple polylines (g_4 in Figure 2.1(b)). Data type LINE consists of polylines of any kind (g_4 , g_5 and g_6 in Figure 2.1(b)). Data type LINE* consists of both POINT and LINE types. Data type S_REGION consists of polygons without holes (g_8 in Figure 2.1(c)). Data type REGION consists of polygons of any kind (g_7 and g_8 in Figure 2.1(c)). Finally, data type REGION* consists of both REGION and LINE* types. Due to the use of a nested relational model, sets of elements of the previous types are also valid data types. A long set of primitive operations is defined. Operation *Intersection* computes the spatial intersection of two spatial objects of any type. The result, in the general case, is a set of spatial objects of type REGION*. Operation *Fusion* computes the spatial union of a set of spatial objects of any data type. The result is a set of spatial objects of the same data type. In [CN97], an algorithm is proposed that enables subtracting regions from either lines or regions. Additional functionality includes: *length* of lines,

area of surfaces, *minimum and maximum Euclidian distance*, *slope* of the straight line linking two points, *coordinates* of points, *end points* of lines, *envelope* and *holes* of surfaces, *centroid*, the set of *paths* linking two points in a network of lines, computation of *buffer*, *split* and *voronoi*, and *projection* of spatial objects in space. The description of a developed prototype is presented in [CW96]. Algorithms for the computation of various spatial operations are proposed in [CN97].

2.2.18 Dedale

A *nested constraint-based model* for the management of multidimensional spatial and spatio-temporal data is described in [GRSS97, GRS98a, GRS98b, GRS00, RSSG02]. In particular, the model for multidimensional spatial data is described in [GRSS97, GRS98a]. The demonstration of its capabilities in spatio-temporal data management is presented in [GRS98b]. In [GRS00] it is shown how the model supports interpolated data. Examples of interpolated data are: (i) Properties that change continuously in space, such as elevation, temperature, etc and (ii) moving spatial objects whose position changes continuously with respect to time. Finally, some implementation issues are addressed in [RSSG02]. The approach enables the incorporation of infinite collections of n-d points as values in a nested relational model. At an *abstract level*, such infinite collections are represented as infinite relations of n attributes, one for each dimension. At a *symbolic level*, infinite relations, called *linear constraint relations*, are represented by finite sets of linear constraints. Such a *constraint* is a formula of the form $a_1x_1 + a_2x_2 + \dots + a_nx_n \theta a_0$, where a_i are integers, x_i are variables, one for each dimension, and θ is either = or \leq . The following are examples of the types of data that can be represented by a *linear constraint relation*:

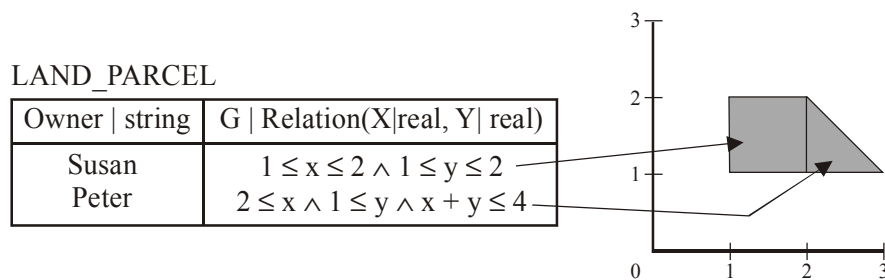


Figure 2.4: Relation with spatial data in the constraint data model.

- $R(x, y)$, where x and y represent the orthogonal coordinates of a 2-d plane, stores the points of a 2-d spatial object. An example of a nested

relation with two linear constraint relations, which represent two spatial objects, is shown in Figure 2.4.

- $R(x, y, m)$ stores the value of a property m for each point (x, y) of a 2-d plane. This enables representing properties that change continuously in space, like elevation, temperature, etc.
- $R(x, y, t)$, stores the value of the x and y coordinates of a moving point for each time instant t , i.e., spatial data that changes continuously with respect to time.

Relational algebra operations are defined between linear constraint relations. For example, spatial union, difference and intersection are achieved by the relational operations *Union*, *Except* and *Intersect*. Operation *Unionnest* applies relational operation *Union* to all the relations of a relation-valued attribute, provided that they belong to tuples whose values for another set of attributes are identical. The behaviour of operation *Internest* is similar to that of *Unionnest* except that *Intersection* is applied instead of *Union*. When relational algebra operations are applied, tuples containing in one attribute an empty linear constraint relation are automatically discarded. Further spatial functionality includes spatial *Boundary* of surfaces, spatial *Complementation*, predicates for testing topological relationships and one predicate to test if the *distance* of two spatial objects is less than a given constant.

2.2.19 CALG

A *nested constraint-based model* for the management of multidimensional spatial data is described in [KRSS98]. The behaviour is very close to that of the Dedale approach, described in the previous subsection. Regarding the management of spatial data, the only difference is related to the management of the empty set. Specifically, the empty set is a valid linear constraint relation in CALG, as opposed to Dedale.

2.2.20 van Roessel's Conceptual Model (RCM)

A *complex object model* for the management of 2-d spatial data is presented in [Ro93, Ro94]. Two primitive operations, *Fold* and *Unfold* (borrowed from research on temporal databases [LJ88b]) enable defining four kinds of *Overlay* between relations with spatial data. The behaviour of the approach is close to the GNTRM approach described in Subsection 2.2.14, therefore, only major differences are outlined here. Points and infinite sets of R^2 points are now valid data types. Contrary to GNTRM, two distinct spatial data types are defined one for connected and another for non-connected subsets of R^2 . Operations *Fold* and *Unfold* resemble operations *G-Compose* and *G-*

Decompose in GNTRM. Operation *Fold* is defined in four steps, and one of them requires the use of nested data structures. Based on *Fold*, *Unfold* and the four types of Codd's outer natural join, four types of *Overlay* operations are defined. Their functionality resembles that of the *Overlay* operations of ArcInfo (Subsection 2.2.5).

2.2.21 Scholl and Voisard's Thematic Map Model (SVTMM)

A *complex object Model* that enables the manipulation of 2-d spatial data is described in [SV89]. An *elementary region* is defined as a subset of R^2 . A *region* is either an elementary region or a set of elementary regions. Various predicates and functions are defined that enable the manipulation of regions. They include: (i) A predicate that tests whether an elementary region is empty. (ii) Functions that compute spatial union, spatial difference and spatial intersection of two elementary regions. The functions return an elementary region and they are defined in terms of the set operations \cup , \cap and $-$. Hence, either empty or non-closed regions may be obtained. (iii) Spatial union, which can also be applied to a set of elementary regions in order to obtain a single elementary region. (iv) A generalization of spatial intersection that is applied to pairs of any type of regions, either elementary or not. A *map* is defined as a relation with at least one attribute of some region data type. Based on the predicates, the functions and the primitive operations of the complex object model, it is demonstrated how operations of practical interest between maps can be achieved (see Figure 2.2). As an example, if A is a set of attributes of any type and G an attribute of an elementary region type, operation *Fusion*[A](R) (Figure 2.2(b)) can be achieved by a combination of the *Nest* operation and of the *Spatial union* of sets of regions.

2.2.22 ISO SQL Multimedia Standard: Spatial (ISOSQLMM)

The SQL:1999 standard includes *Object Relational* capabilities [MS02]. The SQL Multimedia and Application Packages (SQL/MM) [ME01] defines a set of *class libraries* of the SQL:1999 object types for the management of various types of complex data. The part for spatial data management [Iso02] includes a hierarchy of classes that enables the manipulation of 2-d spatial objects. Figure 2.5 shows the hierarchy of classes in UML notation. Some superclasses of the hierarchy, those in grey boxes, are not instantiable. Their definition provides compatibility between their subclasses. The ST_GEOMETRY consists of spatial objects of any type. ST_POINT represents single locations in a 2-d Euclidian space. ST_LINESTRING consists of simple polylines (g_4 in Figure 2.1(b)). ST_CIRCULARSTRING is

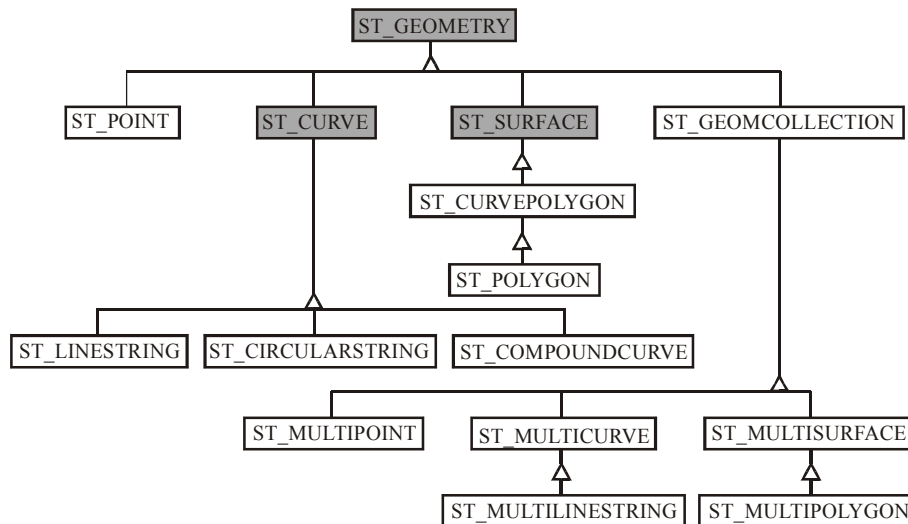


Figure 2.5: Hierarchy of classes supported by the spatial SQL/MM standard.

similar except that arcs are used instead of line segments. `ST_COMPOUNDCURVE` consists of connected sequences of elements of the two previous data types. `ST_CURVE` represents any kind of line. `ST_POLYGON` consists of vector polygons (Figure 2.1(c)). The boundary of a `ST_CURVEPOLYGON` is a collection of `ST_CURVES`. `ST_SURFACE` is a non-instantiable type that represents any kind of a 2-d surface. `ST_GEOMCOLLECTION` consists of collections of type `ST_GEOMETRY`. Subtypes of `ST_GEOMCOLLECTION` are collections of elements of the relevant atomic data types. Two surfaces in an element of type `ST_MULTISURFACE` may intersect only in a finite set of points. The empty set is a valid object of any of the previous data types. A long set of methods is defined in the classes of the above hierarchy. The inclusion of such methods in SQL statements enables the manipulation of spatial objects. The set of predicates *st_equals*, *st_disjoint*, *st_intersects*, *st_touches*, *st_overlaps*, *st_crosses*, *st_within*, and *st_contains* enable testing topological relationships between two spatial objects. Their definition follows the *Calculus Based Method* defined in [CFO93, CF94, CF96]. Method *st_boundary* yields the boundary of a spatial object. In the general case, the boundary of a `ST_SURFACE` is defined as set of `ST_CURVES` and the boundary of a `ST_CURVE` is defined as the possibly empty set of its end points. The boundary of an `ST_POINT` element is the empty set. The boundary of an `ST_GEOMCOLLECTION` is the spatial union of the boundaries of its elements. Given two spatial objects g_1 and g_2 , the methods *st_union*, *st_difference* and *st_intersection* enable computing their spatial union, spatial difference and spatial intersection, respectively. In the general case, the result

is a possibly empty element of type ST_GEOMCOLLECTION. Function *st_symdifference* returns $(g_1 \text{ st_difference } g_2) \text{ st_union } (g_2 \text{ st_difference } g_1)$. Additional functionality enables obtaining the *length* of lines, the *area* of surfaces, the *minimum Euclidian distance* of two spatial objects, the *gravity centre* of a spatial object, the *minimum bounding rectangle* enclosing a spatial object, the *buffer* of a spatial object, and testing for emptiness, simplicity and circularity (line without end points).

2.2.23 OpenGis Simple Features Specification for SQL (OGISSQL)

The OpenGIS Simple Features Specification for SQL [Ogis99] gives guidelines for the implementation of the abstract model specified in [Ogis01a] in two distinct approaches, (i) pure *relational* SQL92 and (ii) SQL2, extended by *object relational* capabilities. The set of classes and the behaviour of the methods defined in the OpenGIS Simple Features Specification for SQL is almost identical to that supported by the spatial part of the SQL/MM Standard, described in the previous subsection.

2.2.24 Oracle8i Spatial Cartridge (ORACLE)

The Oracle9i Spatial Cartridge [Ora00] extends the Oracle8i *object relational* DBMS with functions and procedures for spatial data management. Only one spatial data type is provided, SDO_GEOMETRY, whose definition follows the guidelines provided in the OpenGIS Simple Features Specification for SQL (Subsection 2.2.23). Contrary to the standard, the empty set is not a valid spatial object. A set of operations, predicates and functions enable the manipulation of spatial objects. Operations *Sdo_filter*, *Sdo_relate*, *Sdo_nn* and *Sdo_within_distance*, enable the use of spatial indexes for the retrieval of spatial objects that satisfy certain conditions. Such conditions enable testing topological relationships according to the *Nine Intersection Model* [EH92] and testing if an object is within a given distance from another. Beyond this functionality, a set of predicates and functions is also available that does not take advantage of the spatial indexes. Predicate *relate* enables testing topological relationships between two spatial objects. The functionality of functions *union*, *difference*, *intersection* and *xor* resembles, respectively, the relevant functions *st_union*, *st_difference*, *st_intersection* and *st_symdifference* of the spatial SQL Multimedia standard (Subsection 2.2.22). The difference is that when the empty set is to be obtained in the standard, a null value is produced in Oracle Spatial. Additional spatial functionality provided by other functions includes the computation of the *length* of lines, the *area* of surfaces, the *minimum Euclidian distance* of two spatial objects, the *buffer* of a spatial object and the *gravity centre* of a spatial object.

2.2.25 IBM Informix Spatial DataBlade (INFORMIX)

The IBM Informix Spatial DataBlade Module [Inf01] extends the *object relational* IBM Informix Dynamic Server with new data types, functions and index structures that simplify the manipulation of 2-d spatial objects. The system implements, to some degree, the SQL/MM standard (Subsection 2.2.22). However, heterogeneous collections of primitive spatial objects are not supported as spatial objects, since the data type `ST_GEOMCOLLECTION` is not instantiable. This leads to limitations and data loss in functions whose result is, in the general case, an heterogeneous set. Examples of such functions are *st_union*, *st_difference*, *st_symdifference* and *st_intersection*. The three first have to be applied to spatial objects whose *primitive atomic elements* are of the same data type. Their result type is the most appropriate relevant to the input data types. Function *st_intersection* can be applied to two spatial objects g_1 and g_2 of any data type. Its result data type is either `ST_MULTIPPOINT` or `ST_MULTILINESTRING`, or `ST_MULTIPOLYGON`, depending on whether the spatial intersection of g_1 and g_2 contains, respectively, only points or curves (and perhaps points) or surfaces (and perhaps curves and points). The aggregate function *st_dissolve* enables computing the spatial union of a group of spatial objects whose primitive atomic elements are of the same data type.

2.2.26 IBM DB2 Spatial Extender (DB2)

The IBM DB2 Spatial Extender [Ibm01] extends the IBM DB2 *object relational* DBMS with new data types, functions and index structures that enable the management of 2-d spatial objects. Its functionality is almost identical to that of the IBM Informix Spatial DataBlade Module (Subsection 2.2.25) except that the aggregate function *st_dissolve* is not supported.

2.2.27 PostgreSQL

PostgreSQL [Post01] is an Open-Source *Object Relational* DBMS, whose design is based on the Postgres system [SR86, RS87, SRH90]. An element of type `POINT` is defined as a pair of real coordinates (x, y). A `LINE` is an infinite straight line. An `LSEG` is a line segment linking two points. A `BOX` is a rectangle. A `PATH` is a simple vector polyline (g_4 in Figure 2.1(b)). A `POLYGON` is a vector polygon without holes (g_8 in Figure 2.1(c)). Many functions and predicates are supported as methods for the above spatial types. The functionality provided by such functions and predicates is very primitive. Some example functions return the intersection point of two line segments,

the intersection box of two boxes and a *path* defining the boundary of a polygon.

2.2.28 GEO++

The GEO++ system [VO92] is a GIS developed on top of the *object relational* DBMS Postgres [SR86, RS87, SRH90]. A set of spatial data types, and relevant methods, enables the manipulation of 2-d spatial objects. One more data type enables the storage of images in a raster format. Functions for the manipulation of raster images are not provided. The set of spatial data types is almost identical to that of PostgreSQL (Subsection 2.2.27). Regarding spatial functionality, GEO++ includes a function that computes the spatial intersection of two polygons. The result is a set of polygons.

2.2.29 GEUS

The GEUS system [PLL+98] has been developed by extending a commercial *object relational* DBMS with new spatial data types, operations and spatial indexes (R^* -tree). The behaviour of the system is very close to that of GEO++ (Subsection 2.2.28). No functions are provided for the computation of spatial union, intersection, difference and boundary.

2.2.30 Object-Oriented Geographic Data Model (OOGDM)

An *object-oriented* data model for the management of 2-d and 3-d spatial data in both vector and raster format is defined in [Vo97, BVH96, DBVH97, VBH97]. A hierarchy of classes is defined that supports the representation and management of spatial data. Such a hierarchy of classes, in UML notation, is shown in Figure 2.6. *Elementary features* are vector or raster objects in either a 2-d or a 3-d space. Vector classes represent points (Figure 2.1(a)), simple polylines (Figure 2.1(b)) and polygons (Figure 2.1(c)). Furthermore, class *Solid* represents 3-d polyhedra. Raster elements in both a 2-d and a 3-d space are represented by classes *Profile*, *Grid* and *Lattice*. A *Feature* is either elementary or a collection of features. Features incorporate, therefore, connected atomic spatial objects and non-connected set-valued spatial objects. A *GeoObject* combines a set of non-spatial properties with a collection of at least one feature. User classes incorporating spatial functionality are defined as subclasses of *GeoObject*. Finally, a *SpatialObject* is either a feature or a *GeoObject*. Functions and predicates that enable the manipulation on spatial data are defined as methods of classes in this hierarchy. The definition of predicates, to test topological relationships between spatial objects, follows

the *Calculus Based Method* presented in [CFO93, CF94, CF96]. Predicates to test directional relationships are also provided and their definition follows the approach in [PTS94, TPS96]. Operations *touch*, *overlap* and *cross* enable obtaining parts of the spatial intersection of two spatial objects for which a relevant predicate evaluates to true. Two polygons *touch* if their intersection contains only points and/or polylines. A polyline and a polygon *touch* if they intersect only at the end points of the polyline. Finally, two polylines *touch* if they intersect only at their end points. Two polygons *overlap* if their intersection contains at least one polygon. Two polylines *overlap* if their intersection contains at least one polyline. A polyline and a polygon *cross* if their intersection contains at least one polyline. Finally, two polylines *cross* if they do not *touch* and their intersection contains only isolated points. The *boundary* of a point is the empty set. The *boundary* of a polyline is the set of its end points and the *boundary* of a polygon is a set of polylines. Further spatial functionality includes the computation of the *length* of polylines, the *area* of polygons, the *minimum Euclidian distance* of two spatial objects, the *gravity center* of a spatial object, the *minimum bounding rectangle* of a spatial object, the *holes* of a polygon, and the *buffer* of a spatial object. Implementation issues are described in [VBH97].

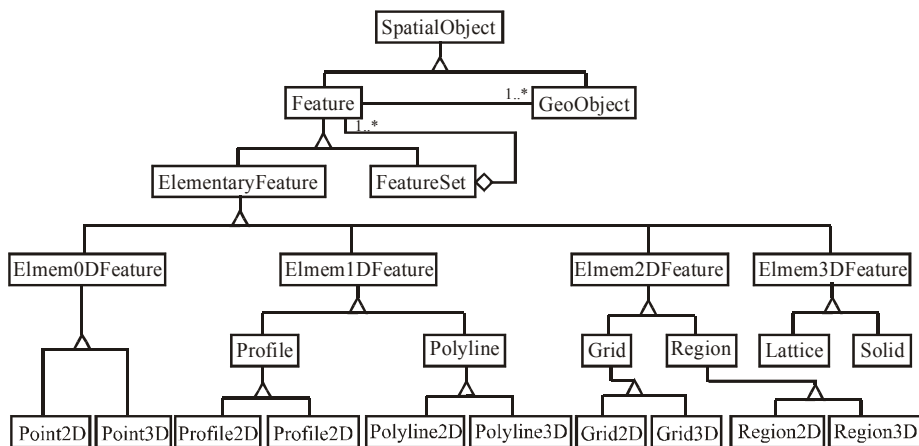


Figure 2.6: Class hierarchy supported by the OOGDM approach.

Other object-oriented models are [CF93], concerning an object calculus, DASDBS [SW86, ELNR88, WS92], GEOQA [SDS00], GODOT [GR93], PROBE [MO86], GeoToolKit [BBC97] and LEGAL [CCF+96, CSFG96].

2.3 Previous Research in Spatio-temporal Approaches

Many models have been proposed for the management of temporal data [JM80, Be82, CT85, Ar86, Ta86, Sn87, Ga88, LJ88a, LJ88b, TG89, Sa90a, Sa90b, TC90, MS91, JJ92, Lo93, CC93, NA93, TCG+93, Sn95, EJS98, CZ99]. A relevant potential SQL standard has also been proposed in [Iso96a]. Some of the characteristics of these models have been incorporated in the spatio-temporal approaches that are briefly described below.

2.3.1 Güting et al Spatio-temporal Model (G+STM)

A *spatio-temporal object model* for the representation and management of moving objects is defined in [GBE+00, FGNS00, CFG01]. Time is defined as infinite and continuous.

Operation	Type of First Operand	Type of Second Operand	Type of Result
<i>Intersection</i>	POINT	ANY	POINT
	ANY	POINT	POINT
	LINE	LINE	LINE
	LINE	REGION	LINE
	REGION	LINE	LINE
	REGION	REGION	REGION
<i>Common_border</i>	REGION	REGION	LINE
<i>Crossings</i>	LINE	LINE	POINTS
<i>Touch_points</i>	LINE	REGION	POINTS
	REGION	LINE	POINTS
	REGION	REGION	POINTS

Figure 2.7: Operations for spatial intersection in G+STM.

Data type INSTANT consists of *time points*. A PERIOD is the union of a set of non-overlapping and non-adjacent time intervals, where each time interval is a closed or open set of successive time points. PERIOD actually matches RANGE(INSTANT), where RANGE is a generic data type. Data types POINT, POINTS, LINE and REGION consist, respectively, of points, sets of points, sets of lines and sets of surfaces. Spatial objects are closed. If S is a spatial data type, then data type MOVING(S) describes the continuous evolution with respect to time of elements of type S. INTIME(S) is another

data type, with elements of the form (s, t) , which describes a spatial object s and its position at time t . Similar data types MOVING(D) and INTIME(D) are defined for a conventional data type D. A long set of primitive operations is formalized between elements of the previous data types. Amongst them, operations *Union*, *Minus* and *Intersection* can be applied to either time periods or spatial objects. For time periods, the functionality is that of the relevant set operation. Regarding spatial data, *Union* yields the spatial union of two spatial objects of the same type. *Minus* gives the spatial difference of two spatial objects. The data type of the result is that of the first operand and an implicit *Closure* is applied to obtain this result. Four different operations enable retrieving the different pieces of the spatial intersection of two spatial objects, as is shown in Figure 2.7. Operations *Union* and *Intersection* can also be used as aggregate functions. The operations supported for a data type D can be *lifted* so as to be applied to combinations of data types D and MOVING(D). As an example, operation *Intersection* can also be applied to data types MOVING(POINT) and REGION to produce a value of type MOVING(POINT). Finally, operation *Decompose* enables decomposing a non-connected spatial object or a time period into its connected components. The design and implementation of a prototype is left for further research.

2.3.2 Moreira, Ribeiro and Abdessalem's Spatio-temporal Model (MRASTM)

A classification of the set of operations required in a *spatio-temporal object model* for the management of moving objects is proposed in [MRA00, MSR99]. Data types for time and space are not formalized. A *movement* is defined as an infinite set of elements of the set $T \times G$, where T is a set of time instants and time intervals and G is a set of points, lines and polygons. A *movement value* is a finite representation of a movement. Such finite representations [MSR99] are finite sets of tuples, each of them describing the movement within a time interval. *Projections* are operations that enable extracting from a movement its period of time, its trajectory and numeric values derived from the movement, such as average speed. *Restrictions* enable retrieving pieces of a movement when a temporal, a spatial or a numeric predicate is satisfied. *Metric* operations enable retrieving maximum, minimum and moving distances of movements and static spatial objects. Finally, the *topological-temporal* predicates *enter*, *leave* and *cross* describe the relationship between a movement and a static spatial object. The approach incorporates uncertainty.

2.3.3 Worboys's Spatio-temporal Model (WSTM)

A *spatio-temporal object model* with operations that enable the management of spatial objects that change discretely with respect to time is defined in [Wo94]. *Bitemporal elements* are defined as unions of Cartesian products of intervals of transaction and valid time. Set operations, union, difference and intersection can be applied between bitemporal elements, always resulting in a bitemporal element. *Spatial objects* are defined as collections of points, non-overlapping straight line segments and non-overlapping triangles. If a triangle belongs to a spatial object, then the edges and vertices of the triangle also belong to the object, and are recorded separately. The same also applies to line segments. Set operations *Union*, *Difference* and *Intersection* can also be applied to spatial objects. Operation *Boundary* is presented informally. A formalism for spatial and temporal operations are out of the scope of the approach. *Spatio-temporal objects* are defined as finite sets of pairs $ST = \{(S_1, T_1), (S_2, T_2), \dots, (S_n, T_n)\}$, where each S_i is either a point or a straight line segment or a triangle and each T_i is a bitemporal element. The set of S_i values must form a spatial object and for each pair of tuples $(S_i, T_i), (S_j, T_j)$ in ST , if S_i is contained in S_j , then T_i is also contained in T_j . Operations relevant to the work in the present thesis are those that enable retrieving the evolution with respect to time of the spatial union, spatial difference, spatial intersection and spatial boundary of spatial objects. Implementation issues are not discussed.

2.3.4 Erwig and Schneider's Spatio-temporal Partition Model (ESSTPM)

A *temporal map model*, resulting from the generalization of a map model (Subsection 2.2.3) is presented in [ES99]. In particular, the model defines a new data structure that extends by a temporal dimension, a previously defined data structure for the representation of a special kind of maps (*partitions of space into non-overlapping surfaces*). A *spatio-temporal partition of type A* is formalized as a kind of a semi-continuous mapping P from the temporal domain T (infinite and continuous set of time instants) to the set of spatial partitions of type A (see Subsection 2.2.3). Thus, each time instant has an associated spatial partition. The primitive operations *Intersection* and *Relabel*, already defined for spatial partitions (see Subsection 2.2.3), are generalized for spatio-temporal partitions. Informally, the *Intersection* of two spatio-temporal partitions is the result of applying the *Intersection* operation to spatial partitions at each point in time. Operation *Relabel* enables applying a function, whose definition changes with respect to time, to a spatio-temporal partition of type A in order to obtain a spatio-temporal partition of type B ($f_{t \in T} (a \in A) = b \in B$).

2.3.5 d’Onofrio and Pourabbas’s Temporal Map Model (OPTMM)

A *temporal map model* is presented in [OP01]. Its functionality is very close to that of the previous approach, therefore, only the main differences are given here. The approach is not formal. A *thematic map* is defined as a mapping from a finite set of surfaces to a set of either colours or strings. The concept of thematic map is also generalized for lines. Relevant to the work of the present thesis, operation *Fusion* enables merging adjacent surfaces that have the same colour. Operation *Intersection* can be applied to a pure surface or a set of pure surfaces and to a thematic map. A *temporised thematic map* is defined as a set of pairs $\{(I_1, T_1), (I_2, T_2), \dots, (I_n, T_n)\}$, where each I_i is a time interval and each T_i is a thematic map. Operations *Fusion* and *Intersection* are generalized so as to apply to temporised thematic maps.

2.3.6 Tryfona and Hadzilacos’s Temporal GIS Approach (THTGIS)

In [TH98] the *GIS* model for spatial data, which has been described in Subsection 2.2.4, is extended so as to apply to spatio-temporal data. Both relations and layers are extended to their temporal counterparts. Temporal relations and temporal layers are extended by a pair of implicit attributes, one for a period of valid time and another for a period of transaction time. Definitions are missing. Beyond spatial constraints, temporal constraints can also be used for the construction of objects in queries.

2.3.7 Kemp and Kowalczyk’s Temporal GIS Approach (KKTGIS)

The design and implementation of a *temporal GIS* on top of Postgres [SR86, RS87, SRH90], is described in [KK94]. Regarding spatial data, the approach is similar to that described in Subsection 2.2.8. Spatial data is recorded separately from conventional data in four system tables, namely POINTS, LINES, AREAS and AREA_LINK. The use of attributes of type period enables recording the evolution of both conventional and spatial data with respect to time. The current versions of the data are maintained in user tables, whereas past versions are recorded in system tables called *history tables*. Figure 2.8 shows the set of tables needed to represent the evolution of both spatial and conventional data with respect to time, for an application of HOTELS, ROADS and land OWNERSHIP. Implicit transaction time support is directly provided by Postgres. Spatio-temporal data management is achieved by the use of the object-relational query language of Postgres. Other pieces of work, addressing issues related to the incorporation of versions in geographic information systems, are discussed in [La92, NTE92, WH94, MJ94, RK98].

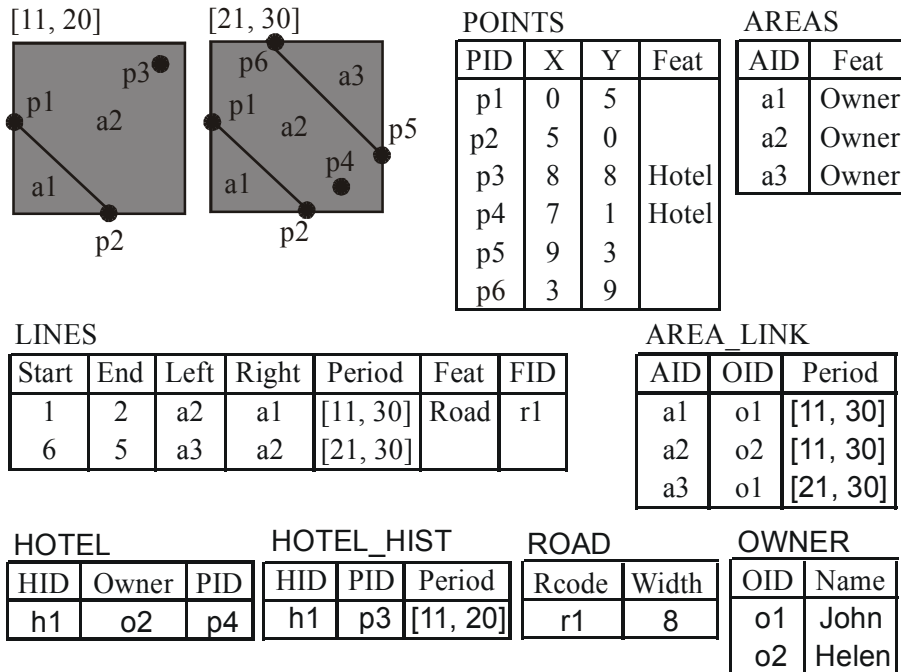


Figure 2.8: Spatio-temporal data in a temporal GIS.

2.3.8 STSQL

An SQL extension for the management of spatio-temporal data is described in [BJS98], and it is a generalization of an earlier extension for temporal data [BJ96]. Only one data type PERIOD is provided for time data. Data types 1D_REGION, 2D_REGION and 3D_REGION enable the representation of spatial data. Definitions are not provided. Predicates and functions are out of the scope of the approach. Attributes of the above data types can be incorporated in relations in two possible ways, as *explicit* attributes, treated the same way as conventional attributes and as *implicit* attributes. Implicit PERIOD attributes are used for valid and transaction time. Implicit XD_REGION attributes are used to record the set of XD points of a spatial object. Explicit attributes are manipulated using standard SQL:1992 statements, possibly by the incorporation of spatial and temporal predicates and functions. Such statements apply implicitly to tuples that are valid at the present time. Implicit spatial attributes are ignored (projected out) before a conventional SQL:1992 statement is executed. To use implicit attributes in queries, a prefix has to be added before an SQL statement. For an example, consider the scheme of two relations $R_1(\mathbf{A}, G_1, T_1)$ and $R_2(\mathbf{B}, G_2, T_2)$, where

A, **B** are two sets of explicit attributes of any type, G_1 , G_2 are two implicit attributes of a spatial type and T_1 , T_2 are two implicit attributes of time type. Then the effect of the syntax

```

REDUCIBLE ( $R_1.G_1$ ,  $R_2.G_2$ ) AS G
REDUCIBLE ( $R_1.T_1$ ,  $R_2.T_2$ ) AS T
SELECT  $R_1.A$ ,  $R_2.B$ 
FROM  $R_1$ ,  $R_2$ 

```

is that the SELECT-FROM statement is executed for every time instant and for every point of each spatial object. In a next step, the resulting tuples are combined to tuples in which explicit attributes are stamped by the values of implicit attributes. This way, the previous SQL statement computes the spatio-temporal intersection of spatial objects in relations R_1 and R_2 . On the other hand, implicit attributes can be treated as explicit by adding the statement NONREDUCIBLE (<implicit attribute list>) before the relevant SQL statement. Definitions are left for further research as they are issues related to implementation. As is reported, the temporal SQL extension has been implemented.

2.3.9 SQLST

In [CZ00] an SQL extension for the management of spatio-temporal data is described, which is again a generalization of an earlier extension for temporal data [CZ99]. A special attribute called VTime of type *time instant* enables the storage of valid time. Time instant allows various granularities, such as DAY, TIME, etc. The use of the SQL clause GROUP BY enables merging time instants into periods. Next, functions and predicates for periods can be included in the SELECT and HAVING clauses of SQL [A183]. The approach for spatial data is almost the same. Data types POINT, LINE and REGION are defined and their elements are always composed of triangles. Hence a point is a triangle whose vertices are co-located. Similarly, a line is a triangle whose vertices are collinear. This enables compatibility between spatial types. The GROUP BY clause enables merging triangles into spatial objects. Spatial predicates, such as *equal*, *overlap*, etc. and functions, such as *intersection*, can be used on the SELECT and HAVING clauses. Yet defining predicates and functions is out of the scope of the approach. An implementation is also reported. Internally, time intervals are used instead of time instants. It is reported that further research includes the implementation of the spatial approach with a vector representation and the efficient implementation of spatial objects whose position and shape changes continuously with respect to time.

2.3.10 Dedale

A *nested constraint-based model* for the management of multidimensional spatial and spatio-temporal data is described in [GRS98b, GRS00]. Actually, the approach extends the work described in Subsection 2.2.18 by *linear constraint relations* in which time is also incorporated.

2.3.11 Yeh and de Cambray's Spatio-temporal Model (YCSTM)

A *complex object model* for the management of both discrete and continuous changes in conventional and spatial data is described in [YC95]. It is also based on an earlier temporal approach [YC94b]. Regarding spatial data, the model provides just one GEOMETRY data type, as in the approach described in Subsection 2.2.10. Temporal data is recorded in *variability tuples* containing two implicit attributes, (i) a surrogate tuple identifier and (ii) a closed time interval. The remainder attributes of the tuple are called *variability attributes*. Variability attributes are used to record the continuous evolution with respect to time. At a high level of abstraction, variability values can be seen as infinite sets of *states* of the form (*time instant, value*). At a lower level of abstraction, they are recorded as the values at the end points of a time interval and as the definition of an interpolation function. Internally, a 2-d spatial variability value is represented by a 3-d geometric object. Geometric operations between 3-d spatio-temporal objects are used. Relational algebra is extended so as to handle sets of variability values. *Cartesian product* yields tuples with more than one pair (*surrogate, time interval*). Each variability value of a variability tuple is associated with just one such pair. When a variability value is *projected*, the relevant (surrogate, time interval) is also projected. Relational *Union*, *Except* and *Intersect* operations are applied to states of variability tuples. *Selection* is extended to yield the set of states for which a given predicate evaluates to true. Finally, *Temporal projection* enables obtaining the set of states within a time interval.

2.3.12 Informix Geodetic DataBlade (GEODETTIC)

The IBM Informix Geodetic DataBlade Module [Inf00] extends the *object relational* IBM Informix Dynamic Server with new data types, functions and index structures that simplify the manipulation of spatio-temporal data. A value of a spatio-temporal data type is defined as a tuple of the form (s, a, t), where s is a spatial object, a is an *altitude range* and t is a *time range*. Depending on the type of the spatial object s, various spatio-temporal data types are defined. Some of them are the following: (i) GEOPOINT, which represents points (g_i in Figure 2.1(a)). (ii) GEOSTRING, which represents

simple polylines (g_4 in Figure 2.1(b)). (iii) GEOPOLYGON, which represents polygons, possibly with holes (g_i in Figure 2.1(c)). Other spatio-temporal data types include GEOBOX, GEOCIRCLE, GEOLINESEG, etc. A general and non-instantiable data type, GEOOBJECT, represents spatio-temporal objects of any type. A time range is either a time instant or a time interval. The value *any* denotes a time range that consists of all the time instants. Similar observations also apply to altitude ranges. A set of predicates and functions is provided, which enable the management of spatio-temporal data. Predicate *intersect* evaluates to true if the spatial objects, time ranges and altitude ranges have points in common. Predicates *inside* and *outside* are defined in a similar manner. Certain functions return the *area* of polygons, the *gravity centre*, the time and altitude ranges, the *buffer*, etc.

2.3.13 ORParaDB

The approach undertaken in [CG94] proposes a pattern matching query language for spatio-temporal databases in the context of an *object relational* model (ORParaDB). The language extends the SQL-like query language ORParaSQL, defined for parametric databases [CGN93], special cases of which are temporal, spatial and conventional databases. In a *parametric database*, relations are either conventional or parametric. In a parametric relation (object type in the object-relational terminology), each attribute is defined as a mapping from a set of *parametric elements* to the values of a conventional data type. *Parametric elements* are defined as subsets of a given *parametric domain*, and at the same time, they are called the *parametric domain* of the attribute. Examples of parametric domains are the set T of all time instants, the set R of all spatial objects and the set T×R of all spatio-temporal elements. Parametric elements are closed under the set operations *Union*, *Intersection*, *Difference* and *Complementation*. The parametric domains of values of a given tuple must be the same. This requirement is termed *homogeneity*. Relational operations *Union*, *Except* and *Intersection* are applied to the parametric elements of tuples. *Projection* and *Cartesian product* are defined the usual way, except that, due to the homogeneity property, the domain of each tuple, in the result relation, equals the intersection of the domains of the tuples, from which it is deduced. *Select* retrieves *portions* of tuples.

2.3.14 Temporal Object-Oriented Geographic Data Model (TOOGDM)

The approach described in [Vo97, BVH96] extends the object-oriented spatial data model presented in Subsection 2.2.30 for the management of spatio-

temporal data. Valid time Timestamps of type either time instant or time period (opened or closed) can be incorporated at the level of attribute. Only periods for transaction time are supported at the tuple level. The object-oriented query language is extended by (i) a VALID TIME clause, which enables the restriction of the result to a valid time interval, (ii) a SNAPSHOT keyword, which enables projecting out timestamps from the result and (iii) predicates and functions for time intervals in the WHERE clause.

2.3.15 Tripod

Within the framework of an *object-oriented* data model, a set of primitive data types and operations for the management of both valid time and spatial data is defined in [GFP+01a, GFP+01b, GFDP01]. In addition, a specific class, called *History*, provides support for transaction time. The set of primitive spatial data types follows the approach described in Subsection 2.2.1. INSTANT, INSTANTS, TIMEINTERVAL and TIMEINTERVALS represent, respectively, instants, finite collections of instants, closed time intervals and finite collections of closed time intervals [GFDP01]. Operations *Union*, *Difference* and *Intersection* are defined for time types. Regarding transaction time, if *D* is either a primitive data type or class, *History(D)* is a finite set of pairs of the form $\{(d_1, t_1), \dots, (d_n, t_n)\}$, where each d_i is an instance of *D* and t_i are values of some time data type. Each d_i and t_i is required to be unique in the history. Operations between histories include *Union*, *Intersection* and *Difference*.

2.3.16 MOST

The M.O.S.T. (Moving Object Spatio-Temporal) approach [SWCD97] proposes an *object-oriented* model for the modelling of the past, present and future locations of moving objects in a database system. A *future temporal logic* is used as a query language. The coordinates of spatial objects are recorded in conventional attributes. A *dynamic attribute* is used to record the evolution of the relevant data from a time instant *t* to the future. Internally, the evolution is represented by an initial value, an initial time instant and a function for time that computes future values. Valid and transaction time are considered to be the same in the approach. Examples of spatial predicates are those that enable testing whether a point is inside or outside a polygon. By default, queries apply to the data that is valid at the present time. Temporal predicates enable testing whether a given non-temporal predicate evaluates to true at specific future time instants. If *g* and *f* are predicates then two temporal predicates are the following: (i) “*g until f*”, which evaluates to true if *f* is true at some time in the future and until then, *g* evaluates to true. (ii) *Nexttime g*,

which evaluates to true if g evaluates to true in the next time instant. Other temporal predicates can be defined in terms these two. Queries regarding the past are left as open for further research. Issues concerning the integration of the model in a commercial DBMS are discussed in [WXCJ98].

2.4 Classification of Spatial Approaches

In this section spatial approaches are classified with respect to general characteristics.

2.4.1 Discrete versus Continuous Change in Space

Two different approaches in spatial data modelling have been identified [Go92, LGMR01, Wo95, RSV02, LT92, BM98], *Object-based* and *Field-based* models.

Object-based models view geographic space as being populated by discrete *spatial entities* (*spatial objects* in [PT98]). Spatial objects are classified with respect to their geometric representation into various types such as *point*, *line* and *surface*. For example a well, a river and a country are of a point, line and surface type, respectively.

Field-based models view spatial data as mappings from the points of a geographic space to the domain of some *property of space* (*spatial attribute* in [PT98]). The domain of a property of space can be either discrete or continuous. Examples of a discrete attribute are the type of soil and the type of vegetation. Continuous properties of space are temperature, atmospheric pressure and elevation.

The functionality required for the management of *data that changes discretely in space*, i.e. of *spatial entities* and of *discrete properties of space* (Figure 2.2), is different than that for the management of *data that changes continuously in space* i.e. of *continuous properties of space* (Figure 2.3). Different data structures are also considered in the two approaches. Models of the first approach are [GS95, DHT94, ES97, FVM97, VE93, HT96, Esri00, Int02, Mapi02, Bent01, Int95, LPV93, Eg94, RFS88, SV92, GNT91a, Gu88, CZ96, Ro93, SV89, Iso02, Ogis99, Ora00, Inf01, Ibm01, Post01, VO92, PLL+98]. Models of the second approach are [To91, Ogis00, MJ01, Keig02, Gras02, Lo00, BK01, RHS01].

Some approaches attempt the integration of the two. Hence some commercial geographic information systems [Esri00, Int02], which support the management of spatial objects, provide extensions for the management of

properties of space [MJ01, Keig02]. To bridge the gap, data transformations have also to be performed. In [Wo95], it is discussed how some object-oriented models [Vo97, ELNR88, SDS00, CCF+96] provide a different set of data types and operations to support each of the approaches. In [SH91] both types of data are integrated in the same data structure of a many sorted algebra. Finally, in [GRS00] it is shown how constraint-based spatial models [GRS98a, KRSS98], initially developed for the manipulation of spatial objects, can also be used for the manipulation of continuously changing interpolated spatial data.

2.4.2 Raster-based versus Vector-based Representation of Space

The finite representation of spatial data can roughly be classified into *raster-based* and *vector-based* (for a finer classification see [Pe90, LT92]).

In *raster* representations, space is partitioned into a finite grid of cells (usually squares, which are called *pixels*) of the same size and shape. The geometry of a spatial object is approximated by a set of cells [GNT91a, RFS88]. The representation of a *property of space* can be achieved by associating a value of a given domain to each of these cells [Ogis01b, MJ01, Keig02, Gras02, Lo00, BK01, RHS01, Vo97, ELNR88, SDS00, CCF+96].

Vector-based representations describe the boundaries of *spatial entities* and of *discrete properties of space* by the use of finite sets of points and of line segments [GS95, DHT94, HT96, Esri00, Int02, Mapi02, Bent01, Int95, LPV93, Eg94, RFS88, SV92, Gu88, SH91, CZ96, Iso02, Ogis99, Ora00, Inf01, Ibm01, Post01, VO92, PLL+98, Vo97, ELNR88, SDS00, CCF+96]. One typical vector representation, which is concerned with *continuous properties of space*, is the partition of space into a finite set of triangles called *Triangulated Irregular Network* (TIN). In this representation the value of a property in a given point is computed by an interpolation function that is applied to the vertices of the triangle that contain the point.

Issues related to the advantages and disadvantages of the two types of representations can be found in [Pe90, LT92, BM98, RSV02] and are outlined as follows:

- *Vector-based representations* are largely used for the management of data that changes discretely in space. On the other hand, *raster-based representations* are still used for the management of *continuous properties of space*. This is not likely to change, due to the fact that the growing volumes of data, which are received from satellites, are available in *raster* format.
- *Vector-based representations* achieve a high precision by few resources, but they make use of complex algorithms and data structures.

On the other hand, *raster-based* representations achieve a lower precision by the use of big volumes of data, but they make use of simple algorithms and data structures.

2.4.3 GIS-centric versus Database-centric Spatial Data Management

Relevant to the architecture of the underlying system, spatial data modelling approaches can be classified into *GIS-centric* and *Database-centric* [Da98].

GIS-centric architectures provide data structures and operations for the direct manipulation of *maps*, also known as *layers*, *coverages* or *feature classes*. They consider either vector-based (Figure 2.2) or raster-based (Figure 2.3) representations. Map algebras are defined in [To91, DHT94, ES97]. In [HT96], a layer algebra is combined with a relational algebra within a single GIS-centric architecture. One effort for a standard coverage type has been undertaken by the OpenGIS consortium [Ogis00]. Commercial systems that are based on raster maps are [MJ01, Keig02, Gras02, Lo00, BK01, RHS01]. Regarding vector maps, the first GIS-centric implementations were based on CAD tools, which support only graphic data, and attempted to link graphic objects with conventional data stored in separate files. Such systems were missing the majority of the advantages provided by the use of database technology. The architecture of current GIS-centric systems is either *layered* or *dual* [VO92, LPV93, Gu94, SA95, SS96].

In a layered architecture both conventional and spatial data are stored in a conventional DBMS, thus, they take advantage of the database technology. However, optimised spatial data management cannot be achieved by the DBMS, since spatial data are treated by the DBMS as bit strings. Such systems are Esri ArcInfo [Esri99, Esri00, Ze99, Ma99, Tu00, MSW99, Esri02a, Esri02b], Intergraph Geomedia [LH98, Int02] and MapInfo Professional [Mapi01, Mapi02].

In a dual architecture, conventional data remains in the database but one specialized subsystem is considered for the management of spatial data. The spatial subsystem enables the optimisation of spatial data access. Such systems are Bentley Microstation Geographics [Bent01] and Intergraph MGE [Int95].

Although GIS-centric approaches are very flexible in the management of maps, they do not take full advantage of database technology. Typical examples are the lack of a powerful generic query language that combines both spatial and non-spatial data (therefore excess programming is required) and the lack of data independence. According to [DA98], GIS-centric approaches form the first two generations of spatial management systems.

Database-centric approaches form the third generation of spatial management systems [DA98]. Their inherent characteristic is that facilities for spatial support are integrated within a DBMS, whose underlying data model ranges from the simple pure relational model to the complex object-oriented. The management of spatial data is usually achieved by the use of spatial predicates and functions that are applied to data of a spatial type. Relevant research approaches and prototype implementations include [GS95, LPV93, Eg94, RFS88, SV92, GNT91a, SH91, Gu88, CZ96, GRS98a, KRSS98, Ro93, SV89, VO92, PLL+98, Vo97, ELNR88, SDS00, CCF+96]. An open source product of this sort is PostgreSQL [Post01]. Commercially available products are the spatial extensions of ORACLE [Ora00], INFORMIX [Inf01] and DB2 [Ibm01]. Standards for an SQL extension are the OpenGIS Simple feature specification for SQL [Ogis99] and the spatial part of the ISO SQL Multimedia [Iso02]. Database-centric approaches take full advantage of database technology. Due to the fact that spatial functionality is integrated within the DBMS, queries that combine spatial and conventional data can also be optimised. On the other hand, DBMS-centric approaches lack the flexibility of GIS-centric in the management of maps. As an example, map operations like *Overlay* or *Superimposition* (Figure 2.2) are not supported by many such approaches [Eg94, RFS88, SV92, Gu88, CZ96, Iso02, Ogis99, Ora00, Ibm01, Post01, VO92, PLL+98, Vo97].

2.5 Remarks on Spatial Approaches

A number of comments are now made on the various spatial approaches. The comments have primarily their origin in the many distinct results of an operation between spatial objects. Such representative operations are *spatial union*, *spatial difference* and *spatial intersection*. As can be seen in Figures 2.9-2.11, the result of any of these operations may yield zero, one or more spatial objects, and there are cases in which the type of the resulting spatial object does not even match the type of the objects involved in the operation. Particular effort has been made for the remarks to be objective. Throughout the remainder of this thesis, discussion restricts to only 2-d spatial objects unless otherwise is explicitly stated.

2.5.1 Informal Data Types and Operations for Space

Formalism is missing in many approaches and, instead, either descriptions or examples of the data types and the operations between them are given [To91, HT96, LPV93, Eg94, RFS88, SV92, SH91, Gu88, CZ96, Vo97]. As a consequence, it is not easy to determine precisely which are the valid data types and which is the precise functionality of an operation.

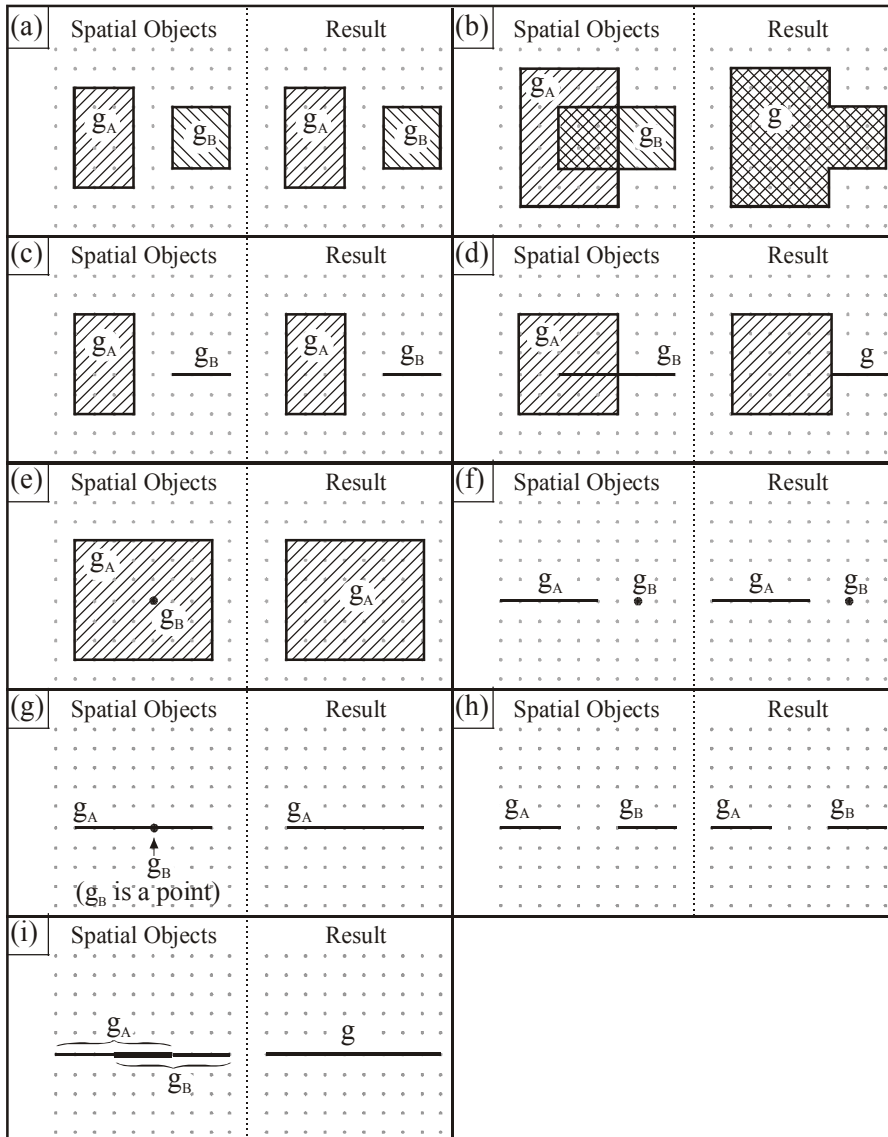


Figure 2.9: Examples of spatial union of g_A with g_B .

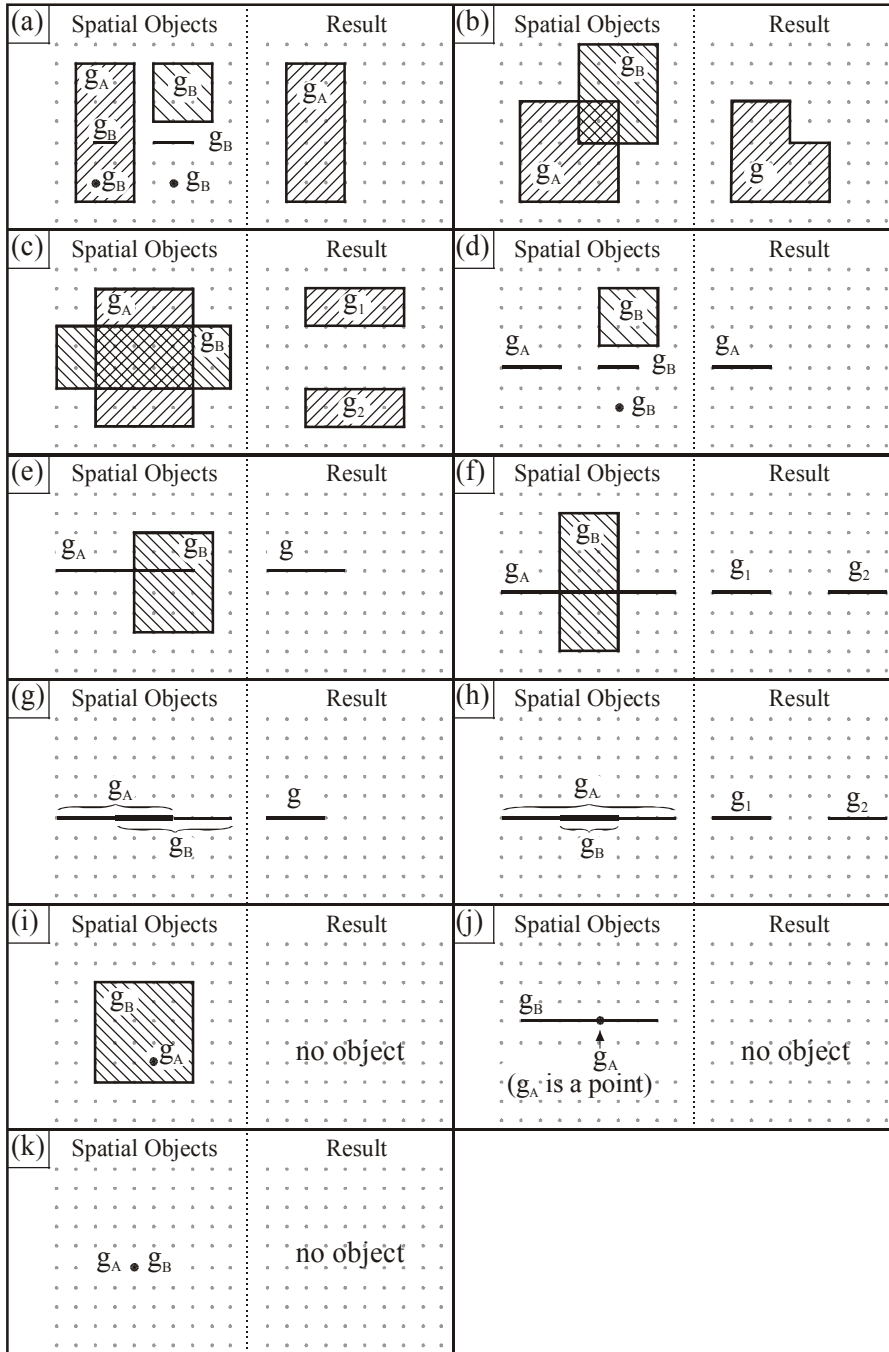


Figure 2.10: Examples of spatial difference of g_B from g_A .

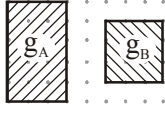
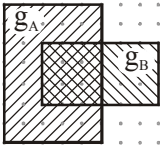

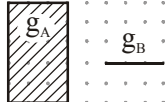
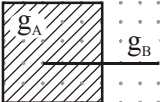

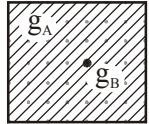


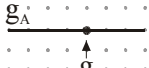


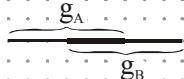

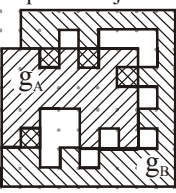
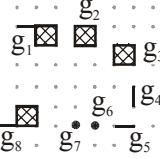
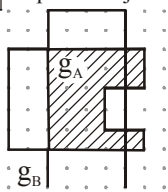
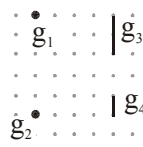
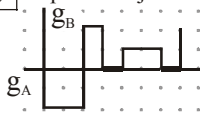
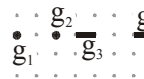
(a)	Spatial Objects 	Result no object	(b)	Spatial Objects 	Result 
(c)	Spatial Objects 	Result no object	(d)	Spatial Objects 	Result 
(e)	Spatial Objects 	Result 	(f)	Spatial Objects 	Result no object
(g)	Spatial Objects  (g_B is a point)	Result 	(h)	Spatial Objects 	Result no object
(i)	Spatial Objects 	Result 	(j)	Spatial Objects 	Result 
(k)	Spatial Objects 	Result 	(l)	Spatial Objects  g_A is horizontal	Result 

Figure 2.11: Examples of spatial intersection of g_A with g_B .

2.5.2 Spatial Data Types Deviating from Human Perception

According to [TL82] a data model must enable an accurate mapping of the real world. Contrary to it, the spatial data types that are considered in various approaches deviate from human perception. Four distinct such cases are enumerated below.

1 No Support of User-friendly Data Types for Space

In daily practice people consider three primitive *types* of spatial objects, *points*, *lines* and *surfaces*. Given however the many distinct results of representative operations between spatial objects (Figures 2.9-2.11), various approaches fail in defining user-friendly spatial data types. Some of them support only one generic spatial data type [To91, Bent01, Int95, LPV93, GRS98a, KRSS98, Ro93, SV89, ORA00]. Others restrict, in principle, to supporting only surfaces [ES97, FVM97, VE93, GNT91a, MJ01, Keig02, Gras02, Lo00, BK01, RHS01]. Others define *spatial sets* rather than *spatial elements* [GS95, Esri00, Int02, Mapi02, SV92, ELNR88]. Typical examples are *sets of points*, *sets of lines*, *sets of surfaces* or *sets of various combinations of points, lines and surfaces*. Finally, in all vector-based approaches, the user has to think in terms of arcs, when defining lines and surfaces.

2 Incompatibility Between spatial Data Types

In daily practice people tend to consider a point as either a degenerate line or as a degenerate surface. Similarly, they consider a line as a degenerate surface. However, the *compatibility* between spatial types, which is implied by these observations, is not considered in the definition of spatial types. Due to this, database design and manipulation problems are encountered. To make them clear, two examples are first given.

Example 2.1: A meteorological information system usually requires the study of physical phenomena, which are identical in nature, yet their geometric shape varies. One such example is temperature, which may have to be recorded either for isolated points (cities) or for lines (connecting cities with the same temperature) or for surfaces (regions).

Example 2.2: Spatio-temporal data models aim at modelling spatial data whose geometry changes with respect to time. However, there are sorts of spatial objects whose *data type* also changes with respect to time. One typical example is that a spring, originally recorded as a point, may change to a river (to be recorded as a line), then to a lake (to be recorded as a surface) and finally shrink to a point again.

Given now that spatial data compatibility has not been addressed satisfactorily, three distinct data structures have to be used for the recording of the spatial data in the above examples in [GS95, Esri00, SV92, SH91, Gu88, Post01, VO92, PLL+98, ELNR88]. Subsequently, even a simple display of spatial data requires referring to all these structures.

3 No Support of Set-Theoretically Closed Spatial Objects

All non-computerised spatial applications (e.g. cartography, topography, cadastral systems etc) consider *closed* spatial objects. Informally, this means that lines with missing points (whether end points or not) are not allowed. Similarly, surfaces with missing points or lines are not allowed either. Yet, this property is not satisfied in [Ro93, SV89]. Note that the support of open or closed spatial objects is left as an open issue in [SV92].

4 Support of the Empty Set as a Valid Spatial Object

As can be seen in Figure 2.10(j), the spatial difference of two pieces of spatial data may not yield any spatial object. Due to this, the empty set is treated as a valid spatial object in [GS95, Esri00, LPV93, Eg94, SV92, GNT91a, KRSS98, SV89, Iso02, Ogis99, Inf01, Ibm01, Vo97]. In practice, however, it does not make too much sense to consider an *empty* spatial object. Alternatively, a null value replaces the empty set in [Ora00].

2.5.3 Spatial Objects of Practical Interest Treated as Invalid Data

As can be seen in Figure 2.9(d), the spatial union of two pieces of spatial data may yield a spatial object composed of a surface and a line component. For ease of discussion, an object like this is called *hybrid surface*. Such an object may also have practical interest, e.g. a hydrological application may require treating combinations of lakes and rivers pouring into them as integral spatial objects. Contrary to this, hybrid surfaces are not supported in [ES97, FVM97, VE93, GNT91a, Ro93, MJ01, Keig02, Gras02, Lo00, BK01, RHS01]. Indirect alternatives and relevant problems are outlined below.

In [Int02, Mapi02, LPV93, Iso02, Ogis99, Ora00, Vo97], the line and surface components of the hybrid surface in Figure 2.12(a) have to be recorded as separate elements under an attribute whose type is some sort of *set of spatial objects*. Alternatively, distinct tuples of the same data structure have to be used for the recording of the components of hybrid surfaces in [Eg94, DHT94, HT96, Inf01, Ibm01]. Finally, distinct structures have to be considered for the recording of the components of a hybrid surface in [GS95, Esri99, SV92, SH91, Gu88, Post01, VO92, PLL+98, ELNR88]. In all cases, the end-user has

to ascertain that the pieces of spatial data are recorded as is shown in Figure 2.12(b), since a recording like that in either Figure 2.12(c) or Figure 2.12(d) may cause undesirable side effects. Alternatively, the application programmer may have to write code so as to enforce integrity constraints. The existence of database design and manipulation problems, in most of the above alternatives, is obvious.

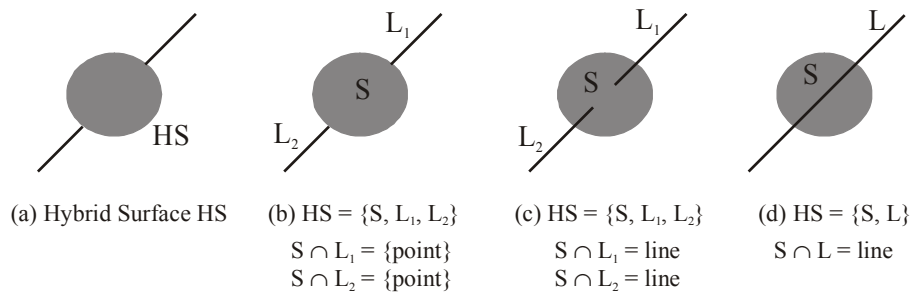


Figure 2.12: Hybrid surface and examples of possible decompositions.

2.5.4 Lack of Spatial Data Validation Mechanisms

Only one spatial data type, *geometry*, is defined in [To91, Bent01, Int95, LPV93, GRS98a, KRSS98, Ro93, SV89, Ora00]. Similarly, data types like *set of points*, *set of lines* or *set of surfaces* are defined in [GS95, Esri00, Int02, Mapi02, SV92, ELNR88]. Subsequently, the recording of valid spatial data, such as *only one line* or *only one surface* in relevant applications, is left to the responsibility of the end-user. Alternatively, the application programmer has to develop data validation code.

2.5.5 Need to Support Non-connected Spatial Objects as Valid Data

The fact, that the result of a spatial operation may yield two or more non-connected spatial objects, has been the reason for defining *sets of spatial objects* data types. Given that such types are complicated, compared to the simple *point*, *line* and *surface* types, it is estimated that the code to support them is also more difficult to develop.

2.5.6 Need to Support Complex Spatial Data Structures

Obtaining more than one spatial object in a spatial operation has also led to the use of complex data structures. Hence, nested data structures are used in [CZ96, Ro93, SV89, Post01, VO92, PLL+98]. Clearly, they are much more

powerful than non-nested but, at the same time, they are also much more difficult both to use and also to develop, as is witnessed by the fact that many alternative formalisms and query languages have been proposed [JS82, PA86, SS86, RKS88, LR94, LD98, GJ00]. In addition, the modelling of spatial data in such structures has considered only the individualities of spatial data. Consequently, relevant approaches do not take full advantage of the whole of the functionality of a general-purpose nested model.

Alternatively, two distinct data structures, one for relations and another for layers, are adopted in [HT96, Bent01, Int95]. Hence, the user has to consider two distinct sets of tools, one for each of these structures.

Finally, two relation structures are adopted in [BJS98]: One is the ordinary non-nested. The other consists of two types of attributes, *implicit* to record spatial data, and the ordinary attributes, termed *explicit*, to record conventional data. The two types of attributes differ in behaviour. This is also estimated to introduce a degree of complexity.

2.5.7 Limitations on Spatial Data Structures

As is well known, the relational model does not impose any limitation on the data types of the attributes of a relation. Contrary to this, some prototype systems and commercial products, which follow GIS-centric approaches, violate this principle. Hence, a relation (actually a layer implemented in terms of a relation) does not allow more than one attribute of a spatial type [DHT94, HT96, Bent01, Int95].

Alternatively, more than one attribute is allowed in relations (actually *feature classes*) in [Esri00, Int02, Mapi02] but only one of them, termed *the main geometry*, can be involved in spatial operations. This is also a limitation, in that it disallows operations of practical interest which may involve two spatial attributes. Note that although more than one spatial attribute are supported in [SV89], the authors argue that this property is of poor interest. This is also estimated not to be precise for the above stated argument.

Finally, both *explicit* and *implicit* attributes are used in [BJS98]. This is also estimated to be a kind of limitation.

2.5.8 Data Loss in Spatial Operations

The precise spatial intersection in Figure 2.11(j) yields a set of surfaces, hybrid surfaces, points and lines. However, the points, lines and the line components of hybrid surfaces are discarded in [DHT94, HT96, Esri00, Bent01, SV92, GNT91a, Gu88, SH91, Inf01, Ibm01, Post01, VO92]. This

also applies to other operations. Clearly, this results in loss of data, which may be useful in certain applications. Contrary to this, a second operation, to obtain only the lines, is applied in [GS95, Vo97].

2.5.9 Limited Functionality of Spatial Operations

By this, it is meant that certain operations are allowed only between spatial objects of a given type.

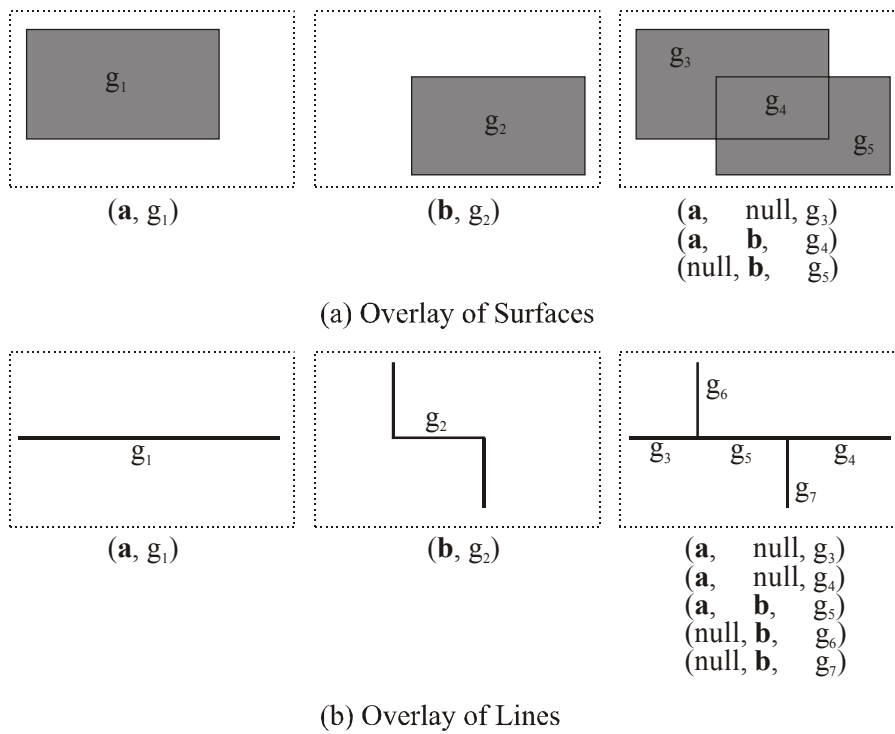


Figure 2.13: Overlay of spatial objects.

As an example, Figures 2.13(a) and Figures 2.13(b) show the result of a (*full*) *overlay* operation, respectively, when applied to two surfaces and to two lines. The tuples of the input relations, as well as those of the result relation, are also shown in the figures. However, only the first of these operations is supported in [GS95, ES97, Esri00, Bent01, Gu88]. An *overlay*, in which spatial objects of different types are involved, is not supported either. This delimits the functionality of spatial operations. Similar observations apply to other spatial operations defined in [GS95, DHT94, HT96, SH91, Inf01, Ibm01].

2.5.10 Definition of Too Many Primitive Operations

A basic set of few spatial operations is common to all approaches. Yet, there are differences in the functionality of even these operations. In addition, the following can be noted:

Certain research approaches define many primitive operations and they are open-ended, in that they provide tools for the user to define new [GS95, LPV93, Eg94, RFS88, SH91, Gu89, CZ96, VO92, PLL+98, Vo97]. The same also applies to the commercial products [Esri00, Int02, Mapi02, Bent01, Ora00, Inf01, Ibm01, Post01].

ARC/INFO [Esri00], supports excessively many operations, more than 100 geo processing tools. Similarly, SQL standards [Iso02, Ogis99] and commercial DBMS [Inf01, Ibm01] support more than 50 spatial functions and predicates. Although few of them are widely used, the set of operations tends to increase from one new release to another.

Finally, certain operations are supported only in few approaches. Examples are *voronoi* [SH91, Gu88, CZ96], *simple path* [CZ96], *split* [CZ96, SH91], and *spatial projection* [CZ96].

From the above, it is obvious that the spatial data management community has not yet established a commonly accepted set of spatial operations, as is for example the case in the relational model. This is also witnessed by the fact that almost all the operations defined in each approach are primitive. Contrary to this, it is argued that a data model should consist of a well-chosen set of general-purpose *fundamental* operations, few of which should be primitive, that will enable users to face the majority of their requirements.

2.5.11 Dimension Dependent Spatial Data Types and Operations

In their majority, various approaches restrict to the management of only 2-d data [GS95, To91, DHT94, ES97, HT96, Esri00, Int02, Mapi02, Bent01, Int95, MJ01, Keig02, Gras02, Lo00, BK01, RHS01, LPV93, RFS88, SV92, GNT91a, SH91, Gu88, CZ96, Ro93, SV89, Iso02, Ogis99, Ora00, Inf01, Ibm01, Post01, PLL+98, CF93, ELNR88, CCF+96]. Some others handle only 2-d and 3-d data [YC94a, Eg94, VO92, Vo97, BBC97, MO86, SDS00]. It is argued, however, that the definition of a model should be more general, independent of the dimension of the spatial data to which it is applied.

2.5.12 Availability of Implementation

A data model is really useful if it can also be implemented. This is really the case in most of the proposed approaches.

2.6 Classification of Spatio-temporal Approaches

Spatio-temporal models are now classified with respect to general characteristics.

2.6.1 Discrete versus Continuous Change with Respect to Time

Spatio-temporal approaches can be classified with respect to the type of changes they intend to model.

Applications like cartography, cadastral systems and topography are mainly concerned with spatial changes that occur at discrete time points. In principle, such applications assume that, after a change, spatial data remains steady until a new change occurs. Relevant spatio-temporal approaches are [Wo94, OP01, TH98, KK94, BJS98, CZ00, Inf00, CG94, BVH96, GFP+01a].

Other applications, such as navigational systems and, more generally, real time applications consider that spatial data changes continuously with respect to time. Spatial objects whose position changes continuously with respect to time are widely known as *moving objects* [Pf00]. Relevant approaches are [GBE+00, MRA00, ES99, GRS00, YC95, SWCD97].

2.6.2 Valid Time versus Transaction Time Modelling

Consider the relation in Figure 2.14(a), which is used to record employee salaries and the time during which each salary was in effect. Hence, the first tuple shows that John's salary became 10k on date d11 and it remained so until date d30. The next tuple shows that on date d31 his salary changed to 20k and it remained so until date d40. Hence, in this relation the data of each tuple is associated with the time during which it was true in the real world. In temporal databases, this kind of time is termed *valid time* and the data associated with this time is called *valid time data*. Such data may be associated with past, present or future time. Valid time is typically managed by the user and reflects his perception of reality. Spatio-temporal approaches that consider only this type of time are [GBE+00, MRA00, ES99, OP01, CZ00, GRS00, Inf00, CG94, SWCD97].

EMPLOYEE			PERSON	
Name	Salary	Time	Name	Birth_date
John	10k	[d11, d30]	John	d1
John	20k	[d31, d40]	Helen	d2
Helen	50k	[d11, d20]		

(a) Time Periods

(b) Time Instants

Figure 2.14: Examples of user-friendly time data types.

Now consider a relation with scheme $R(\text{Name}, \text{Salary})$ and assume that at time d_{11} a user inserts the tuple $(\text{John}, 10k)$, at time d_{31} he updates this tuple to $(\text{John}, 20k)$ and at time d_{40} he deletes it. Now consider another relation with scheme $\text{EMPLOYEE}(\text{Name}, \text{Salary}, \text{Time})$ (Figure 2.14(a)) and assume that it consists of the tuples of R associated with the time during which these tuples remained recorded in R . This type of time, which is associated with the data, is called *transaction time*. The relevant data is called *transaction time data*. Such data may be associated only with the past and the present time. Transaction time is typically maintained by the system.

Finally, it is possible both data to be associated in a relation with both valid and transaction time. Such data is called *bitemporal*. Spatio-temporal approaches that consider both of these types are [Wo94, TH98, KK94, BJS98, BVH96, GFP+01a].

2.6.3 Tuple versus Attribute Time Recording

In some temporal approaches time is recorded at the level of a tuple (Figure 2.14(a)). In general, therefore, each insertion or update results in the addition of a tuple to the relation. Spatio-temporal approaches of this type are [ES99, OP01, TH98, KK94, BJS98, CZ00].

In other approaches, time is recorded at the attribute level. Hence, in each attribute both a value and the relevant time is also recorded. Clearly, such approaches require either complex data types or nested data structures. Spatio-temporal approaches usually consider complex data types. Spatio-temporal approaches of this type are [GBE+00, MRA00, Wo94, GRS00, Inf00, YC95, CG94, SWCD97].

Finally, approaches that adopt both levels of time recording are [BVH96, GFP+01a].

2.7 Remarks on Spatio-temporal Approaches

A number of comments are now made on the various spatio-temporal approaches. As in the case of spatial models, effort has again been made for the remarks to be objective.

2.7.1 Informal Data Types and Operations for Time

As in the case of spatial data management, formalism on data types and operations is missing in many spatio-temporal approaches [MRA00, Wo94, TH98, KK94, BJS98, YC95, CG94, Vo97, SWCD97].

2.7.2 Time Data Types Deviating from Human Perception

As in the case of spatial data management, it is desirable to define time data types close to human perception. Contrary to this, the following are noticed.

1 No Support of User-friendly Data Types for Time

In daily practice people use to consider *time instants*, for events that occur at a given time, and *time intervals (periods of time)* for events that have duration (Figure 2.14). Contrary to this, some approaches support only time instants [ES99, CZ00]. Others support only time periods and they model a time instant t as a time interval $[t, t]$ [OP01, Wo94, GRS00, Inf00, TH98, KK94, BJS98, YC95, CG94, SWCD97]. Clearly, recording this way the date of a birth deviates from human perception. Finally, some others consider a time type composed of the union of non-connected periods [GBE+00, Wo94, GRS00, Inf00]. Such types are complex and not commonly used.

2 Empty set not a Valid Time Period

Approaches that define a time type of the form *union of non-connected periods*, consider the empty set as a valid value [GBE+00, Wo94, CG94, GFP+01a]. This is due to their effort, to define closed operations between elements of this type. Clearly, this also deviates from human perception.

2.7.3 No Support of Various Granularities of Time

The requirement of supporting various granularities of time is widely accepted in the field of temporal databases [LM95]. Yet, the majority of spatio-

temporal approaches provide only one time granularity [GBE+00, Wo94, ES99, OP01, BJS98, YC95, Inf00, CG94, BVH96, GFP+01a, SWCD97].

2.7.4 Need to Support Spatio-temporal Data Types

Some approaches do not consider distinct spatial and time data types but a composite *spatio-temporal* type [GBE+00, MRA00, Wo94, CG94, GRS00]. Operations between elements of this type are also defined. It is estimated that manipulating such elements is far more complicated than manipulating spatial objects and time separately. Yet, on the other hand, it has to be appreciated that some of these approaches aim at manipulating *moving objects* and, more generally, they are closer to real time applications.

2.7.5 Need to Support Complex Data Structures

Certain approaches consider complex data structures. Such a structure, called *History*, is adopted in [GFP+01a] to record sets of the form $\{(p_i, v_i)\}$, where p_i are time periods and v_i are values. Alternatively, two types of relations are considered in some approaches [TH98, BJS98]. The first type consists of the ordinary relations whereas the second type adopts *implicit* attributes, to record time, in addition to the ordinary attributes, in which conventional data is recorded. As a side effect, the semantics of such structures become obscure, as is also reported in [CZ00]. Finally, a complex object model is adopted in [YC95], yet it aims at the modelling of moving objects.

2.7.6 Non-Generic Support of Temporal Data

A data model must be application independent. Contrary to this, some approaches provide spatio-temporal support but they fail to support temporal data when the spatial data is projected out [Wo94, ES99, OP01, Inf00]. This fact has also been reported in [GFP+01a].

2.7.7 Limitations on Data Structures for Time

As in the case of spatial data management, some approaches, which support time at the level of tuple, disallow the incorporation of more than one attribute of a time data type [TH98, KK94, CZ00]. This delimits the power of the underlying data model. Alternatively, other approaches record time in *implicit* attributes [TH98, BJS98], similarly as in [Sn95]. This is also a limitation, as has already been reported in Subsection 2.5.7.

2.7.8 Redefinition of Functionality of Conventional Operations

To support time, certain models have redefined the functionality of known relational operations [YC95, CG94]. For example, if relations $R_1(A, \text{Time})$ and $R_2(B, \text{Time})$ consist, respectively, of the tuple $(a, [d11, d30])$ and $(b, [d21, d40])$ then the join operation has been redefined so as to function as is commonly called, *intersection join*, yielding the tuple $(a, b, [d21, d30])$. This may really be useful for certain types of queries. However, there are also queries of practical interest in which the user would like to obtain the tuple $(a, [d11, d30], b, [d21, d40])$. Such redefinition applies to all the conventional relational algebra operations. However, it delimits the functionality of the model, as has also been identified in [LM95].

2.7.9 No Support of Evolution of Spatial Operations with Respect to Time

Consider a relation $\text{LAND_PARCEL}(\text{Name}, \text{Shape}, \text{Time})$ with one tuple $(\text{John}, g_1, [d11, d30])$, which shows that John was the owner of land g_1 during the period $[d11, d30]$. Let also another relation $\text{LAND_USE}(\text{Use}, \text{Shape}, \text{Time})$ contain the tuple $(\text{Industrial}, g_2, [d21, d40])$, which shows that land g_2 was industrial during the period $[d21, d40]$. Now consider the query

“give the pieces of land owned by John that had an industrial use and the respective periods, as well”.

The result should consist of one or more tuples of the form $(g_{3i}, [d21, d30])$, where each g_{3i} should be one of the pieces of land of the *spatial intersection* of g_1 with g_2 . Although such a result has practical interest, queries of this form are not supported in [MRA00, TH98, KK94, YC95, Inf00, BVH96, SWCD97]. An operation that enables replying the above query is termed to yield the *evolution of spatial intersection with respect to time* (Section 5.7).

2.7.10 Definition of Too Many Primitive Operations

As in the case of spatial data management, a spatio-temporal model should provide a limited set of general-purpose primitive operations. Contrary to this many primitive spatio-temporal operations or functions are defined in [GBE+00, MRA00, Wo94, OP01, Inf00, BVH96, GFP+01a].

2.7.11 Availability of Implementation

As already reported, a data model is really useful if it can also be implemented. It is estimated that the few implementations of spatio-temporal models is due to the fact that this research area is fairly new.

2.8 Thesis Objectives

The major objective of this thesis is the definition of a spatial and of a spatio-temporal model, which overcome the limitations of the relevant approaches that were identified in the previous sections. Detailed objectives are analysed below.

A. Formalization of a Spatial Model

Model Characteristics

- C1. It considers discrete change in space.
- C2. It is closer to raster-based approaches.
- C3. It is database-centric.

Model Properties

- P1. Formalism for spatial types and operations is provided.
- P2. Spatial data types match human perception and, in particular:
 - P2.1 They consist only of *point*, *line* and *surface* types.
 - P2.2 The spatial types are *compatible*.
 - P2.3 The spatial objects are set-theoretically *closed*.
 - P2.4 The empty set is not a valid spatial object.
- P3. A *hybrid* surface is supported as a valid spatial object.
- P4. Spatial data validation mechanisms are enforced by spatial data types.
- P5. All valid spatial objects are connected.
- P6. Only the simple structures of non-nested relations are supported.
- P7. No limitations are enforced by the relation scheme.
- P8. No data loss is encountered in operations.
- P9. Full functionality of all the operations is achieved.
- P10. Few kernel operations achieve full functionality.
- P11. It applies to 2-d spatial data but its extension to n-d is straightforward.
- P12. It is argued that it can be implemented.

B. Formalization of a Spatio-temporal Model

Model Characteristics

- C1. It considers discrete change in time.
- C2. It considers valid time.
- C3. Time is recorded at the level of tuple.

Model Properties

- P1. Formalism for time types and operations is provided.
- P2. Time data types match human perception and, in particular:
 - P2.1 They consist of only generic *instant* and *period* types.
 - P2.2 The empty set is not a valid time period.
- P3. Various granularities of time are supported.
- P4. Spatio-temporal data types are not required.
- P5. Only the simple structures of non-nested relations are supported.
- P6. Generic support of temporal data is provided.
- P7. No limitations are enforced by the relation scheme.
- P8. Conventional operations need not have to be redefined.
- P9. It supports the evolution of spatial operations with respect to time.
- P10. Few kernel operations achieve full functionality.
- P11. It is argued that it can be implemented.

2.9 Conclusions

The characteristics of various spatial and spatio-temporal approaches, whether data modelling approaches or not, have been outlined. A number of properties have thus been identified, which a relevant model should satisfy. Hence, the objectives of this thesis were also identified, i.e. the formalization of a spatial and of a spatio-temporal model, which overcome a number of limitations.

CHAPTER 3

QUANTA AND DATA TYPES FOR SPACE

3.1 Introduction

A set of spatial data types is formalized in this chapter, based on a prior definition of *spatial quanta* [LTV99, LVT99, LVT00]. The formalism enables the discrete representation of 2-d spatial objects. It is closer to raster-based approaches but it does not really match them. The approach enables meeting the relevant objectives specified in Section 2.8. The remainder of this chapter is organized as follows. *Spatial quanta* are formalized in Section 3.2. Based on them, spatial data types are next defined in Section 3.3. Although the definition of a *complete* set of predicates and functions is beyond the objectives of this thesis, some of them are defined in Sections 3.4 and 3.5, respectively. Operations between spatial objects are defined in Section 3.6. Finally, conclusions are drawn in the last section.

3.2 Spatial Quanta

In the remainder of this chapter I (R) denotes the set of integer (real) numbers.

Consider some $n \in I$, $n > 0$, with a fixed value. Let $I_n = \{0, 1, \dots, n-1\}$ and let $i, j \in I_n$, i.e. $0 \leq i \leq n-1$, $0 \leq j \leq n-1$. Then there is exactly one integer k , $0 \leq k \leq n^2-1$, such that $k = n*j + i$. Inversely, it is known that for each such k there is exactly one such pair (i, j) satisfying $k = n*j + i$. As is shown in Figure 3.1, each pair (i, j) , equivalently each k , can then be represented in $R \times R$ by a dot and i (j) is the *horizontal* (*vertical*) *coordinate* of (i, j) or of k . Figure 3.2 depicts an example for $n = 15$. It is said that k is the *ordinal number* of (i, j) . If $k \text{ div } n$ ($k \text{ mod } n$) is the function that returns the quotient (remainder) of the integer division of integer k by integer n , the following functions are defined:

$$\begin{aligned}
 h_coord: I \rightarrow I: h_coord(k) &= k \bmod n \Leftrightarrow 0 \leq k \leq n^2 - 1. \\
 v_coord: I \rightarrow I: v_coord(k) &= k \div n \Leftrightarrow 0 \leq k \leq n^2 - 1. \\
 ord: I^2 \rightarrow I: ord(i, j) &= n*j + i \Leftrightarrow 0 \leq i, j \leq n - 1.
 \end{aligned}$$

Clearly, the inverse function of *ord* is (*h_coord*, *v_coord*).

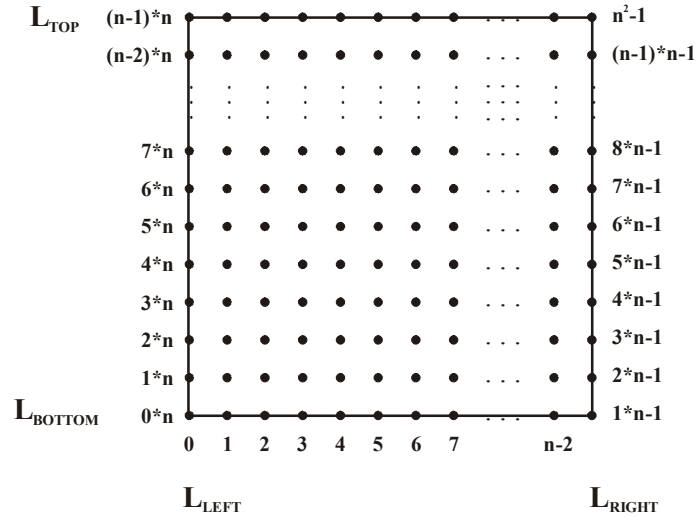


Figure 3.1: Assignment of ordinal numbers to pairs of integers (i, j).

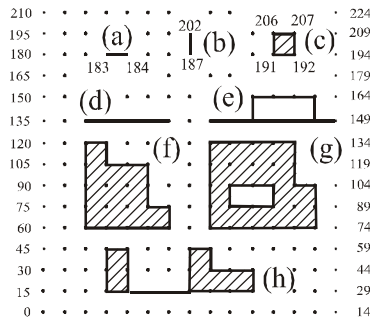


Figure 3.2: Spatial quanta and spatial objects.

It is said that

- (i+1, j) is to the east of (i, j) and that
- (i, j+1) is to the north of (i, j).

It is noticed that if k is the ordinal number of (i, j) then

- k+1 is the ordinal number of (i+1, j) and
- k+n is the ordinal number of (i, j+1).

Using this, if p denotes a pair (i, j) with ordinal number k then

- p_E denotes the pair $(i+1, j)$ whose ordinal number is $k+1$,
- p_{NE} denotes the pair $(i+1, j+1)$ whose ordinal number is $k+n+1$,
- p_N denotes the pair $(i, j+1)$ whose ordinal number is $k+n$.

The pairs p , p_E , p_{NE} , and p_N are called *corner pairs*.

As an example, Figure 3.2 shows that if $ord(p) = 16$ then $ord(p_E) = 17$, $ord(p_{NE}) = 32$, $ord(p_N) = 31$ and p , p_E , p_{NE} , p_N are corner pairs.

Three types of *spatial quanta* are now defined.

Definition 3.1: The singleton $P_k = \{k \mid k \in I_{n^2}\}$ is called a *2-dimensional (2-d) spatial point* or a *2-d quantum point* or simply *point*.

The set of all quantum points is denoted as Q_{POINT} . The following are also defined:

- The geometric representation of P_k is that of k .
- The *coordinates (horizontal, vertical)* of P_k are those of k , i.e.
 $h_coord(P_k) \equiv h_coord(k)$ and $v_coord(P_k) \equiv v_coord(k)$.
- The ordinal number of P_k is that of its coordinates, i.e.
 $ord(P_k) \equiv ord(h_coord(P_k), v_coord(P_k))$.

The *east, north* and *northeast point* of P_k are denoted, respectively, P_{Ek} , P_{Nk} and P_{NEk} , i.e. $P_{Ek} \equiv P_{k+1}$, $P_{Nk} \equiv P_{k+n}$ and $P_{NEk} \equiv P_{k+n+1}$. P , P_{Ek} , P_{Nk} , and P_{NEk} are *corner points*. On a system of orthogonal coordinates a spatial point is conventionally denoted only by its ordinal number (Figure 3.1).

Definition 3.2: Let $P_k, P_{Ek} \in Q_{POINT}$, i.e. the coordinates of one of them are (i, j) and those of the other are $(i+1, j)$. Then the set

$$H_k \equiv \{(x, y) \in \mathbb{R}^2 \mid i \leq x \leq i+1 \wedge y = j\}$$

is a *pure horizontal quantum line*.

Similarly, if P_k and $P_{Nk} \in Q_{POINT}$, i.e. the coordinates of one of them are (i, j) and those of the other are $(i, j+1)$, then the set

$$V_k \equiv \{(x, y) \in \mathbb{R}^2 \mid x = i \wedge j \leq y \leq j+1\}$$

is called a *pure vertical quantum line*.

Finally, *pure quantum line* is called any pure horizontal or any pure vertical quantum line.

Note in the above definitions that the index k of a pure horizontal (vertical) line H_k (V_k) matches that of point P_k . The set of all pure horizontal (pure vertical) quantum lines is denoted Q_{PH} (Q_{PV}). The set of all pure quantum lines is denoted Q_{PL} .

By definition, a pure quantum line consists of an infinite number of \mathbb{R}^2 elements.

The points P_k and P_{Ek} (P_k and P_{Nk}) of a horizontal (vertical) pure quantum line are its *end points*. Alternatively, a pure quantum line is denoted as $ql_{p,q}$, where p and q are the ordinal numbers of its end points. By definition, $ql_{p,q} = ql_{q,p}$, i.e. no direction is considered. A pure quantum line $ql_{p,q}$ can geometrically be interpreted as a line segment. Hence, (a) and (b) in Figure 3.2 are two pure quantum lines, a pure horizontal quantum line, $ql_{183,184}$, and a pure vertical quantum line, $ql_{187,202}$.

Definition 3.3: Let P_k and P_{Ek} , P_{NEk} and P_{Nk} be four corner points. Then the set

$$S_k \equiv \{(x, y) \in \mathbb{R}^2 \mid i \leq x \leq i+1 \wedge j \leq y \leq j+1\}$$

is called a *pure quantum surface*.

The set of all pure quantum surfaces is denoted as Q_{PS} .

Note in this definition that the index k of a pure surface S_k matches that of point P_k . By definition, a pure quantum surface consists of an infinite number of \mathbb{R}^2 elements.

Alternatively, a pure quantum surface is denoted as $qs_{p,q,r,s}$, where p, q, r and s is a clockwise or a counter-clockwise order of the ordinal numbers of the points P_k, P_{Ek}, P_{NEk} and P_{Nk} . Hence, $qs_{p,q,r,s}, qs_{q,r,s,p}, qs_{s,r,q,p}$ are equivalent notations of the same pure quantum surface, i.e. no direction is considered again. A pure quantum surface can geometrically be interpreted as a square. Hence, Figure 3.2(c) depicts a pure quantum surface, $qs_{191,192,207,206}$. The points P_k, P_{Ek}, P_{NEk} and P_{Nk} are called the *corners* of S_k . The quantum lines H_k, H_{Nk}, V_k and V_{Ek} are called the *sides* of S_k .

Definition 3.4: $Q_{LINE} \equiv Q_{PL} \cup Q_{POINT}$ is called the *set of all quantum lines*.

Definition 3.5: $Q_{SURFACE} \equiv Q_{PS} \cup Q_{LINE}$ is called the *set of all quantum surfaces*.

By definition, $Q_{SURFACE}$ consists of all the pure quantum surfaces, of all the pure quantum lines and of all the pure quantum points. Due to this, an element in $Q_{SURFACE}$ is called *quantum of space* or *spatial quantum* and $Q_{SURFACE}$ is alternatively denoted as Q_{GEO} .

Definition 3.6: If q is a quantum other than point, it is defined that

$$ord(q) \equiv ord(P_k).$$

Hence, $ord(H_k) = ord(V_k) = ord(S_k) \equiv ord(P_k) = k$.

3.3 Data Types for Space

Based on the concept of spatial quanta, a series of data types for space are now defined. A spatial object consists in general of an infinite number of \mathbb{R}^2 elements, but it is composed of only spatial quanta. Before the formalism is given, preliminary definitions for *quantum set* and *spatial connectivity* are provided.

Definition 3.7: If $\emptyset \neq S = q_1 \cup q_2 \cup \dots \cup q_n \subset \mathbb{R}^2$, where $q_i \in Q_{\text{GEO}} \forall i = 1, 2, \dots, n$, S is called a *quantum set*.

Conventionally, the notation $g = g_1 g_2 \dots g_n$ is used in the sequel in place of $g = g_1 \cup g_2 \cup \dots \cup g_n$.

Definition 3.8: A *quantum set* $S \subset \mathbb{R}^2$ is called *connected* iff for every pair of reals $x, y \in S$ there exists a sequence of spatial quanta $q_1, q_2, \dots, q_n \subseteq S$ that satisfies the following two properties:

- (i) $x \in q_1$ and $y \in q_n$.
- (ii) $q_i \cap q_{i+1} \neq \emptyset$ for $i = 1, 2, \dots, n-1$.

Based on this definition, spatial data types are now formalized.

Definition 3.9: Let g be a non-empty, connected quantum set. It is then defined that g is of a (*2-d spatial*) type

- POINT $\Leftrightarrow g \equiv q_i, q_i \in Q_{\text{POINT}}$.
- PLINE $\Leftrightarrow g \equiv \bigcup_i q_i, q_i \in Q_{\text{PL}}$.
- LINE $\Leftrightarrow g \equiv \bigcup_i q_i, q_i \in Q_{\text{LINE}}$.
- PSURFACE $\Leftrightarrow g \equiv \bigcup_i q_i, q_i \in Q_{\text{PS}}$.
- SURFACE $\Leftrightarrow g \equiv \bigcup_i q_i, q_i \in Q_{\text{SURFACE}}$.

An element of one of the above types is called, respectively, (*2-d spatial*) *point, pure line, line, pure surface* and *surface*.

Figure 3.2 depicts the geometric interpretation of the following spatial objects:

Points: $\{0\}, \{2\}, \dots, \{224\}$.

Pure lines: (a), (b), (d) (e). Object (d) is the union of four pure quantum lines.

Lines: Any of the previous pure lines and points.

Pure surfaces: (c), (f) and (g). Object (f) is the union of eleven pure quantum surfaces. Object (g) is a surface with a *hole*. A relevant predicate is defined later in Section 3.4.

Surfaces: Any of the above pure surfaces, any of the above lines and (h). Object (h) is a *hybrid surface*. A relevant predicate is defined later in Section 3.4.

Some remarks, on Definitions 3.7 - 3.9 are the following.

1. A point is not an element; it is a unary set. Defining it as a set is due to the fact that points are practically involved in set operations, as will also be seen in the next chapter.
2. A spatial object of a PLINE type is also of a SURFACE type. Hence, it is said that a pure line is also a *degenerate pure surface*.
3. A spatial object of a POINT type is also of both a LINE type and of a SURFACE type. Hence, it is said that a point is also a *degenerate pure line* and a *degenerate pure surface*.

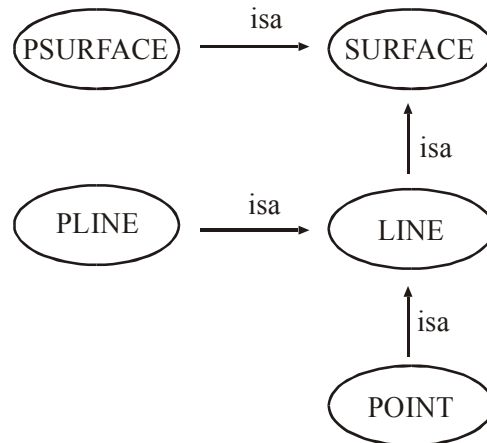


Figure 3.3: Data types for space.

It is noted that in cartography it is meaningless to consider a line from which some point is missing, whether it is an end point or not. Similarly, it is meaningless to consider a surface from which some point or line is missing. The formalism undertaken has taken particular provision for this property to be satisfied. Hence, the following property is satisfied:

4. By definition, all the spatial objects are *closed*.

The term *closed set* is assumed to be known from mathematics. Note that remarks 2, 3 and 4 match human perception. Figure 3.3 illustrates the relationship between all the spatial data types.

Two spatial objects of particular interest are used in subsequent chapters, $g_{\text{SURF_ALL}}$ and $g_{\text{LINE_OUT}}$, defined as follows:

$$g_{\text{SURF_ALL}} = S_0 S_1 S_2 \dots S_{(n-1)*n-2} = \cup_i S_i, \text{ where } S_i \in Q_{\text{PS}}.$$

Hence, $g_{\text{SURF_ALL}}$ is the union of all the quanta (Figure 3.1).

$$g_{\text{LINE_OUT}} = L_{\text{BOTTOM}}L_{\text{RIGHT}}L_{\text{TOP}}L_{\text{LEFT}}$$

where

- $L_{\text{BOTTOM}} = H_0H_1 \dots H_{n-2}$,
- $L_{\text{TOP}} = H_{(n-1)*n}H_{(n-1)*n+1} \dots H_{n^2-2}$,
- $L_{\text{LEFT}} = V_0V_n \dots V_{(n-2)*n}$,
- $L_{\text{RIGHT}} = V_{n-1}V_{2*n-1} \dots V_{(n-1)*n-1}$,

are shown in Figure 3.1.

3.4 Predicates

Providing definitions for a *complete* set of predicates is beyond the objectives of this thesis. Hence, the definitions below restrict only to those that are either necessary for the subsequent formalism or are used in examples. Initially, however, a total ordering on spatial quanta and two canonical representations of spatial objects are defined.

Definition 3.10: If q_1, q_2 are spatial quanta it is defined that

$$q_1 \triangleleft q_2 \Leftrightarrow \neg(q_1 = q_2).$$

Definition 3.11: Let $q_1, q_2 \in Q_{\text{SURFACE}}$. If $\text{ord}(q_1) \triangleleft \text{ord}(q_2)$ then it is defined that

$$q_1 < q_2 \Leftrightarrow \text{ord}(q_1) < \text{ord}(q_2).$$

If $\text{ord}(q_1) = \text{ord}(q_2)$ it is defined that

$$\begin{aligned} q_1 < q_2 \Leftrightarrow & (q_1 \in Q_{\text{POINT}} \wedge q_2 \in Q_{\text{SURFACE}} - Q_{\text{POINT}}) \vee \\ & (q_1 \in Q_{\text{PH}} \wedge q_2 \in Q_{\text{SURFACE}} - Q_{\text{POINT}} \cup Q_{\text{PH}}) \vee \\ & (q_1 \in Q_{\text{PV}} \wedge q_2 \in Q_{\text{PS}}). \end{aligned}$$

By definition, it follows that

$$P_1 < H_1 < V_1 < S_1 < P_{i+1} < H_{i+1} < V_{i+1} < S_{i+1}.$$

Definition 3.12: Let g be a spatial object and let $L_g = \langle q_1, q_2, \dots, q_n \rangle$ be a list of spatial quanta that satisfies the following properties:

- (i) $q_i \in L_g \Leftrightarrow q_i \subseteq g \ \forall i = 1, 2, \dots, n$.
- (ii) $q_i \neq q_j \ \forall i, j = 1, 2, \dots, n, i \neq j$.
- (iii) $q_i < q_{i+1} \ \forall i = 1, 2, \dots, n-1$.

Then L_g is called the *maximal canonical form* of g and is denoted as $L_g = \text{maxcform}(g)$.

Definition 3.13: Let g be a spatial object and let $L_g = \langle q_1, q_2, \dots, q_n \rangle$ be a list of spatial quanta that satisfies the following properties:

- (i) $q_i \in L_g \Leftrightarrow q_i \subseteq g \ \forall i = 1, 2, \dots, n.$
- (ii) $q_i \neq q_j \ \forall i, j = 1, 2, \dots, n, i \neq j.$
- (iii) $q_i < q_{i+1} \ \forall i = 1, 2, \dots, n-1.$
- (iv) $(\forall q_i \in L_g)(\forall q_j \in L_g, i \neq j) (q_i \not\subset q_j).$

Then L_g is called the *minimal canonical form* of g and is denoted as $L_g = \text{mincform}(g)$ or simply $L_g = \text{cform}(g)$.

It should be obvious that for a given g , both of these forms are uniquely defined. Moreover, each of them can be obtained from the other. For example, $\text{cform}(g)$ can be obtained from $\text{maxcform}(g)$ by applying the following steps:

S1: Let $L = \text{maxcform}(g)$.

S2: Eliminate P_k from L if any of $H_k, H_{k-1}, V_k, V_{k-n}, S_k, S_{k-1}, S_{k-n}, S_{k-n-1}$ is present.

S3: Eliminate H_k from L if any of S_k, S_{k-n} is present.

S4: Eliminate V_k from L if any of S_k, S_{k-1} is present.

Definition 3.14: The number of quanta in $\text{cform}(g)$ is called the *string_length* of g and it is denoted as $\text{string_length}(g)$.

If $\text{cform}(g) = \langle q_1 q_2 \dots q_n \rangle$ then $g[i]$ is used to denote the i -th spatial quantum within $\text{cform}(g)$, i.e.

$$g[i] = q_i, i = 1, 2, \dots, n.$$

Based on the above definitions, a number of spatial predicates are now defined.

- If g_1 and g_2 are spatial objects, it is defined that

$$g_1 = g_2 \Leftrightarrow (\forall q \in Q_{\text{SURFACE}})((q \subseteq g_1 \Rightarrow q \subseteq g_2) \wedge (q \subseteq g_2 \Rightarrow q \subseteq g_1)).$$

The validity of the following corollary is obvious.

Corollary 3.1: If g_1, g_2 are spatial objects then

$$g_1 = g_2 \Leftrightarrow \text{cform}(g_1) = \text{cform}(g_2)$$

$$\Leftrightarrow (\text{string_length}(g_1) = \text{string_length}(g_2) = n) \wedge (g_1[i] = g_2[i], i = 1, 2, \dots, n).$$

- $g_1 \diamond g_2 \Leftrightarrow \neg(g_1 = g_2).$

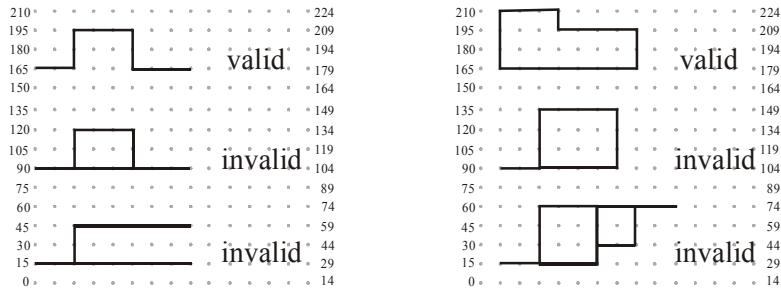
Based on these two predicates, a total ordering on spatial objects can be defined in terms of the total ordering on spatial quanta. It can easily be verified that the definition matches the total ordering on character strings in terms of the total ordering on characters.

- Let g_1, g_2 be two spatial objects, $g_1 \diamond g_2$ and let $\text{string_length}(g_1) = m$, $\text{string_length}(g_2) = n$. It is then defined that

$$\begin{aligned}
g_1 < g_2 &\Leftrightarrow (g_1[1] < g_2[1]) \vee \\
&(\exists k, 1 \leq k < \min(m, n)) \\
&\quad (g_1[i] = g_2[i], i = 1, 2, \dots, k \wedge g_1[k+1] < g_2[k+1]) \vee \\
&\quad (g_1[i] = g_2[i], i = 1, 2, \dots, m) \wedge (m < n).
\end{aligned}$$

As an example, assume that $n = 15$ (Figure 3.2). Then an ordering of some spatial objects (not shown in the figure), is the following:

$$H_0H_1 < H_0H_1H_2 < H_0V_1 < H_0V_1H_{16}H_{17} < S_1H_2H_3.$$



(a) Valid and invalid *simple* lines (b) Valid and invalid *circular* lines

Figure 3.4: Examples of valid and invalid *simple* and *circular* lines.

Based on the above, the following predicates between spatial objects are also defined:

- $g_1 \leq g_2 \Leftrightarrow (g_1 < g_2) \vee (g_1 = g_2).$
- $g_1 > g_2 \Leftrightarrow \neg(g_1 \leq g_2).$
- $g_1 \geq g_2 \Leftrightarrow \neg(g_1 < g_2).$

Two more predicates of practical interest are the following:

- $g_1 \text{ } cp \text{ } g_2 \Leftrightarrow g_1 \cap g_2 \neq \emptyset$ (g_1 and g_2 have common points).
- $g_1 \text{ } disjoint \text{ } g_2 \Leftrightarrow g_1 \cap g_2 = \emptyset.$

The unary predicates below enable determining whether a spatial object is of a particular data type. This functionality is useful in retrievals.

- $is_point(g) \Leftrightarrow g \in \text{POINT}.$
- $is_pure_line(g) \Leftrightarrow g \in \text{PLINE}.$
- $is_line(g) \Leftrightarrow g \in \text{POINT} \vee g \in \text{PLINE}.$
- $is_pure_surface(g) \Leftrightarrow g \in \text{PSURFACE}.$
- $is_surface(g) \Leftrightarrow g \in \text{SURFACE}.$

- $is_hybrid_surface(g) \Leftrightarrow g \in SURFACE \wedge g \notin LINE \cup PSURFACE.$

If $card(A)$ denotes the cardinality of a set A , the following two predicates determine whether a pure line is *simple* or *circular*.

- $is_simple(g) \Leftrightarrow$
 $g \in PLINE \wedge (\forall p \in Q_{POINT}, p \subset g)(card(\{q \mid q \in Q_{PL} \wedge q \subseteq g \wedge p \subset q\}) \leq 2).$
- $is_circular(g) \Leftrightarrow$
 $g \in PLINE \wedge (\forall p \in Q_{POINT}, p \subset g)(card(\{q \mid q \in Q_{PL} \wedge q \subseteq g \wedge p \subset q\}) = 2).$

According to the first definition, a pure line g is simple iff every quantum point of it belongs to at most two quantum lines that are subsets of g . Similarly, a pure line is circular iff every quantum point of it belongs to exactly two quantum lines that are subsets of g . Examples of valid and invalid simple and circular lines are shown in Figure 3.4.

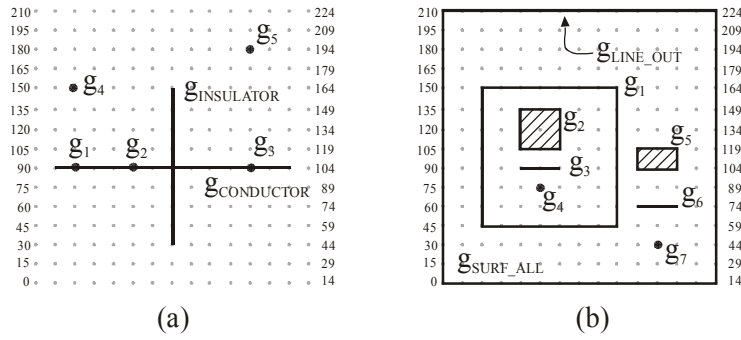


Figure 3.5: Illustration of functionality of predicate *conductive*.

If g_1 , g_2 , $g_{CONDUCTOR}$ and $g_{INSULATOR}$ are spatial objects of any type, the next predicate enables determining whether there is a *path* from g_1 to g_2 in $g_{CONDUCTOR}$, which does not cross $g_{INSULATOR}$.

- $conductive(g_A, g_B, g_{CONDUCTOR}, g_{INSULATOR}) \Leftrightarrow$
there exists a sequence of quanta $q_1, q_2, \dots, q_n \subseteq g_{CONDUCTOR}$ which satisfies the following three conditions:
 - $q_1 \subseteq g_A.$
 - $q_n \subseteq g_B.$
 - $(\forall i, 1 \leq i \leq n-1)(\emptyset \neq q_i \cap q_{i+1} \not\subseteq g_{INSULATOR}).$

Considering therefore the objects in Figure 3.5(a), it follows that

$$conductive(g_1, g_B, g_{CONDUCTOR}, g_{INSULATOR})$$

evaluates to true if $g_B = g_2$. If g_B is any of g_3, g_4, g_5 the predicate evaluates to false. As should be obvious from the definition, the first two objects may be interchanged, i.e.

$$\begin{aligned} & \text{conductive}(g_A, g_B, g_{\text{CONDUCTOR}}, g_{\text{INSULATOR}}) = \\ & \text{conductive}(g_B, g_A, g_{\text{CONDUCTOR}}, g_{\text{INSULATOR}}). \end{aligned}$$

The objects do not have to be distinct. Hence, for $i = 1, 2, 3$,

$$\text{conductive}(g_i, g_i, g_{\text{CONDUCTOR}}, g_{\text{INSULATOR}})$$

evaluates to true but for $i = 4, 5$, it evaluates to false. Similarly,

$$\text{conductive}(g_A, g_{\text{INSULATOR}}, g_{\text{CONDUCTOR}}, g_{\text{INSULATOR}})$$

always evaluates to false. More generally, the objects may have points in common. Hence, if $g_B \subseteq g_{\text{INSULATOR}}$ then

$$\text{conductive}(g_A, g_B, g_{\text{CONDUCTOR}}, g_{\text{INSULATOR}})$$

always evaluates to false. Finally, the objects can be of any data type. For example, if $g_{\text{LINE_OUT}}$ and $g_{\text{SURF_ALL}}$ are the spatial objects defined in Section 3.3 then, for the objects in Figure 3.5(b),

$$\text{conductive}(g_i, g_{\text{LINE_OUT}}, g_{\text{SURFACE_ALL}}, g_1)$$

evaluates to true for $i = 5, 6, 7$, but it evaluates to false for $i = 2, 3, 4$.

The next predicate is defined in terms of *conductive* and is of spatial interest, as is witnessed by its name.

- $g_1 \text{ surrounds } g_2 \Leftrightarrow \neg \text{conductive}(g_2, g_{\text{LINE_OUT}}, g_{\text{SURF_ALL}}, g_1)$.

Hence, for the objects in Figure 3.5(b),

$$g_1 \text{ surrounds } g_i$$

evaluates to true for $i = 2, 3, 4$, but it evaluates to false for $i = 5, 6, 7$. Note that, as opposed to *conductive*,

$$g_1 \text{ surrounds } g_2 \neq g_2 \text{ surrounds } g_1 .$$

Based on the previously defined predicates, another one is the following:

- $\text{has_holes}(g) \Leftrightarrow (\text{is_pure_surface}(g) \vee \text{is_hybrid_surface}(g)) \wedge (\exists qs \in Q_{\text{PS}})((g \text{ surrounds } qs) \wedge (qs \not\subseteq g))$.

3.5 Functions

Providing definitions for a *complete* set of functions is also beyond the objectives of this thesis. Hence, the definitions below restrict only to those that are either necessary for the subsequent formalism or are used in examples.

Function *ord* has been defined for any spatial quantum. Functions *h_coord* and *v_coord* have been defined only for a point P_k and are now extended, so as to apply to any spatial quantum.

- If q is any of H_k or V_k or S_k , it is defined that

$$\begin{aligned} h_coord(q) &= h_coord(P_k), \\ v_coord(q) &= v_coord(P_k). \end{aligned}$$

The next function enables obtaining a spatial point from a pair of integers:

- $form_point(i, j) = P_k \Leftrightarrow k = n*j + 1 \wedge 0 \leq k \leq n^2 - 1$.

Finally, functions can be defined that return not only any of P_k, H_k, V_k, S_k from any of the others but also points, pure horizontal lines, pure vertical lines and surfaces that are to the east, west, north, south etc of these quanta.

Definition 3.15: Given two spatial data types DT_1 and DT_2 , their *least common supertype*, $lcs(DT_1, DT_2)$, is defined as the data type DT that satisfies the following two properties:

- $DT_1 \cup DT_2 \subseteq DT$.
- If $DT_1 \cup DT_2 \subseteq DT'$ and $DT' \neq DT$ then $DT \subset DT'$.

For example,

$$\begin{aligned} lcs(\text{POINT}, \text{PLINE}) &= \text{LINE}. \\ lcs(\text{PLINE}, \text{PSURFACE}) &= \text{SURFACE}. \\ lcs(\text{PLINE}, \text{PLINE}) &= \text{PLINE}. \end{aligned}$$

Given that the least common supertype of the data types of any two objects g_1 and g_2 is always defined uniquely (Figure 3.3), it is said that g_1 and g_2 are *spatially compatible*. This enables involving spatial objects of distinct types in binary set operations such as union, except and intersect. In spite of this, the following type transformation functions are defined, which can be useful in operations (see relevant discussion on operation *Compute*, in Definition 4.18).

- $to_point(g) = g \Leftrightarrow is_point(g)$.
- $to_pure_line(g) = g \Leftrightarrow is_pure_line(g)$.
- $to_line(g) = g \Leftrightarrow is_line(g)$.
- $to_pure_surface(g) = g \Leftrightarrow is_pure_surface(g)$.
- $to_surface(g) = g \Leftrightarrow is_surface(g)$.
- If P_{k1}, P_{k2} are points with coordinates $(i_{k1}, j_{k1}), (i_{k2}, j_{k2})$, respectively, their Euclidian distance is defined as the real number

$$distance(P_{k1}, P_{k2}) = \sqrt{(i_{k2} - i_{k1})^2 + (j_{k2} - j_{k1})^2}.$$

To extend this function so as to apply to any two spatial objects, one definition is first given, that returns all the pure quanta of a given type that are subsets of g .

Definition 3.16: If g is a spatial object then

$$\begin{aligned} qpoints(g) &\equiv \{q \mid q \subseteq g \wedge q \in Q_{\text{POINT}}\} \\ qplines(g) &\equiv \{q \mid q \subseteq g \wedge q \in Q_{\text{PL}}\} \\ qpsurfaces(g) &\equiv \{q \mid q \subseteq g \wedge q \in Q_{\text{PS}}\} \end{aligned}$$

- Now let g_A and g_B be two spatial objects. It is then defined that
 $distance(g_A, g_B) = \min(distance(P_{A_i}, P_{B_i})),$
 where $P_{A_i} \in qpoints(g_A)$ and $P_{B_i} \in qpoints(g_B)$.

Similarly, the next function enables retrieving the greatest distance between one point in g_A and another in g_B .

- $greatest_distance(g_A, g_B) = \max(distance(P_{A_i}, P_{B_i})).$

In a similar manner, it is possible to obtain the coordinates, and subsequently define the *minimum bounding rectangle* of an object g .

If $card(A)$ denotes the cardinality of a set A the following functions are defined:

- $length(g) = card(\{q \mid q \subseteq g \wedge$
 $q \in Q_{PLINE} \wedge$
 $(\forall q' \subseteq g)(q' \in Q_{SURFACE})(q \not\subseteq q')\}).$
- $area(g) = card(qpsurfaces(g)).$

The first function returns the number of pure line quanta that are subsets of g but, at the same time, they are not subsets of pure surfaces of g . The second one returns the number of pure surfaces that are subsets of g . By definition, these functions can be applied to a spatial object of *any* data type. However, if g is either a point or a pure surface then $length(g) = 0$. Similarly, if g is either a point or a pure line then $area(g) = 0$.

3.6 Operations

In this section, operations between sets of spatial objects are formalized. From the way they are defined, it is guaranteed that the result set consists of closed objects. Moreover, the results obtained match those of the relevant mathematical definitions. Use of these operations is made in the definition of Relational Algebra operations.

Definition 3.17: The *spatial union* of two sets of spatial objects $S_A = \{g_{A_i} \mid i=1, 2, \dots, r\}$ and $S_B = \{g_{B_j} \mid j=1, 2, \dots, s\}$ is a set $S = \{g_{U_k} \mid k=1, 2, \dots, n\}$ of *disjoint* spatial objects, defined as follows: For a quantum $q \in Q_{SURFACE}$ and a given $k \in I, 1 \leq k \leq n$,

$$q \subseteq g_{U_k} \Leftrightarrow (\exists i, 1 \leq i \leq r)(q \subseteq g_{A_i}) \vee (\exists j, 1 \leq j \leq s)(q \subseteq g_{B_j}).$$

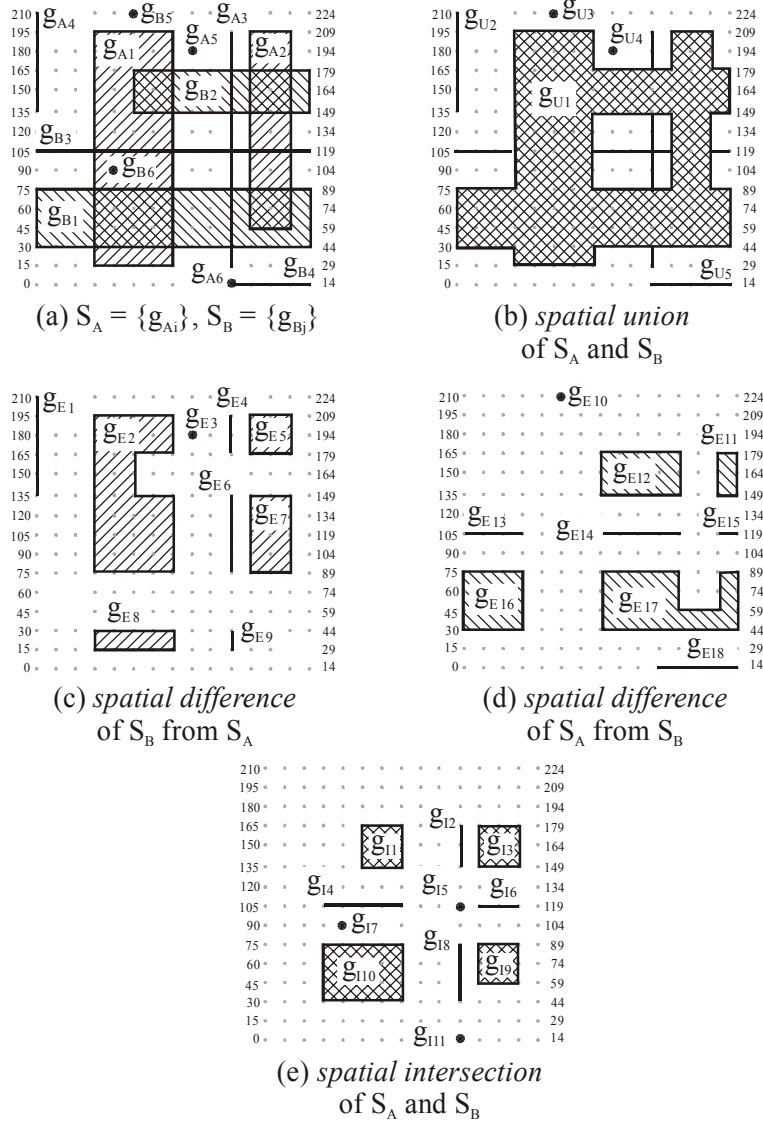


Figure 3.6: Illustration of *spatial union*, *difference* and *intersection*.

For an example, consider any pair of the spatial objects g_A and g_B in Figure 2.9. Then the respective spatial union of $\{g_A\}$ and $\{g_B\}$ is also shown in the same figure. For another example, let S_A and S_B be the sets of spatial objects in Figure 3.6(a). Then their spatial union consists of the objects whose geometric representation is shown in Figure 3.6(b).

Definition 3.18: The *spatial difference* of a set of spatial objects $S_B = \{g_{B_j} \mid j=1, 2, \dots, s\}$ from another set $S_A = \{g_{A_i} \mid i=1, 2, \dots, r\}$ is a set $S = \{g_{D_k} \mid k=1, 2, \dots, n\}$ of *disjoint* spatial objects, defined as follows:

For a quantum $q \in Q_{\text{SURFACE}}$ and a given $k \in I$, $1 \leq k \leq n$,

$$q \subseteq g_{D_k} \Leftrightarrow (\exists q' \in Q_{\text{SURFACE}}) ((q \subseteq q') \wedge (q' \subseteq g_{A_i}) \wedge (q' \not\subseteq g_{B_j})).$$

Note that, due to the definition, if $S_A = \{g_A\}$, $S_B = \{g_B\}$, g_A is a pure surface and g_B is either a pure line or a point then the spatial difference of S_B from S_A always matches S_A (i.e. it does not matter whether g_A and g_B have points in common). The same is also true if g_A is a pure line and g_B is a point. Finally, the same is also true if g_A is, for example, a pure horizontal line and g_B is a vertical line. Contrary to this, if g_A is a point and g_B is either a pure surface or a pure line and $g_A \subset g_B$ then $S = \emptyset$. Similarly, if g_A is a pure line, g_B is a pure surface and $g_A \subset g_B$ then, again, $S = \emptyset$. In either case, the result is either the empty set or it consists of closed spatial objects. This functionality is demonstrated by the examples below.

Consider any pair of the spatial objects g_A and g_B in Figure 2.10. Then the respective spatial difference of $\{g_B\}$ from $\{g_A\}$ is also shown in the same figure. For another example, let S_A and S_B be the sets of spatial objects in Figure 3.6(a). Then the spatial difference of S_B from S_A consists of the objects whose geometric representation is shown in Figure 3.6(c) and the spatial difference of S_A from S_B consists of the objects whose geometric representation is shown in Figure 3.6(d).

Definition 3.19: The *spatial intersection* of two sets of spatial objects $S_A = \{g_{A_i} \mid i=1, 2, \dots, r\}$ and $S_B = \{g_{B_j} \mid j=1, 2, \dots, s\}$ is a set $S = \{g_{I_k} \mid k=1, 2, \dots, n\}$ of *disjoint* spatial objects, defined as follows: For a quantum $q \in Q_{\text{SURFACE}}$ and a given $k \in I$, $1 \leq k \leq n$,

$$q \subseteq g_{I_k} \Leftrightarrow (\exists i, 1 \leq i \leq r)(q \subseteq g_{A_i}) \wedge (\exists j, 1 \leq j \leq s)(q \subseteq g_{B_j}).$$

For an example, consider any pair of the spatial objects g_A and g_B in Figure 2.11. Then the respective spatial intersection of $\{g_A\}$ and $\{g_B\}$ is also shown in the same figure. For another example, let S_A and S_B be the sets of spatial objects in Figure 3.6(a). Then their spatial intersection consists of the objects whose geometric representation is shown in Figure 3.6(e).

Definition 3.20: The *spatial complementation* of a set of spatial objects $S_B = \{g_{B_j} \mid j=1, 2, \dots, s\}$ is a set $S = \{g_{C_k} \mid k=1, 2, \dots, n\}$ of *disjoint* spatial objects, which matches the spatial difference of S_B from $S_A = \{g_{\text{SURF_ALL}}\}$.

As an example, if S_B is the set of spatial objects whose geometric representation is shown in Figure 3.7(a), then its spatial complementation

consists of the spatial objects whose geometric representation is shown in Figure 3.7(b). It is worth noticing that the spatial complementation of g_{SURF_ALL} is the empty set. This result matches fully the complementation of the base set in mathematics. Notice also that if S_B consists of spatial objects of either a POINT or PLINE type then its spatial complementation is $S = \{g_{SURF_ALL}\}$. Note finally that the result of this operation is either the empty set or it consists of closed spatial objects.

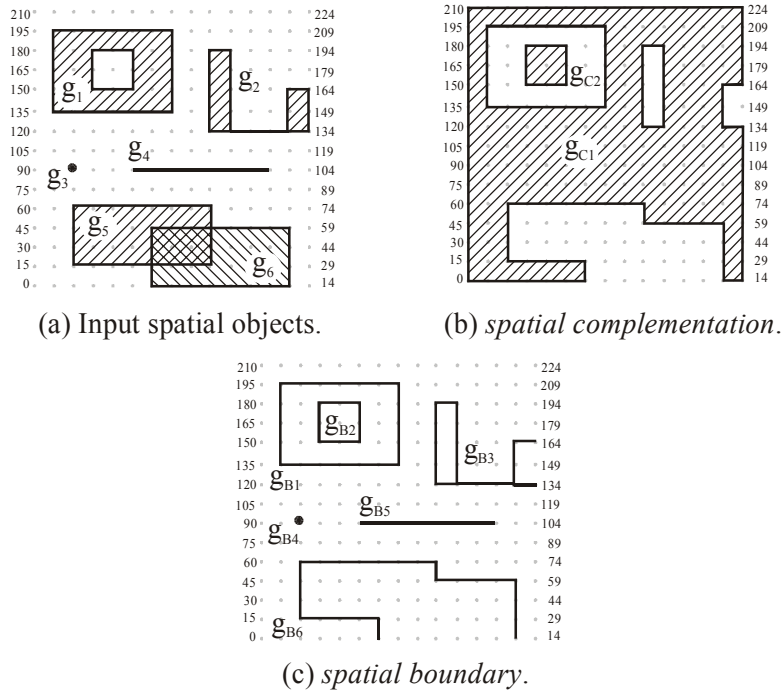


Figure 3.7: Illustration of *spatial complementation* and *boundary*.

Definition 3.21: If $S_A = \{g_{A_i} \mid i=1, 2, \dots, r\}$ and $S_B = \{g_{B_i} \mid i=1, 2, \dots, s\}$ is its spatial complementation then the *spatial boundary* of S_A (and also of S_B) is a set of *disjoint* spatial objects defined as the spatial intersection of S_A with S_B .

As an example, if S_A is the set of spatial objects whose geometric representation is shown in Figure 3.7(a), then its spatial boundary consists of the objects whose geometric representation is shown in Figure 3.7(c). The following observations are worth noticing:

- (i) If S_A consists of only points and pure lines, its spatial boundary is again S_A . This matches fully the mathematical definition of the boundary of a set of points and lines.

- (ii) If g_2 is the spatial object in Figure 3.7(a) then the spatial boundary of $\{g_2\}$ is the set $\{g_{B3}\}$, whose geometric representation is shown in Figure 3.7(c). As can be seen, g_{B3} does not contain any of the pure quantum lines V_{134}, V_{149} .

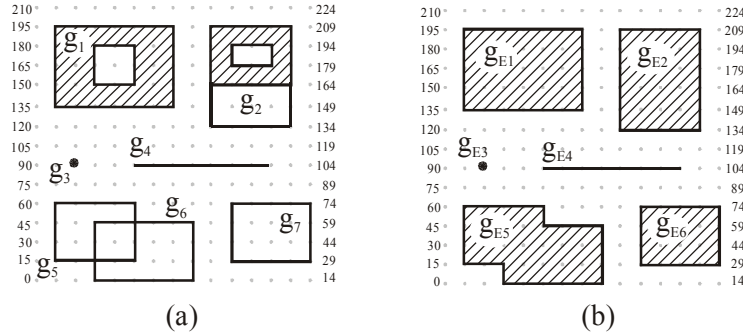


Figure 3.8: Illustration of *spatial envelope*.

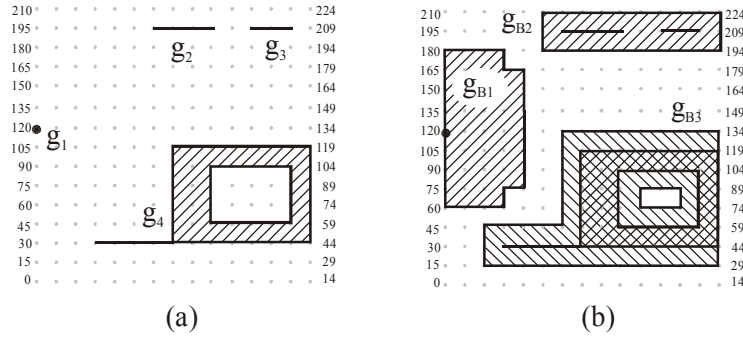


Figure 3.9: Illustration of *spatial buffer*.

Definition 3.22: The *spatial envelope* of a set of spatial objects $S_A = \{g_{A_i} \mid i=1, 2, \dots, r\}$ is a set $S = \{g_{E_k} \mid k=1, 2, \dots, n\}$ of *disjoint* spatial objects, defined as follows: For a quantum $q \in Q_{\text{SURFACE}}$ and a given $k \in I, 1 \leq k \leq n$,

$$q \subseteq g_{E_k} \Leftrightarrow (\exists i, 1 \leq i \leq r)(g_{A_i} \text{ surrounds } q).$$

As an example, if S_A consists of the spatial objects whose geometric representation is shown in Figure 3.8(a) then its spatial envelope consists of the spatial objects whose geometric representation is shown in Figure 3.8(b).

Definition 3.23: The *spatial buffer* of a set of spatial objects $S_A = \{g_{A_i} \mid i=1, 2, \dots, r\}$, within a distance of $d \in \mathbb{R}, d > 0$, is a set

$S = \{g_{B_k} \mid k=1, 2, \dots, n\}$ of *disjoint* spatial objects, defined as follows: For a quantum $q \in Q_{PS}$ and a given $k \in I$, $1 \leq k \leq n$,

$$q \subseteq g_{B_k} \Leftrightarrow (\exists i, 1 \leq i \leq r)(distance(g_{A_i}, q) < d).$$

As an example, if $S_A = \{g_1\}$, where the geometric representation of g_1 is shown in Figure 3.9(a) then its spatial buffer within a distance of 4 is $\{g_{B1}\}$, shown in Figure 3.9(b). Similarly, if $S_B = \{g_i \mid i = 2, 3, 4\}$, where the geometric representation of g_i is shown in Figure 3.9(a), then its spatial buffer within a distance of 1 is $S = \{g_{Bk} \mid k = 2, 3\}$, where g_{Bk} are shown in Figure 3.9(b).

3.7 Conclusions

A set of spatial data types have been defined that enable the discrete representation of 2-d spatial objects. The characteristics of these types can be summarized as follows:

- A spatial object is a connected closed subset of R^2 .
- A spatial object is the union of a finite number of subsets of R^2 , namely *quanta of space*. To the best of this author's knowledge, such *quanta* have not been considered in any other spatial data modelling approach.
- The empty set is not a spatial object.
- The POINT, PLINE and PSURFACE types match human perception. In this sense, it is estimated that they are user friendly.
- The objects are spatially compatible. This enables objects of distinct types to be involved in set operations such as union, except and intersect.
- Beyond points, pure lines and pure surfaces, spatial objects of practical interest are also supported, *hybrid surfaces*.
- Although the formalism considers 2-d spatial objects, its generalization to n-d spatial objects is straightforward.

In conclusion, the spatial data types defined in this chapter satisfy all the relevant properties specified in Section 2.8. Finally, operations between sets of spatial objects have been defined.

CHAPTER 4

FORMALISM FOR SPATIAL DATA MANAGEMENT

4.1 Introduction

A relational algebra is formalized in this chapter, which enables the management of 2-d spatial objects [LTV99, LVT99, LVT00]. The algebra consists of just few kernel operations, in that only two more, *Unfold* and *Fold*, had to be added to the five primitive operations of Codd's relational algebra [Co70, Co72]. Hence, all the remainder operations are defined in terms of seven kernel operations. The formalism satisfies the properties specified in Section 2.8 that are related to the management of spatial data. Some properties of the operations defined in this chapter can be found in Appendix A. Although the model is database-centric, it provides the necessary functionality for the management of thematic maps [SV89]. The remainder of this chapter can be outlined as follows. The valid data structures of the model are defined in Section 4.2. Relational operations are formalized in Section 4.3. Finally, conclusions are drawn in the last section.

4.2 Data Structures

A relation is defined in the known way, except that the underlying domain of one or more of its attributes can now be of some spatial data type.

Definition 4.1: If A_1, A_2, \dots, A_n are distinct names and DT_1, DT_2, \dots, DT_n , are names of data types, not necessarily distinct, then a *relation* R with scheme $R(A_1 | DT_1, A_2 | DT_2, \dots, A_n | DT_n)$ is defined as a finite subset of $DT_1 \times DT_2 \times \dots \times DT_n$.

Hence a relation consists of elements of the form $t_i = (a_{i1}, a_{i2}, \dots, a_{in})$, where $a_{ij} \in DT_j$ for all $j = 1, 2, \dots, n$. It is said that a_{ij} is the value of *tuple* t_i for *attribute* A_j and that DT_j is the domain of A_j . By definition, a spatial data type may be

the domain of more than one attribute. In case that the domain of the attributes is immaterial, the relation scheme is denoted either as $R(A_1, A_2, \dots, A_n)$ or as $R(\mathbf{A})$, where \mathbf{A} represents a set of one or more attributes. G , perhaps subscripted, is used to denote an attribute of some spatial data type. Hence, $R(\mathbf{A}, G)$ denotes a relation scheme with attributes $\mathbf{A} \cup \{G\}$. In this case \mathbf{A} may be the empty set. By $R(\mathbf{A}, G)$ it is not necessarily assumed that G is the last attribute of R . Finally, (\mathbf{a}, g) denotes a tuple of a relation with scheme $R(\mathbf{A}, G)$, where \mathbf{a} denotes the values of the tuple for attributes \mathbf{A} and g is the value (spatial object) of the tuple for attribute G . To ease discussion, a relation with at least one attribute of a spatial type is called *spatial relation*. Such an example is the relation in Figure 4.1(a) whose scheme is

$R(\text{Name} \mid \text{CHAR}(20), G \mid \text{SURFACE})$.

As can be seen in Figure 4.1(b), the geometric representation of the objects recorded in attribute G is a point, a pure line and a pure surface.

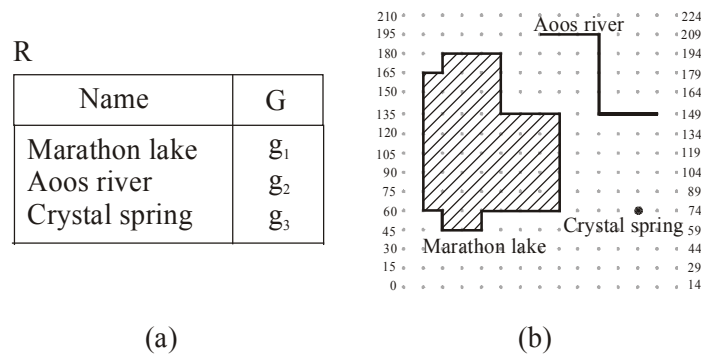


Figure 4.1: Spatial objects within a relation and their geometric representation.

One relation of particular interest is $\text{SURF_ALL}(G \mid \text{SURFACE})$, which contains a single tuple $t = (g_{\text{SURF_ALL}})$, where $g_{\text{SURF_ALL}}$ is the spatial object defined in Section 3.3. SURFACE has been defined as the domain of G because (i) it is the most general type and (ii) SURF_ALL is used in a series of operations where union-compatibility (defined in the next section) is a requirement.

4.3 Relational Algebra Operations

The relational algebra operations are formalized in this section. The general syntax of a unary relational operation is

$$S = \text{Operation}[\text{Parameters}](R),$$

where *Operation* is the operation name, R is the relation to which *Operation* is applied and S is the result relation. Parameters vary from one operation to another. Similarly, the general syntax of a binary operation is

$$S = R_1 \textit{Operation}[\textit{Parameters}] R_2$$

where now R_1 and R_2 are the relations to which *Operation* is applied. Parameters may be missing in some binary operations. The operations defined in this chapter are listed in Figure 4.2.

Operation Name	Scheme of Input Relation(s)	Syntax	Scheme of Output Relation	Remarks
Union	$R_1(\mathbf{A}), R_2(\mathbf{A})$	$R_1 \textit{Union} R_2$	$S(\mathbf{A})$	
Except	$R_1(\mathbf{A}), R_2(\mathbf{A})$	$R_1 \textit{Except} R_2$	$S(\mathbf{A})$	
Project	$R(\mathbf{A}, \mathbf{B})$	$\textit{Project}[\mathbf{A}](R)$	$S(\mathbf{A})$	
Select	$R(\mathbf{A})$	$\textit{Select}[F](R)$	$S(\mathbf{A})$	F is a wff
Product	$R_1(\mathbf{A}), R_2(\mathbf{B})$	$R_1 \textit{Product} R_2$	$S(\mathbf{A}, \mathbf{B})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Intersect	$R_1(\mathbf{A}), R_2(\mathbf{A})$	$R_1 \textit{Intersect} R_2$	$S(\mathbf{A})$	
Inner Theta Join	$R_1(\mathbf{A}), R_2(\mathbf{B})$	$R_1 \textit{ITJoin}[F] R_2$	$S(\mathbf{A}, \mathbf{B})$	F is a wff $\mathbf{A} \cap \mathbf{B} = \emptyset$
Inner Natural Join	$R_1(\mathbf{A}, \mathbf{C}), R_2(\mathbf{C}, \mathbf{B})$	$R_1 \textit{INJoin} R_2$	$S(\mathbf{A}, \mathbf{C}, \mathbf{B})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Left Natural Join	$R_1(\mathbf{A}, \mathbf{C}), R_2(\mathbf{C}, \mathbf{B})$	$R_1 \textit{LNJoin} R_2$	$S(\mathbf{A}, \mathbf{C}, \mathbf{B})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Right Natural Join	$R_1(\mathbf{A}, \mathbf{C}), R_2(\mathbf{C}, \mathbf{B})$	$R_1 \textit{RNJoin} R_2$	$S(\mathbf{A}, \mathbf{C}, \mathbf{B})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Full Natural Join	$R_1(\mathbf{A}, \mathbf{C}), R_2(\mathbf{C}, \mathbf{B})$	$R_1 \textit{FNJoin} R_2$	$S(\mathbf{A}, \mathbf{C}, \mathbf{B})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Unfold	$R(\mathbf{A}, \mathbf{G})$	$\textit{Unfold}[\mathbf{G}](R)$	$S(\mathbf{A}, \mathbf{G})$	
Fold	$R(\mathbf{A}, \mathbf{G})$	$\textit{Fold}[\mathbf{G}](R)$	$S(\mathbf{A}, \mathbf{G})$	
Quantum Union	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{A}, \mathbf{G})$	$R_1 \textit{QUnion}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{G})$	
Quantum Except	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{A}, \mathbf{G})$	$R_1 \textit{QExcept}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{G})$	
Quantum Intersect	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{A}, \mathbf{G})$	$R_1 \textit{QIntersect}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{G})$	
Pair-Wise Union	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{B}, \mathbf{G})$	$R_1 \textit{WUnion}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{B}, \mathbf{G})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Pair-Wise Except	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{B}, \mathbf{G})$	$R_1 \textit{WExcept}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{B}, \mathbf{G})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Pair-Wise Intersect	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{B}, \mathbf{G})$	$R_1 \textit{WIntersect}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{B}, \mathbf{G})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Inner Overlay	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{B}, \mathbf{G})$	$R_1 \textit{IOverlay}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{B}, \mathbf{G})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Left Overlay	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{B}, \mathbf{G})$	$R_1 \textit{LOverlay}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{B}, \mathbf{G})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Right Overlay	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{B}, \mathbf{G})$	$R_1 \textit{ROverlay}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{B}, \mathbf{G})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Full Overlay	$R_1(\mathbf{A}, \mathbf{G}), R_2(\mathbf{B}, \mathbf{G})$	$R_1 \textit{FOverlay}[\mathbf{G}] R_2$	$S(\mathbf{A}, \mathbf{B}, \mathbf{G})$	$\mathbf{A} \cap \mathbf{B} = \emptyset$
Complementation	$R(\mathbf{A}, \mathbf{G})$	$\textit{Complementation}[\mathbf{G}](R)$	$S(\mathbf{A}, \mathbf{G})$	
Boundary	$R(\mathbf{A}, \mathbf{G})$	$\textit{Boundary}[\mathbf{G}](R)$	$S(\mathbf{A}, \mathbf{G})$	
Envelope	$R(\mathbf{A}, \mathbf{G})$	$\textit{Envelope}[\mathbf{G}](R)$	$S(\mathbf{A}, \mathbf{G})$	
Compute	$R(\mathbf{A})$	$\textit{Compute}[C_i = f_i(\mathbf{A}_i)](R)$	$S(\mathbf{A}, C_i)$	$\mathbf{A}_i \subseteq \mathbf{A}$
Buffer	$R(\mathbf{A}, \mathbf{D}, \mathbf{G})$	$\textit{Buffer}[\mathbf{D}, \mathbf{G}](R)$	$S(\mathbf{A}, \mathbf{D}, \mathbf{G})$	
Replicate	$R(\mathbf{A}, B_i)$	$\textit{Replicate}[B_i^? : B_i](R)$	$S(\mathbf{A}, B_i, B_i^?)$	

Figure 4.2: List of relational algebra operations defined.

One requirement of some binary operations is that the relations, which are involved in them, must be *union-compatible*. Hence, the relevant definition is given below.

Definition 4.2: Two n -ary relations R_1 and R_2 are *union-compatible* iff for all i , $1 \leq i \leq n$, the name of the i -th attribute of the first relation matches that of the i -th attribute of the second and both of these attributes have the same underlying domain.

It is recalled that all the spatial types defined in this thesis are spatially compatible (Section 3.5). To comply however with the requirement of union-compatibility, it is occasionally necessary to transform two distinct such types to their least common supertype (Definition 3.15). This is achieved by the use of operation *Compute* (see later Definition 4.18).

As a final remark, it is noted that, occasionally, it is necessary to assign a new name to the attribute of a relation. A back arrow (\leftarrow) is used for this purpose. Hence, $R(G_1 \leftarrow G)$ means that attribute G of relation R is renamed to G_1 . Such a renaming may appear in a relational algebra expression in which R is involved. The assumption is that the renaming occurs before the relational algebra expression is evaluated.

4.3.1 Conventional Operations

The model includes all the well-known algebraic operations of Codd's relational algebra [Co70, Co72, Co79]. The five primitive operations are defined as follows.

Definition 4.3: Given two union-compatible relations with scheme $R_1(\mathbf{A})$ and $R_2(\mathbf{A})$, operations *Union* and *Except (Difference)* are defined as follows:

$U = R_1 \text{ Union } R_2$ has scheme $U(\mathbf{A})$ and extension $\{t \mid t \in R_1 \vee t \in R_2\}$.

$E = R_1 \text{ Except } R_2$ has scheme $E(\mathbf{A})$ and extension $\{t \mid t \in R_1 \wedge t \notin R_2\}$.

Definition 4.4: If R is a relation with scheme $R(\mathbf{A}, \mathbf{B})$ then relation

$P = \text{Project}[\mathbf{A}](R)$

has scheme $P(\mathbf{A})$ and extension

$P = \{(\mathbf{a}) \mid (\mathbf{a}, \mathbf{b}) \in R\}$.

Definition 4.5: If R is a relation with scheme $R(\mathbf{A})$ and F is a *well-formed formula* then relation

$S = \text{Select}[F](R)$

has scheme $S(\mathbf{A})$ and extension

$S = \{(\mathbf{a}) \mid (\mathbf{a}) \in R \wedge F(\mathbf{a})\}$.

As is known, a well-formed formula may include comparisons with constants. In the case of spatial objects, it is practically more convenient for a spatial constant to be specified via a graphical user interface. However, for reasons of completeness of the relational algebra, which is defined in this chapter, spatial literals are also defined. Such a literal is defined as a string of spatial quanta

enclosed in simple quotes ('), preceded by an appropriate keyword that identifies the spatial data type. If the format of each q_i , $1 \leq i \leq n$, below is either P_k or H_k or V_k or S_k , for some valid k , then the following spatial literals are defined:

POINT literal: POINT 'P_i', where $P_i \in \text{POINT}$.
 PLINE literal: PLINE 'q₁q₂...q_n', where $g \equiv q_1q_2...q_n \in \text{PLINE}$
 LINE literal: LINE 'q₁q₂...q_n', where $g \equiv q_1q_2...q_n \in \text{LINE}$.
 PSURFACE literal: PSURFACE 'q₁q₂...q_n', where $g \equiv q_1q_2...q_n \in \text{PSURFACE}$.
 SURFACE literal: SURFACE 'q₁q₂...q_n', where $g \equiv q_1q_2...q_n \in \text{SURFACE}$.

A spatial literal may not necessarily be in some canonical form. Note also that the above syntax convention matches fully that of SQL:1999 for types DATE, TIME, TIMESTAMP and INTERVAL.

With reference to Figure 3.2, examples of spatial literals are the following:

- (a) PLINE 'H183'
- (b) PLINE 'V187'
- (c) PSURFACE 'S191'
- (d) PLINE 'H137H138H139H140'
- (f) PSURFACE 'S62S63S64S65S77S78S79S92S93S94S107'
- (h) SURFACE 'S18H19H20H21S22S23S24S33S37'

Definition 4.6: If R_1, R_2 are relations with scheme $R_1(\mathbf{A}), R_2(\mathbf{B})$, respectively, and $\mathbf{A} \cap \mathbf{B} = \emptyset$, then relation

$$P = R_1 \text{ Product } R_2$$

has scheme $P(\mathbf{A}, \mathbf{B})$ and extension

$$P = \{(\mathbf{a}, \mathbf{b}) \mid (\mathbf{a}) \in R_1 \wedge (\mathbf{b}) \in R_2\}.$$

Beyond the previous five primitive operations, some derived have also been defined. Some of them, which are used in this thesis, are the following.

Definition 4.7: Let $R_1(\mathbf{A}), R_2(\mathbf{A}), R_3(\mathbf{B}), R_4(\mathbf{A}, \mathbf{C})$ and $R_5(\mathbf{C}, \mathbf{B})$ be the scheme of five relations, such that R_1 and R_2 are union-compatible and $\mathbf{A} \cap \mathbf{B} = \emptyset$. Let also F be a well-formed formula defined on attributes $\mathbf{A} \cup \mathbf{B}$. The following operations are then defined:

$$R_1 \text{ Intersect } R_2 \equiv R_1 \text{ Except } (R_1 \text{ Except } R_2).$$

$$R_1 \text{ ITJoin}[F] R_3 \equiv \text{Select}[F](R_1 \text{ Product } R_3).$$

$$R_4 \text{ INJoin } R_5 \equiv \text{Project}[\mathbf{A}, \mathbf{C}, \mathbf{B}](R_4 \text{ ITJoin}[\mathbf{C} = \mathbf{D}] R_5(\mathbf{D} \leftarrow \mathbf{C})).$$

Operation *ITJoin* (*Inner Theta Join*) is called *Theta-Join* in [Co79] and just *Join* in [Co70]. Operation *INJoin* (*Inner Natural Join*) is called *Natural Join* in [Co79]. Finally, the *Outer Join* operations [Co79] are required for the formalism of a series of *Overlay* operations (Definition 4.14). These join operations are defined below.

Definition 4.8: Let $R_1(\mathbf{A}, \mathbf{C})$ and $R_2(\mathbf{C}, \mathbf{B})$ be the scheme of two relations, $\mathbf{A} \cap \mathbf{B} = \emptyset$. Let also R_3 and R_4 be relations with scheme $R_3(\mathbf{A})$ and $R_4(\mathbf{B})$, respectively, and extension $\{\text{null}\}$. Finally, let

$$I = R_1 \text{ INJoin } R_2,$$

$$L = (R_1 \text{ Except } (\text{Project}[\mathbf{A}, \mathbf{C}](I))) \text{ Product } R_4,$$

$$R = R_3 \text{ Product } (R_2 \text{ Except } (\text{Project}[\mathbf{C}, \mathbf{B}](I))),$$

Then the following natural join operations are defined:

$$\text{Left Natural Join: } R_1 \text{ LNJoin } R_2 \equiv L \text{ Union } I.$$

$$\text{Right Natural Join: } R_1 \text{ RNJoin } R_2 \equiv I \text{ Union } R.$$

$$\text{Full Natural Join: } R_1 \text{ FNJoin } R_2 \equiv (L \text{ Union } I) \text{ Union } R.$$

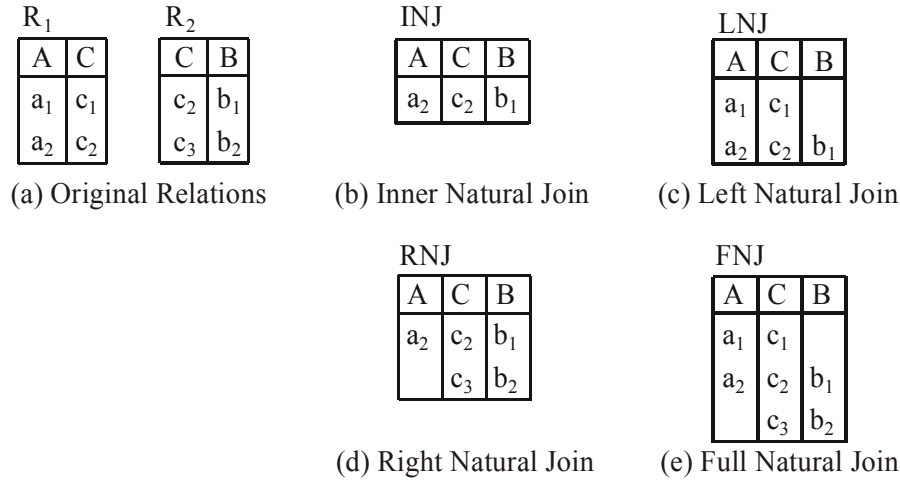


Figure 4.3: Illustration of the result of *Natural Join* operations.

As an example, if R_1 and R_2 are the relations in Figure 4.3(a) then Figure 4.3 shows the extension of

$$\text{INJ} = R_1 \text{ INJoin } R_2,$$

$$\text{LNJ} = R_1 \text{ LNJoin } R_2,$$

$$\text{RNJ} = R_1 \text{ RNJoin } R_2,$$

$$\text{FNJ} = R_1 \text{ FNJoin } R_2.$$

Note that natural join operations have practical interest for the management of conventional data, and are supported in SQL standard (SQL:1992[Iso92], SQL:1999[Iso99]).

4.3.2 Additional Basic Operations

In addition to the previous operations, two more basic operations are now defined, *Unfold* and *Fold*. As will be seen, the remainder operations can be

defined in terms of both the conventional and of these two operations. Moreover, it is estimated that the set of all these operations enable incorporating within the relational model the functionality required for the management of spatial data.

For an informal description of *Unfold*, let $R(\mathbf{A}, G)$ be the scheme of a relation. Let also (\mathbf{a}, g) be any of the tuples in R . Then

$$Unfold[G](R)$$

yields in the result relation the set of tuples $\{(\mathbf{a}, g_i)\}$, where $\{g_i\}$ consists of all the spatial quanta that are subsets of g . The operation is formalized as follows:

R.G	$(Unfold[G](R)).G$	$(Fold[G](R)).G$
POINT	POINT	POINT
PLINE	LINE	PLINE
LINE	LINE	LINE
PSURFACE	SURFACE	PSURFACE
SURFACE	SURFACE	SURFACE

(a) Unary Operations

$R_1.G$	$R_2.G$	$(R_1 \text{ Binary}[G] R_2).G$
POINT	POINT	POINT
PLINE	PLINE	LINE
LINE	LINE	LINE
PSURFACE	PSURFACE	SURFACE
SURFACE	SURFACE	SURFACE

(b) Binary Operations

Figure 4.4: Result data types in spatial operations.

Definition 4.9: If R is a relation with scheme $R(\mathbf{A}, G)$ then relation

$$U = Unfold[G](R)$$

has scheme $U(\mathbf{A}, G)$, where the data type of $U.G$ is that in Figure 4.4(a), and extension

$$\{(\mathbf{a}, q_i) \mid q_i \in Q_{SURFACE} \wedge q_i \subseteq g \wedge (\mathbf{a}, g) \in R\}.$$

To demonstrate the functionality of *Unfold*, let

$$U = Unfold[G](R).$$

Assume also that R is the relation in Figure 4.1. Then U is the relation in Figure 4.5(a) and the geometric representation of the spatial objects recorded in $U.G$ is shown in Figures 4.5(b)-(d). In particular, Figure 4.5(b) shows the spatial objects, which are quantum surfaces, Figure 4.5(c) shows the spatial

objects, which are quantum lines, and Figure 4.5(d) shows those, which are quantum points.

Operation *Fold* is the converse of operation *Unfold*. For an informal description, let $R(\mathbf{A}, \mathbf{G})$ be the scheme of a relation. Let also $\{(\mathbf{a}, \mathbf{g}_i)\}$ be a subset of R consisting of all the tuples with the same value for attributes \mathbf{A} . Then

$$Fold[\mathbf{G}](R)$$

yields in the result relation the set of tuples $\{(\mathbf{a}, \mathbf{g}_k)\}$, where $\{\mathbf{g}_k\}$ is the spatial union of $\{\mathbf{g}_i\}$ and \emptyset . The operation is formalized below.

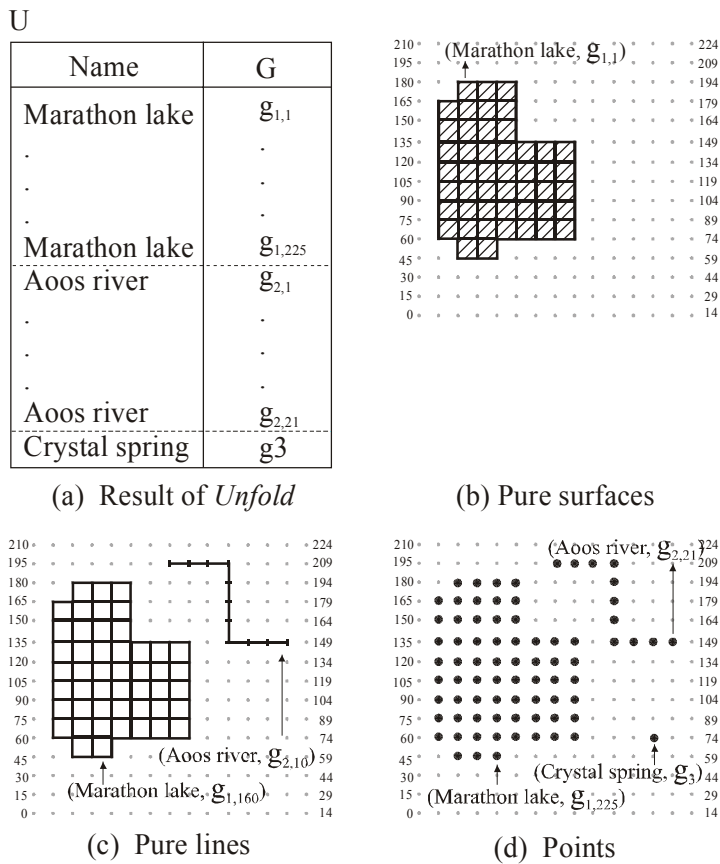


Figure 4.5: Illustration of the result of operation *Unfold*.

Definition 4.10: If R is a relation with scheme $R(\mathbf{A}, \mathbf{G})$ then relation

$$F = Fold[\mathbf{G}](R)$$

has scheme $F(\mathbf{A}, \mathbf{G})$, where the data type of $U.G$ is that in Figure 4.4(a), and extension

$$\{(\mathbf{a}, g = \bigcup_{i=1}^n g_i) \mid (g \text{ is connected}) \wedge \\ ((\mathbf{a}, g_i) \in R, i = 1, 2, \dots, n) \wedge \\ ((\exists(\mathbf{a}, g_k) \in R, k \neq 1, 2, \dots, n)(g \cup g_k \text{ is connected}))\}.$$

To demonstrate the functionality of *Fold*, let

$$F = \text{Fold}[G](R).$$

Assume also that $R = \{(\mathbf{a}, g_A), (\mathbf{a}, g_B)\}$, where g_A and g_B are the spatial objects that appear in one of the Figures 2.9(a)-(i). The spatial union of $\{g_A\}$ and $\{g_B\}$ can also be seen in the same figure. Then F contains one tuple (\mathbf{a}, g_i) for each g_i in this spatial union.

As another example, let $R = \{(\mathbf{a}, g_{A_i})\} \cup \{(\mathbf{a}, g_{B_j})\}$, where g_{A_i} and g_{B_j} are shown in Figure 3.6(a). Then, similarly as before, F contains one tuple (\mathbf{a}, g_i) for each distinct object depicted in Figure 3.6(b). This example is more general, in that the input relation consists of more than two tuples.

It is finally noticed that if the data type of G in $R(\mathbf{A}, G)$ is POINT, then

$$R = \text{Fold}[G](R).$$

Definition 4.11: If R is a relation with scheme $R(\mathbf{A}, G)$ then it is defined that

$$\text{Normalise}[G](R) \equiv \text{Fold}[G](\text{Unfold}[G](R)).$$

It has been shown [LPS95] that

$$\text{Fold}[G](\text{Unfold}[G](R)) = \text{Fold}[G](R).$$

Hence, it also follows that

$$\text{Normalise}[G](R) = \text{Fold}[G](R).$$

One justification, however, for introducing this operation is given in Subsection 4.3.6.

4.3.3 Quantum Operations

Three such operations are defined, *Quantum Union* (*QUnion*), *Quantum Except* (*QExcept*) and *Quantum Intersect* (*QIntersect*). For an informal description, let R_1 and R_2 be two union-compatible relations with respective scheme $R_1(\mathbf{A}, G)$ and $R_2(\mathbf{A}, G)$. Let also

$$S_1 = \{(\mathbf{a}, g_i) \mid i = 1, 2, \dots, p\} \quad (S_2 = \{(\mathbf{a}, g_j) \mid j = 1, 2, \dots, q\})$$

be a subset of R_1 (R_2), consisting of all the tuples with the same value for attributes \mathbf{A} . For these S_1 and S_2 ,

- $R_1 \text{ QUnion}[G] R_2$,
- $R_1 \text{ QExcept}[G] R_2$,
- $R_1 \text{ QIntersect}[G] R_2$,

yields in the result relation the set of tuples $\{(\mathbf{a}, g_k) \mid k = 1, 2, \dots, r\}$, where $\{g_k\}$ is, respectively, the

- spatial union of $\{g_i\}$ and $\{g_j\}$,
 - spatial difference of $\{g_j\}$ from $\{g_i\}$,
 - spatial intersection of $\{g_i\}$ and $\{g_j\}$,
- $i = 1, 2, \dots, p, j = 1, 2, \dots, q$. These operations are formalized below.

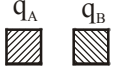
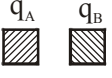




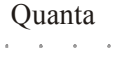


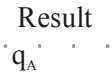
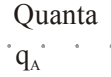




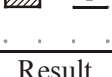
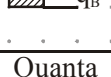
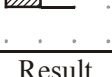
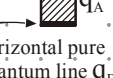




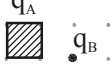





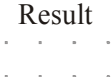
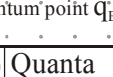



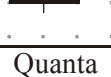

(a) Quanta 	Result 	(b) Quanta 	Result 	(c) Quanta 	Result 
(d) Quanta 	Result 	(e) Quanta 	Result 	(f) Quanta 	Result 
(g) Quanta  horizontal pure quantum line q_B	Result 	(h) Quanta  vertical pure quantum line q_B	Result 	(i) Quanta 	Result 
(j) Quanta  quantum point q_B	Result 	(k) Quanta 	Result 	(l) Quanta 	Result 
(m) Quanta 	Result 	(n) Quanta 	Result 	(o) Quanta 	Result 
(p) Quanta  quantum point q_B	Result 	(q) Quanta 	Result 	(r) Quanta 	Result 

Figure 4.6: Examples of *spatial union* of quantum q_A with quantum q_B .

Definition 4.12: Let $R_1(A, G)$ and $R_2(A, G)$ be two union-compatible relations. Then the following operations are defined:

Quantum Union:

$$R_1 \text{ } QUnion[G] \text{ } R_2 \equiv Fold[G](Unfold[G](R_1) \text{ } Union \text{ } Unfold[G](R_2)).$$

Quantum Except:

$$R_1 \text{ } QExcept[G] \text{ } R_2 \equiv Fold[G](Unfold[G](R_1) \text{ } Except \text{ } Unfold[G](R_2)).$$

Quantum Intersect:

$$R_1 \text{ } Q\text{Intersect}[G] \text{ } R_2 \equiv \text{Fold}[G](\text{Unfold}[G](R_1) \text{ } \text{Intersect} \text{ } \text{Unfold}[G](R_2)).$$

If R is the result of any of these operations, it follows that its scheme is $R(\mathbf{A}, G)$, where the domain of $R.G$ is shown in Figure 4.4(b).

To demonstrate the functionality of the operations, let

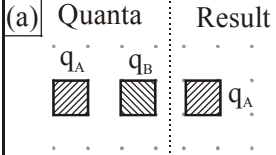

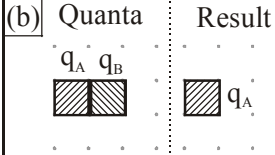

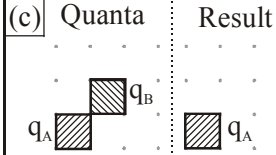

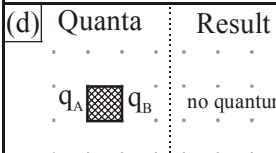
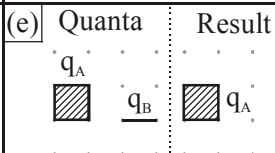

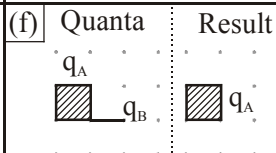

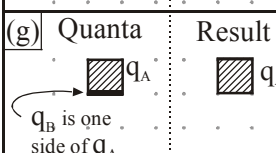

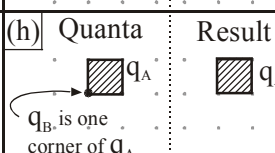

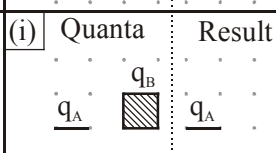

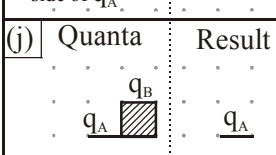

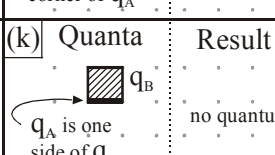
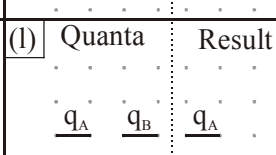

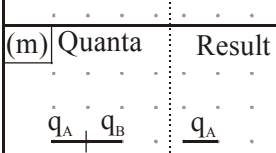

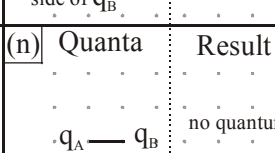
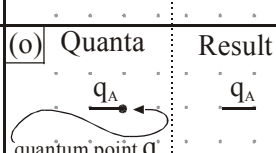

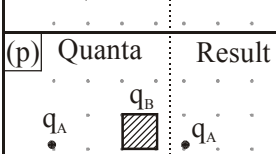

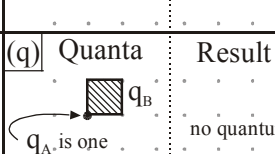
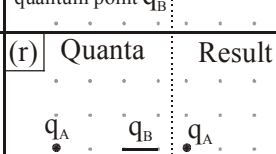

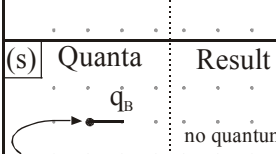
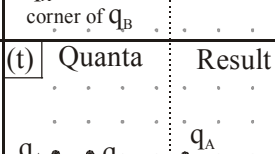

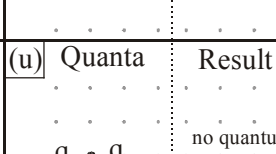
(a)	Quanta 	Result 	(b)	Quanta 	Result 	(c)	Quanta 	Result 
(d)	Quanta 	Result no quantum	(e)	Quanta 	Result 	(f)	Quanta 	Result 
(g)	Quanta 	Result 	(h)	Quanta 	Result 	(i)	Quanta 	Result 
(j)	Quanta 	Result 	(k)	Quanta 	Result no quantum	(l)	Quanta 	Result 
(m)	Quanta 	Result 	(n)	Quanta 	Result no quantum	(o)	Quanta 	Result 
(p)	Quanta 	Result 	(q)	Quanta 	Result no quantum	(r)	Quanta 	Result 
(s)	Quanta 	Result no quantum	(t)	Quanta 	Result 	(u)	Quanta 	Result no quantum

Figure 4.7: Examples of *spatial difference* of quantum q_B from quantum q_A .

$$QU = R_1 \text{ } QUnion[G] \text{ } R_2,$$

$$QE = R_1 \text{ } QExcept[G] \text{ } R_2,$$

$$QI = R_1 \text{ } QIntersect[G] \text{ } R_2.$$

Assume also that $R_1 = \{(\mathbf{a}, g_A)\}$ and $R_2 = \{(\mathbf{a}, g_B)\}$, i.e. each of them consists of one tuple, where g_A and g_B are the spatial objects that appear in one of the Figures 2.9(a)-(i) (Figures 2.10(a)-(k), Figures 2.11(a)-(l)). The spatial union (difference, intersection) of $\{g_A\}$ and $\{g_B\}$ can also be seen in the same figure.

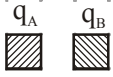






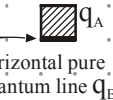

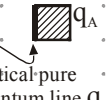

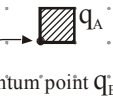



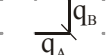

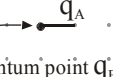


(a)	Quanta 	Result no quantum	(b)	Quanta 	Result q	(c)	Quanta 	Result • q
(d)	Quanta 	Result 	(e)	Quanta 	Result no quantum	(f)	Quanta 	Result • q
(g)	Quanta  horizontal pure quantum line q_B	Result 	(h)	Quanta  vertical pure quantum line q_B	Result q_B	(i)	Quanta 	Result no quantum
(j)	Quanta  quantum point q_B	Result • q_B	(k)	Quanta 	Result no quantum	(l)	Quanta 	Result • q
(m)	Quanta 	Result no quantum	(n)	Quanta 	Result • q	(o)	Quanta 	Result no quantum
(p)	Quanta  quantum point q_B	Result • q_B	(q)	Quanta 	Result no quantum	(r)	Quanta 	Result • q_A

Figure 4.8: Examples of *spatial intersection* of quantum q_A with quantum q_B .

Then QU (QE, QI) contains one tuple (\mathbf{a}, g_i) for each g_i in this spatial union (difference, intersection).

As another example, let $R_1 = \{(\mathbf{a}, g_{Ai})\}$ and $R_2 = \{(\mathbf{a}, g_{Bj})\}$, where g_{Ai} and g_{Bj} are shown in Figure 3.6(a). Then, similarly as before, QU (QE, QI) contains one tuple (\mathbf{a}, g_i) for each distinct object depicted in Figure 3.6(b) (Figure 3.6(c), Figure 3.6(e)). This example is more general, in that each input relation consists of more than one tuple.

As a final example, let $R_1 = \{(\mathbf{a}, q_A)\}$ and $R_2 = \{(\mathbf{a}, q_B)\}$, i.e. each of them consists of one tuple, where q_A and q_B are spatial quanta. Figure 4.6 (Figure 4.7, Figure 4.8) shows representative relative positions of two such distinct quanta q_A and q_B . The spatial union (difference, intersection) of $\{q_A\}$ and $\{q_B\}$ is also shown in the figure. Then QU (QE, QI) contains one tuple (\mathbf{a}, g_i) for each g_i in this spatial union (difference, intersection).

It is worth noting that, although all these operations have practical interest, many spatial data modelling approaches do not support their functionality.

4.3.4 Pair-Wise Operations

Although practically useful, the quantum operations must be performed between two union-compatible relations. More generally, however, it is desirable to obtain the spatial union, spatial difference and spatial intersection of sets of spatial objects that are stored in non-union-compatible relations. This is achieved by the definition of *pair-wise operations*, namely *Pair-Wise Union* (*WUnion*), *Pair-Wise Except* (*WExcept*) and *Pair-Wise Intersect* (*WIntersect*). For an informal description, let $R_1(\mathbf{A}, G)$ and $R_2(\mathbf{B}, G)$, $\mathbf{A} \cap \mathbf{B} = \emptyset$, be the schemes of two relations. Let also

$$S_1 = \{(\mathbf{a}, g_i) \mid i = 1, 2, \dots, p\} \quad (\{(\mathbf{b}, g_j)\} \mid j = 1, 2, \dots, q)$$

be a subset of R_1 (R_2), consisting of all the tuples with the same value for attributes \mathbf{A} (\mathbf{B}). For these S_1 and S_2 ,

- R_1 *WUnion*[G] R_2 ,
- R_1 *WExcept*[G] R_2 ,
- R_1 *WIntersect*[G] R_2 ,

yields in the result relation the set of tuples $\{(\mathbf{a}, \mathbf{b}, g_k) \mid k = 1, 2, \dots, r\}$, where $\{g_k\}$ is, respectively, the

- spatial union of $\{g_i\}$ and $\{g_j\}$,
- spatial difference of $\{g_i\}$ from $\{g_j\}$,
- spatial intersection of $\{g_i\}$ and $\{g_j\}$,

$i = 1, 2, \dots, p, j = 1, 2, \dots, q$. These operations are formalized below.

Definition 4.13: Let R_1, R_2 be relations with scheme $R_1(\mathbf{A}, G), R_2(\mathbf{B}, G)$, respectively, where $\mathbf{A} \cap \mathbf{B} = \emptyset$ and $R_1.G, R_2.G$ are of the same data type. If

$$\begin{aligned} WR_1 &= Project[A, B, G](R_1 Product R_2(G_1 \leftarrow G)), \\ WR_2 &= Project[A, B, G](R_1(G_1 \leftarrow G) Product R_2), \end{aligned}$$

then the three following pair-wise operations are defined:

Pair-Wise Union: $R_1 \text{ } WUnion[G] \text{ } R_2 \equiv WR_1 \text{ } QUnion[G] \text{ } WR_2.$

Pair-Wise Intersect: $R_1 \text{ } WIntersect[G] \text{ } R_2 \equiv WR_1 \text{ } QIntersect[G] \text{ } WR_2.$

Pair-Wise Except: $R_1 \text{ } WExcept[G] \text{ } R_2 \equiv WR_1 \text{ } QExcept[G] \text{ } WR_2.$

As is obvious from the definition, the attributes of both WR_1 and WR_2 are \mathbf{A} , \mathbf{B} , G . Hence the scheme of the result relation R is $R(\mathbf{A}, \mathbf{B}, G)$. It is also noted that if $(\mathbf{a}, g_1) \in R_1$ and $(\mathbf{b}, g_2) \in R_2$ then $(\mathbf{a}, \mathbf{b}, g_1) \in WR_1$ and $(\mathbf{a}, \mathbf{b}, g_2) \in WR_2$. Hence, the extension of R is the one obtained by the application of the respective quantum operation.

To demonstrate the functionality of the operations, let

$WU = R_1 \text{ } WUnion[G] \text{ } R_2,$

$WE = R_1 \text{ } WExcept[G] \text{ } R_2,$

$WI = R_1 \text{ } WIntersect[G] \text{ } R_2.$

Assume also that $R_1 = \{(\mathbf{a}, g_A)\}$ and $R_2 = \{(\mathbf{b}, g_B)\}$, i.e. each of them consists of one tuple, where g_A and g_B are the spatial objects that appear in one of the Figures 2.9(a)-(i) (Figures 2.10(a)-(k), Figures 2.11(a)-(l)). The spatial union (difference, intersection) of $\{g_A\}$ and $\{g_B\}$ can also be seen in the same figure. Then WU (WE , WI) contains one tuple $(\mathbf{a}, \mathbf{b}, g_i)$ for each g_i in this spatial union (difference, intersection).

As another example, let $R_1 = \{(\mathbf{a}, g_{Ai})\}$ and $R_2 = \{(\mathbf{b}, g_{Bj})\}$, where g_{Ai} and g_{Bj} are shown in Figure 3.6(a). Then, similarly as before, WU (WE , WI) contains one tuple $(\mathbf{a}, \mathbf{b}, g_i)$ for each distinct object depicted in Figure 3.6(b) (Figure 3.6(c), Figure 3.6(e)). This example is more general, in that each input relation consists of more than one tuple.

As a final example, let $R_1 = \{(\mathbf{a}, q_A)\}$ and $R_2 = \{(\mathbf{b}, q_B)\}$, i.e. each of them consists of one tuple, where q_A and q_B are spatial quanta. Figure 4.6 (Figure 4.7, Figure 4.8) shows representative relative positions of two such distinct quanta q_A and q_B . The spatial union (difference, intersection) of $\{q_A\}$ and $\{q_B\}$ is also shown in the figure. Then WU (WE , WI) contains one tuple $(\mathbf{a}, \mathbf{b}, g_i)$ for each g_i in this spatial union (difference, intersection).

It is worth noting that these operations do not cause the data loss problems identified in Subsection 2.5.8. For example, the demonstration of the functionality of *WIntersect* shows that all the points, pure lines, pure surfaces and hybrid surfaces of a spatial intersection appear in the result relation. Note however that the majority of the various spatial data modelling approaches lack this capability, as has been shown in Subsection 2.5.8.

4.3.5 Overlay Operations

As is well-known, *Overlay* is a most useful operation in spatial data management. In the present model four distinct such operations are defined namely, *Inner Overlay* (*IOverlay*), *Left Overlay* (*LOverlay*), *Right Overlay* (*ROverlay*) and *Full Overlay* (*FOverlay*). They are formalized in terms of the respective *Inner*, *Left*, *Right* and *Full Natural Join* operations. For an informal description of the operations, let $R_1(\mathbf{A}, G)$ and $R_2(\mathbf{B}, G)$, $\mathbf{A} \cap \mathbf{B} = \emptyset$, be the schemes of two relations. Let also $\{(\mathbf{a}, g_{Ai})\}$ ($\{(\mathbf{b}, g_{Bj})\}$) be a subset of R_1 (R_2) consisting of all the tuples with the same value for attributes \mathbf{A} (\mathbf{B}). Then these tuples yield in

$$IO = R_1 \text{ } I\text{Overlay}[G] R_2$$

the set of tuples $\{(\mathbf{a}, \mathbf{b}, g_{Ik})\}$, where $\{g_{Ik}\}$ is the spatial intersection of $\{g_{Ai}\}$ and $\{g_{Bj}\}$.

Operation

$$LO = R_1 \text{ } L\text{Overlay}[G] R_2$$

yields in the result relation the tuples in IO and also all the tuples $\{(\mathbf{a}, \text{null}, g_{Ek})\}$, where $\{g_{Ek}\}$ is the spatial difference of $\{g_{Bj}\}$ from $\{g_{Ai}\}$.

Operation

$$RO = R_1 \text{ } R\text{Overlay}[G] R_2$$

yields in the result relation the tuples in IO and also all the tuples $\{(\text{null}, \mathbf{b}, g_{Ek})\}$, where $\{g_{Ek}\}$ is the result of the spatial difference of $\{g_{Ai}\}$ from $\{g_{Bj}\}$.

Finally, operation

$$FO = R_1 \text{ } F\text{Overlay}[G] R_2$$

yields in the result relation all the tuples in IO, LO and RO.

The operations are formalized below.

Definition 4.14: Let $R_1(\mathbf{A}, G)$ and $R_2(\mathbf{B}, G)$ be the schemes of two relations, where $\mathbf{A} \cap \mathbf{B} = \emptyset$ and $R_1.G, R_2.G$ are of the same data type. Let also

$$UR_1 = \text{Unfold}[G](R_1),$$

$$UR_2 = \text{Unfold}[G](R_2).$$

Then the following operations are defined:

$$\text{Inner Overlay: } R_1 \text{ } I\text{Overlay}[G] R_2 \equiv \text{Fold}[G](UR_1 \text{ } IN\text{Join } UR_2).$$

$$\text{Left Overlay: } R_1 \text{ } L\text{Overlay}[G] R_2 \equiv \text{Fold}[G](UR_1 \text{ } LN\text{Join } UR_2).$$

$$\text{Right Overlay: } R_1 \text{ } R\text{Overlay}[G] R_2 \equiv \text{Fold}[G](UR_1 \text{ } RN\text{Join } UR_2).$$

$$\text{Full Overlay: } R_1 \text{ } F\text{Overlay}[G] R_2 \equiv \text{Fold}[G](UR_1 \text{ } FN\text{Join } UR_2).$$

If R is the result of any of these operations, it can easily be seen that its scheme is $R(\mathbf{A}, \mathbf{B}, G)$.

To demonstrate the functionality of the operations, let

$$R_1 = \{(\mathbf{a}, g_{Ai})\},$$

$$R_2 = \{(\mathbf{b}, g_{Bj})\},$$

where g_{A_i} and g_{B_j} are shown in Figure 3.6(a). Consider also

$I = \{(\mathbf{a}, \mathbf{b}, g_{I_k})\}$, where g_{I_k} are shown in Figure 3.6(e),

$L = \{(\mathbf{a}, \text{null}, g_{L_k})\}$, where g_{L_k} are shown in Figure 3.6(c),

$R = \{(\text{null}, \mathbf{b}, g_{R_k})\}$, where g_{R_k} are shown in Figure 3.6(d).

Then the extension of

- $IO = R_1 IOverlay[G] R_2$ matches that of relation I.
- $LO = R_1 LOverlay[G] R_2$ matches that of relation L *Union* I.
- $RO = R_1 ROverlay[G] R_2$ matches that of relation I *Union* R.
- $FO = R_1 FOverlay[G] R_2$ matches that of relation (L *Union* I) *Union* R.

It can be easily verified that the result of $R_1 IOverlay[G] R_2$ is identical to that of $R_1 WIntersect[G] R_2$, defined in the previous subsection.

It is noticed that the functionality of the various overlay operations does not restrict to only pure surfaces (Subsection 2.5.9). In addition, they can be applied to two spatial objects one of which is, for example, a pure line whereas the other is a pure surface, provided that both of them are recorded in some attribute G of type SURFACE. Another observation is that if R_1 contains two tuples (\mathbf{a}_1, g_1) and (\mathbf{a}_2, g_2) , and R_2 contains (\mathbf{b}, g_3) , then the result of any of these operations returns separately the overlay of g_1 with g_3 and that of g_2 with g_3 , even if g_1 and g_2 have points in common. Contrary to this, some of the other approaches lack this capability.

4.3.6 Other Operations of Spatial Interest

The operations defined in this subsection enable obtaining the spatial complementation, spatial boundary, spatial envelope and spatial buffer of a set of spatial objects that are recorded in a relation. To illustrate the first of them, let $R(\mathbf{A}, G)$ be the scheme of a relation R. Let also $\{(\mathbf{a}, g_i)\}$ be a subset of R, consisting of all the tuples with the same value for attributes \mathbf{A} . Then

$Complementation[G](R)$

yields in the result relation the set of tuples $\{(\mathbf{a}, g_{Ck})\}$, where $\{g_{Ck}\}$ is the spatial complementation of $\{g_i\}$. If SURF_ALL(G) is the relation defined in Section 4.2, then the relevant formalization is given below.

Definition 4.15: If R is a relation with scheme $R(\mathbf{A}, G | SURFACE)$ then

$Complementation[G](R)$

is defined as

$SURF_ALL WExcept[G] R.$

If C is the result relation, its scheme is $C(\mathbf{A}, G | SURFACE)$. As an example, let $R = \{(\mathbf{a}, g_i)\}$, where the geometric representation of each g_i is depicted in Figure 3.7(a). If

$C = Complementation[G](R)$

then $C = \{(\mathbf{a}, g_{Ck})\}$, where the geometric representation of each g_{Ck} is depicted in Figure 3.7(b).

For an informal description of another operation, *Boundary*, let R be a relation with scheme $R(\mathbf{A}, G)$. Let also $\{(\mathbf{a}, g_i)\}$ be the subset of the tuples of R with the same value for attributes \mathbf{A} . Then

$$\text{Boundary}[G](R)$$

yields in the result relation the set of tuples $\{(\mathbf{a}, g_{Bk})\}$, where $\{g_{Bk}\}$ is the spatial boundary of $\{g_i\}$. This operation is defined in terms of *Complementation*, as follows:

Definition 4.16: If R is a relation with scheme $R(\mathbf{A}, G \mid \text{SURFACE})$, then operation

$$B = \text{Boundary}[G](R)$$

is defined by the following sequence of operations:

1. $TR_1 = \text{Complementation}[G](R)$
2. $B = TR_1 \text{ } Q\text{Intersect}[G] R$

The scheme of the result relation is $B(\mathbf{A}, G \mid \text{SURFACE})$. As an example, if $R = \{(\mathbf{a}, g_i)\}$, where the geometric representation of each g_i is depicted in Figure 3.7(a), then $B = \{(\mathbf{a}, g_{Bk})\}$, where the geometric representation of each g_{Bk} is depicted in Figure 3.7(c).

To illustrate the functionality of *Envelope*, let $R(\mathbf{A}, G)$ be the scheme of a relation R . Let also $\{(\mathbf{a}, g_i)\}$ be the subset of tuples in R with the same value for attributes \mathbf{A} . Then

$$\text{Envelope}[G](R)$$

yields in the result relation the set of tuples $\{(\mathbf{a}, g_{Ek})\}$, where $\{g_{Ek}\}$ is the spatial envelope of $\{g_i\}$. The operation is formalized as follows:

Definition 4.17: If R is a relation with scheme $R(\mathbf{A}, G \mid \text{SURFACE})$ then relation

$$E = \text{Envelope}[G](R)$$

is defined by the following sequence of operations:

1. $TR_1 = \text{Normalise}[G](R)$
2. $TR_2 = \text{Unfold}[G](\text{SURF_ALL})$
3. $TR_3 = TR_1(G_1 \leftarrow G) \text{ } IT\text{Join} [G_1 \text{ surrounds } G] TR_2$
4. $TR_4 = \text{Project}[\mathbf{A}, G](TR_3)$
5. $E = \text{Fold}[G](TR_4)$

Clearly, the scheme of E is $E(\mathbf{A}, G \mid \text{SURFACE})$. Note that the first operation in the definition of this operation could have been *Fold*. The reason for using *Normalise* here is clarified in Chapter 5, where this operation is applied to more than one attribute.

As an example, if $R = \{(\mathbf{a}, g_i)\}$, where the geometric representation of each g_i is depicted in Figure 3.8(a), then $E = \{(\mathbf{a}, g_{Ek})\}$, where the geometric representation of each g_{Ek} is depicted in Figure 3.8(b). From a comparison of g_1 in Figure 3.8(a) and g_{E1} in Figure 3.8(b), it is concluded that *Envelope* eliminates the holes of pure surfaces. Also, a comparison of g_7 in Figure 3.8(a) and g_{E6} in Figure 3.8(b) shows that *Envelope* also yields the pure surface that is surrounded by a circular line. Finally, a comparison of g_2 in Figure 3.8(a) and g_{E2} in Figure 3.8(b) shows a combination of these two functionalities. Note however that, an appropriate combination of *Envelope* with the operations defined so far enable restricting to only one of these two functionalities.

Operation *Compute* enables incorporating functions in the derivation of new relations [LJ88a]. It is included here because it is a requirement for the definition of operation *Buffer*.

Definition 4.18: Let R be a relation with scheme $R(\mathbf{A})$ and let f_1, f_2, \dots, f_m be functions that are applied, respectively, to the sets of attributes $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_m \subseteq \mathbf{A}$. If

$$C = \text{Compute}[C_1 := f_1(\mathbf{A}_1), C_2 := f_2(\mathbf{A}_2), \dots, C_m := f_m(\mathbf{A}_m)](R)$$

it is defined that

$$C = \{(\mathbf{a}, c_1, c_2, \dots, c_m) \mid ((\mathbf{a}) \in R) \wedge (c_1 := f_1(\mathbf{a}_1)) \wedge (c_2 := f_2(\mathbf{a}_2)) \wedge \dots \wedge (c_m := f_m(\mathbf{a}_m))\}.$$

Clearly, the scheme of C is $C(\mathbf{A}, C_1, C_2, \dots, C_m)$ and the underlying domain of attribute C_i , $1 \leq i \leq n$, is the one determined by the application of the respective function f_i . As an example, if the scheme of a relation is $R(\mathbf{A} \mid \text{INTEGER}, \mathbf{B} \mid \text{INTEGER})$ and $C = \text{Compute}[C := \mathbf{A} + \mathbf{B}, D := \mathbf{A} - \mathbf{B}](R)$ then the scheme of C is $C(\mathbf{A} \mid \text{INTEGER}, \mathbf{B} \mid \text{INTEGER}, C \mid \text{INTEGER}, D \mid \text{INTEGER})$ and it consists of the tuples $\{(a, b, c, d) \mid ((a, b) \in R) \wedge (c = a + b) \wedge d = (a - b)\}$. Some functions of spatial interest that can be incorporated in *Compute* are those given in Section 3.5. Of particular interest is the incorporation of spatial data type transformation functions. Their use enables obtaining relations with identical scheme, before they are involved in an operation that requires union-compatibility.

Note by the way that *Compute* enables defining the various *Overlay* operations in a different way. As an example, if $R_1(\mathbf{A}, G)$ and $R_2(\mathbf{B}, G)$, $\mathbf{A} \cap \mathbf{B} = \emptyset$, are the schemes of two relations, where $R_1.G$ and $R_2.G$ are of identical data type, then $FO = R_1 \text{FOverlay}[G] R_2$ can be obtained by the following sequence of operations:

1. $TR_1 = \text{Project}[G](R_1)$
2. $TR_2 = \text{Project}[G](R_2)$
3. $TR_3 = R_1 \text{WExcept}[G] TR_2$
4. $TR_4 = R_2 \text{WExcept}[G] TR_1$

5. $I = R_1 \text{ WIntersect}[G] R_2$
6. $L = \text{Compute}[\mathbf{B} := \text{null}](\text{TR}_3)$
7. $R = \text{Compute}[\mathbf{A} := \text{null}](\text{TR}_4)$
8. $P = I \text{ Union } L$
9. $\text{FO} = P \text{ Union } R$

The definition of operation *Buffer* is based on that of *Compute*. For an informal description, let $R(\mathbf{A}, D \mid \text{REAL}, G \mid \text{SURFACE})$ be the scheme of a relation R , where the value d of every tuple for attribute D satisfies $d > 0$. Then for each subset $\{(\mathbf{a}, d, g_i)\}$ of R with identical values for attributes \mathbf{A} and D ,

$$\text{Buffer}[D, G](R)$$

yields in the result relation a set of tuples $\{(\mathbf{a}, d, g_{\text{BK}})\}$, where $\{g_{\text{BK}}\}$ is the spatial buffer of $\{g_i\}$ within a distance of d . If $\text{SURF_ALL}(G)$ is the relation defined in Section 4.2, the operation is defined as follows:

Definition 4.19: Let R be a relation with scheme $R(\mathbf{A}, D \mid \text{REAL}, G \mid \text{SURFACE})$ and let the value d of every tuple of R for attribute D satisfy $d > 0$. Then

$$B = \text{Buffer}[D, G](R)$$

is defined by the following sequence of operations:

1. $\text{TR}_1 = \text{Unfold}[G](\text{SURF_ALL})$
2. $\text{TR}_2 = R(G_1 \leftarrow G) \text{ Product } \text{TR}_1(G_2 \leftarrow G)$
3. $\text{TR}_3 = \text{Compute}[C := \text{distance}(G_1, G_2)](\text{TR}_2)$
4. $\text{TR}_4 = \text{Select}[C < D](\text{TR}_3)$
5. $\text{TR}_5 = \text{Project}[\mathbf{A}, D, G](\text{TR}_4(G \leftarrow G_2))$
6. $B = \text{Fold}[G](\text{TR}_5)$

Clearly, the scheme of the result relation is $B(\mathbf{A}, D \mid \text{REAL}, G \mid \text{SURFACE})$. As an example, if

$$R = \{(\mathbf{a}, 4, g_1), (\mathbf{a}, 1, g_2), (\mathbf{a}, 1, g_3), (\mathbf{a}, 1, g_4)\},$$

where the geometric representation of each g_i is depicted in Figure 3.9(a), then

$$B = \{(\mathbf{a}, 4, g_{\text{B1}}), (\mathbf{a}, 1, g_{\text{B2}}), (\mathbf{a}, 1, g_{\text{B3}})\},$$

where the geometric representation of each g_{BK} is depicted in Figure 3.9(b).

Note that the second and third tuple of relation R have the same values for attributes \mathbf{A} and D . Hence, these tuples yield one tuple (the third one) in B , due to the application of *Fold* (step 6), in conjunction with the distance of g_2 from g_3 . Contrary to this, tuples with different values for attribute D always yield distinct tuples in the result relation (see for example the first and second tuple of R).

Finally, note that, as an alternative, it is possible to apply the sequence of operations

1. $B = Buffer[D, G](R)$
2. $P = Project[A, G](B)$
3. $F = Fold[G](P)$

and, therefore, two or more tuples with distinct values for attribute D to be combined into one tuple in the result relation.

4.3.7 Enhancement of the Functionality of Operations

Consider two relations with schemes $R_1(A, G)$, $R_2(B, G)$ and respective extensions

$$R_1 = \{(a, g_{A_i}), i = 1, 2\},$$

$$R_2 = \{(b, g_{B_j}), j = 1, 2\},$$

where the geometric representation of g_{A_i} and g_{B_j} is shown, respectively, in Figure 4.9(a) and Figure 4.9(b). Note that the values for attributes **A** of all the tuples in R_1 match. The same is also true for the tuples in R_2 for attributes **B**. If

$$R = R_1 \text{ WExcept}[G] R_2$$

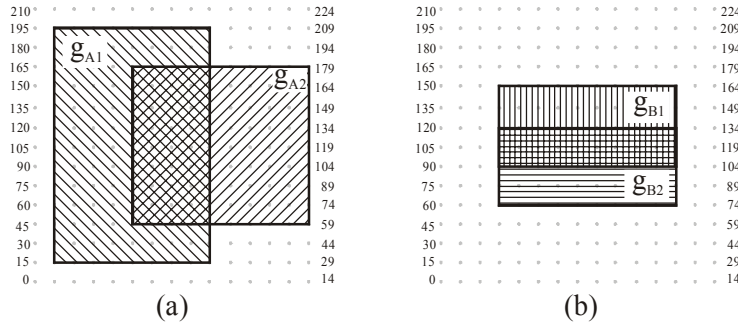


Figure 4.9: Objects involved in *WExcept*.

is issued the scheme of the result relation is $R(A, B, G)$ and its extension is

$$R = \{(a, b, g_{E1})\},$$

where g_{E1} is depicted in Figure 4.10. It is noticed that $\{g_{E1}\}$ is the spatial difference of $\{g_{B1}, g_{B2}\}$ from $\{g_{A1}, g_{A2}\}$. However, this is only one possible result out of four distinct results that might be required. Considering in particular the above extension of R_1 and R_2 some application may ask for *WExcept* to be applied so as to obtain either of the following spatial differences:

- (c1) of set $\{g_{B1}, g_{B2}\}$ from set $\{g_{A1}, g_{A2}\}$ (Figure 4.10),
- (c2) of each of the sets $\{g_{B1}\}, \{g_{B2}\}$ from set $\{g_{A1}, g_{A2}\}$ (Figure 4.11),
- (c3) of set $\{g_{B1}, g_{B2}\}$ from each of the sets $\{g_{A1}\}, \{g_{A2}\}$ (Figure 4.12),
- (c4) of each set $\{g_{B1}\}, \{g_{B2}\}$ from each set $\{g_{A1}\}, \{g_{A2}\}$ (Figure 4.13).

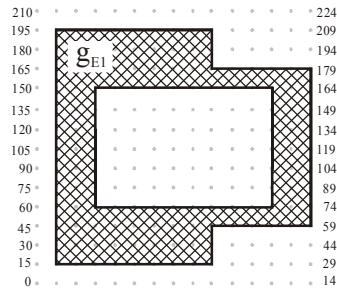


Figure 4.10: Spatial difference of $\{g_{B1}, g_{B2}\}$ from set $\{g_{A1}, g_{A2}\}$.

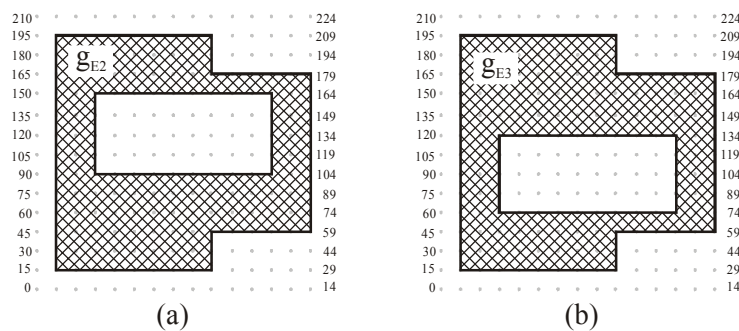


Figure 4.11: Spatial difference of each of $\{g_{B1}\}, \{g_{B2}\}$ from set $\{g_{A1}, g_{A2}\}$.

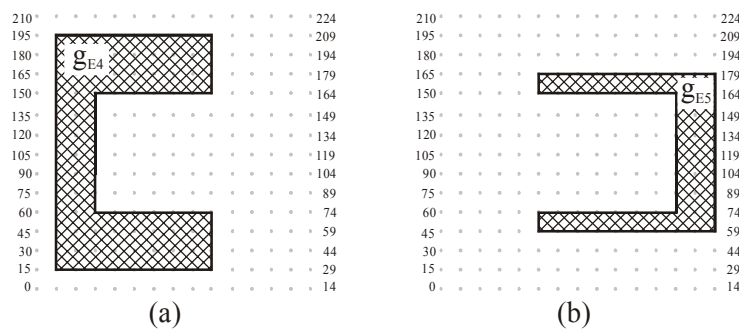


Figure 4.12: Spatial difference of $\{g_{B1}, g_{B2}\}$ from each of $\{g_{A1}\}, \{g_{A2}\}$.

Case (c4) is the most complicated, in that it actually requires the writing of a program that processes separately every tuple in R_1 and every tuple in R_2 and it then applies *WExcept* to each such pair of tuples. Note however that *WExcept* is only an example operation. More generally, however, the following can be noticed:

- (i) Section 3.6 has addressed the formalization of operations between *sets of spatial objects*.
- (ii) Except *Compute*, all the relational algebra operations defined in this chapter *transfer* are *injections* of relevant operations between sets of spatial objects. This has been achieved by considering

subsets of relations with the same value for same attribute, say A , and considering as a set of spatial objects those that are recorded in some other attribute G of these tuples.

To achieve therefore a functionality that requires the involvement of only one tuple at a time, it actually suffices to consider tuples with distinct values for attribute A . This is achieved by the use of operation *Replicate*. Informally, let $R(A, B_1, B_2, \dots, B_n)$ be a relation scheme. If

$$RE = \text{Replicate}[B_1':B_1; B_2':B_2, \dots, B_n':B_n](R)$$

then the scheme of RE is $RE(A, B_1', B_2', \dots, B_n', B_1, B_2, \dots, B_n)$ and for every tuple $(a, b_1, b_2, \dots, b_n)$ in R , RE has a tuple $(a, b_1, b_2, \dots, b_n, b_1, b_2, \dots, b_n)$. Its formalization is as follows:

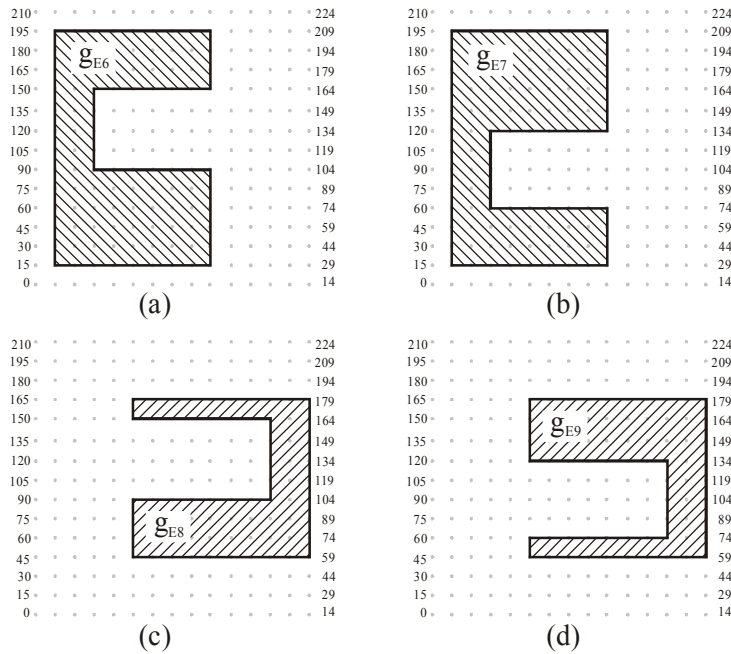


Figure 4.13: Spatial difference of each $\{g_{B1}\}$, $\{g_{B2}\}$ from each $\{g_{A1}\}$, $\{g_{A2}\}$.

Definition 4.20: Let $R(A, B_1, B_2, \dots, B_n)$ be the scheme of a relation R . Then operation

$$\text{Replicate}[B_1':B_1; B_2':B_2, \dots, B_n':B_n](R)$$

is defined as the following sequence of two operations:

1. $TR_1 = R$
 $ITJoin[A = C, B_1 = B_1', B_2 = B_2', \dots, B_n = B_n']$
 $R(C \leftarrow A, B_1' \leftarrow B_1, B_2' \leftarrow B_2, \dots, B_n' \leftarrow B_n)$
2. $RE = \text{Project}[A, B_1', B_2', \dots, B_n', B_1, B_2, \dots, B_n](TR_1)$

For an example, let the scheme of R be $R(A, G)$. If

$$RE = \text{Replicate}[G_A:G](R)$$

then the scheme of the result relation is $RE(\mathbf{A}, G_A, G)$ and its extension is

$$\{(\mathbf{a}, g, g) \mid (\mathbf{a}, g) \in R\}.$$

Given that R does not have two tuples with values matching on both \mathbf{A} and G , it follows that RE does not have two tuples with values matching on both \mathbf{A} and G_A . Let therefore $\mathbf{A}' = \{\mathbf{A}, G_A\}$. Then the scheme of RE is $RE(\mathbf{A}', G_A)$ and RE does not contain two tuples with identical values on \mathbf{A}' . In a similar manner, it is possible to obtain $SE(\mathbf{B}', G_B)$ from $S(\mathbf{B}, G)$. It then follows that the scheme of

$$WE = RE \text{ WExcept}[G] SE$$

is $WE(\mathbf{A}', G_A, \mathbf{B}', G_B, G)$ and consists of tuples

$$(\mathbf{a}, g_1, \mathbf{b}, g_2, g)$$

where g is one of the objects produced by the spatial difference of $\{g_2\}$ from $\{g_1\}$, where $(\mathbf{a}, g_1) \in R$ and $(\mathbf{b}, g_2) \in S$.

In this way, the requirement of case (c4) above is achieved. Similarly, cases (c2) and (c3) are achieved by applying *Replicate* exclusively to R and S , respectively. Another advantage of the solution is that every tuple $(\mathbf{a}, g_1, \mathbf{b}, g_2, g)$ in the result relation enables identifying both the spatial objects g_1 and g_2 and one object g of their spatial difference. Alternatively, attributes $RE.G'$ and $SE.G'$ may be projected out. Finally, note that *Replicate* works equally satisfactorily even if \mathbf{A} is the empty set.

As should be obvious, the use of *Replicate* enables applying four (4) distinct variations of the binary operations

- *WUnion*,
- *WExcept*,
- *WIntersect*,
- *IOverlay*,
- *LOverlay*,
- *ROverlay*,
- *FOverlay*,

and two distinct variations of the unary operations

- *Complementation*,
- *Boundary*,
- *Envelope*,
- *Buffer*.

In this way, the flexibility of the proposed relational algebra has been enriched further.

4.4 Conclusions

An extension of Codd's relational algebra [Co70, Co72, Co79] for the management of 2-d spatial objects was formalized in this chapter. Its characteristics can be summarized as follows:

- Relations are defined in the ordinary way, except that now one or more attribute may be of some spatial data type.
- As a side effect of the previous observation, the geometric representation of objects recorded in one or more spatial attributes of a relation R may be interpreted as a map. In addition, the remainder attributes of R may be used to record values associated with the relevant spatial objects.
- The DBMS provides direct spatial data validation mechanisms to support the spatial data types. Hence, it is not possible, for example, to record a *pure surface* or a *point* as the value of a tuple for an attribute of type PLINE.
- All the spatial objects recorded in a relation are connected.
- The relations are non-nested.
- The management of spatial data actually reduces to the management of relations.
- A unique set of operations enables the handling of both spatial and conventional data. Besides, operations on relations inherit the well-known functionality of operations on thematic maps [SV89].
- A reduced set of kernel relational algebra operations is used. It consists of the known operations of the relational model and of two more operations, *Unfold* and *Fold*. The remainder operations have been defined in terms of them.
- The operations are generic in that their application does not restrict to spatial objects of only one type. For example, *IOverlay* can be applied not only to surfaces but also to lines and points and to combinations of them.
- The model does not face problems of spatial data loss.
- Although the definition of the operations has restricted to the management of 2-d spatial objects, their generalization, so as to apply to n-d spatial objects, is straightforward.

Overall, the spatial model defined in this thesis satisfies all the relevant properties specified in Section 2.8.

CHAPTER 5

TEMPORAL AND SPATIO-TEMPORAL DATA MANAGEMENT

5.1 Introduction

This chapter is concerned with two topics, the management of temporal and of spatio-temporal data, of which the former is a prerequisite for the study of the latter. Regarding the management of temporal data, it is recalled that this thesis focuses on applications like cartography, cadastral systems etc, whose interest restricts to the management of spatial changes at discrete times. Hence, a *discrete* model for time is considered here, as is the case in the majority of the temporal data models that have been defined [JM80, Be82, CT85, Ar86, Ta86, Sn87, Ga88, LJ88a, LJ88b, TG89, Sa90a, Sa90b, TC90, JJ92, CC93, Lo93, NA93]. Note however that the management of temporal data is by itself a distinct research topic. Hence, relevant work in this chapter is actually based on the model defined in [Lo88, LM97], except that now:

- the data types for time are defined in terms of *time quanta* and
- periods of the form $[i, j]$ are considered instead of $[i, j)$, defined in [Lo88, LM97].

As a side effect of the first of these differences, the definitions given, for the application of operations *Unfold* and *Fold* to relations with temporal data, though equivalent with those in [Lo88, LM97], are now based on time quanta. The motivation for the second difference is that closed periods are today a potential ISO standard [Iso96a, Iso96b, Iso96c].

Regarding the management of temporal data, one important conclusion is drawn: Practically, all the operations defined in Chapter 4, for the management of spatial data, can straightforwardly be applied to relations that contain temporal data, i.e. their definition does not have to be revised. In most of the cases, it is also shown that these applications do have practical interest.

The management of spatio-temporal data, including also the *evolution of spatial data with respect to time*, is the second topic this chapter is concerned

with. Relevant to this, one important conclusion is that this management does not require the definition of a new set of operations. Hence, one single set of operations enables the management of either spatial or temporal or spatio-temporal data.

The remainder of this chapter is structured as follows. Quanta and data types for time are formalized in Section 5.2. Predicates and functions that can be applied to time data types are defined in Section 5.3. This set restricts to only those that are necessary for the objectives of this thesis. In Section 5.4 it is shown how temporal data can be represented in relations. Definitions of operations *Unfold* and *Fold* are also given, related to their application to time data types. In Section 5.5 it is shown that all the relational algebra operations defined in Chapter 4 are meaningful when applied to relations that contain temporal data. The issue of *evolution of data with respect to time* is addressed in Section 5.6. The management of spatio-temporal data as well as the evolution of spatial data with respect to time is presented in Section 5.7. Conclusions are drawn in the last section.

5.2 Quanta and Data Types for Time

It is recalled that the elements of $I_n = \{0, 1, \dots, n-1\}$, $n > 0$, are called *l-dimensional (l-d) points* or simply *points*. Then the following definition is given.

Definition 5.1: If $p, q \in I_n$, $p \leq q$, a *period* $[p, q]$ over I_n is defined as the set $[p, q] \equiv \{i \mid i \in I_n \wedge p \leq i \leq q\}$.

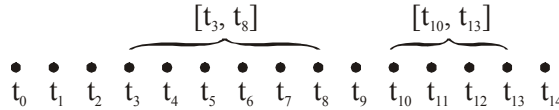


Figure 5.1: Examples of quanta and data types for time.

For the objectives of the present thesis, it is assumed that I_n matches some *time* data type like those supported in SQL, i.e. DATE, TIME, TIMESTAMP. To avoid however restricting to a particular time data type, successive time points of a *generic type* are denoted as $t_0, t_1, t_2, \dots, t_i, t_{i+1}, \dots$, etc., and *periods* of time instants are denoted as $[t_i, t_j]$. Based on this, the following definition is given.

Definition 5.2: Two generic types for time are defined:

- INSTANT $\equiv \{t_0, t_1, \dots, t_{n-1}\}$,
- PERIOD $\equiv \{[t_i, t_j] \mid t_i, t_j \in \text{INSTANT} \wedge t_i \leq t_j\}$.

The elements of the first set are called (*time*) *instants* or *time quanta* and those of the second (*time*) *periods*.

Figure 5.1 depicts the time instants of the generic data type INSTANT, in the case that $n = 15$, and periods over it.

By definition, INSTANT is totally ordered. Taking this into account, the following definition is given.

Definition 5.3: A set P of time instants is *connected* iff it matches the period $[\min(t_i \in P), \max(t_i \in P)]$.

5.3 Predicates and Functions for Time

Predicates and functions for time instants and time periods are defined in [LM97]. Therefore, only those that are necessary for this thesis are given below.

- Since, INSTANT is totally ordered, all the comparison predicates $<$, \leq , $=$, $\langle \rangle$, \geq , $>$ can be used between two instants.

Given two *time periods* $v_1 = [t_i, t_j]$, $v_2 = [t_p, t_q]$, the following binary predicates are defined [LM97]:

- $v_1 = v_2 \Leftrightarrow (t_i = t_p) \wedge (t_j = t_q)$
- $v_1 \langle \rangle v_2 \Leftrightarrow \neg(v_1 = v_2)$
- $v_1 < v_2 \Leftrightarrow (t_i < t_p) \vee (t_i = t_p \wedge t_j < t_q)$
- $v_1 \leq v_2 \Leftrightarrow (v_1 < v_2) \vee (v_1 = v_2)$
- $v_1 > v_2 \Leftrightarrow \neg(v_1 \leq v_2)$
- $v_1 \geq v_2 \Leftrightarrow \neg(v_1 < v_2)$
- $v_1 \text{ surrounds } v_2 \Leftrightarrow t_i \leq t_p \wedge t_j \geq t_q$

It is easy to verify that the definition of these predicates actually matches those given for spatial objects (Section 3.4). The definition of predicate *conductive* between periods has been eliminated because it does not have too much interest. Note however that the above definition of *surrounds* (named *supinterv* in [LM97]) is the reduced form of the relevant definition given for spatial objects. Some more predicates of practical interest are the following:

- $v_1 \text{ } cp \text{ } v_2 \Leftrightarrow v_1 \cap v_2 \neq \emptyset$ (v_1 and v_2 have *common points*)
- $v_1 \text{ } disjoint \text{ } v_2 \Leftrightarrow v_1 \cap v_2 = \emptyset$
- $v_1 \text{ } adjacent \text{ } v_2 \Leftrightarrow (t_j + 1 = t_p) \vee (t_q + 1 = t_i)$

where the addition of a number to a time quantum is assumed to be known.

If t_i is a time instant, then function $ord: \text{INSTANT} \rightarrow I$ is defined as

- $ord(t_i) = i$.

The next function enables obtaining a time instant from an integer:

- $form_instant(i) = t_i \Leftrightarrow 0 \leq i \leq n - 1$.

Each of the above functions is the inverse of the other and it is the analogue of the relevant function defined for spatial points (Section 3.5). It is easy to notice that $t_i \theta t_j \Leftrightarrow ord(t_i) \theta ord(t_j)$, where θ is some comparison predicate.

Given a time period $v = [t_i, t_j]$, the following functions are defined:

- $start(v) \equiv t_i$
- $end(v) \equiv t_j$
- $duration(v) \equiv t_j - t_i + 1$

where the subtraction of one time quantum from another is assumed to be known.

If $v_1 = [t_i, t_j]$, $v_2 = [t_p, t_q]$ are two time periods and $|x|$ denotes the *absolute value* of x , function *distance* is defined as

- $distance(v_1, v_2) = \begin{cases} d = 0 & | v_1 \text{ } cp \text{ } v_2 \\ d = \min(|t_i - t_q|, |t_j - t_p|) & | v_1 \text{ } disjoint \text{ } v_2 \end{cases}$

If $v_1 \text{ } cp \text{ } v_2$ evaluates to true, then function *intersect* [LM97] is defined as

- $v_1 \text{ } intersect \text{ } v_2 = [\max(\{t_i, t_p\}), \min(\{t_j, t_q\})]$

5.4 Data Structures and Operations for Time

In the sequel, only instants of DATE type are considered. For ease of presentation, the set of successive dates $\text{DATE} = \{d_0, d_1, \dots, d_{90}\}$ is considered. Periods of dates are denoted as $[d_i, d_j]$.

A relation is defined the known way, except that the underlying domain of one or more of its attributes can be of some time type, either INSTANT or

PERIOD. As an example, consider relation S in Figure 5.2, used to record the daily salary of each employee during some period of time. Hence, the first tuple shows that John's salary was 10k on each of the days in [d11, d30], as can also be seen in the diagram below the relation. Using such a relation, it is possible to associate pieces of data with the periods during which these data were valid. For the objectives of this thesis, it is said that a relation like this contains *valid time data* or *temporal data*. Relation U in Figure 5.3(a) also contains temporal data, since the salary of each employee is recorded separately for each distinct date. It is thus clear that, as apposed to a relation that lacks a time attribute, a *temporal relation* enables recording pieces of data that are valid at various distinct times.

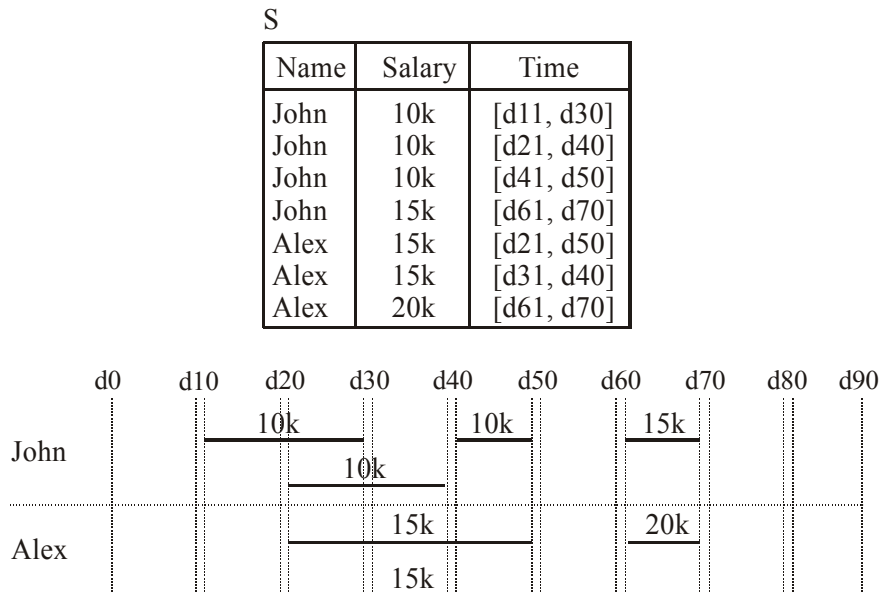


Figure 5.2: Example of a temporal relation.

One relation of particular interest is $\text{TIME_ALL}(T \mid \text{PERIOD})$, which contains a single tuple $t_{\text{TIME_ALL}} = ([t_0, t_{n-1}])$, where t_0 (t_{n-1}) is the smallest (greatest) element in INSTANT . Given that dates are considered in the examples, it is assumed that $t_{\text{TIME_ALL}} = [d_0, d_{90}]$.

Now the formalism of operations *Unfold* and *Fold* is given, when they are applied to relations, on a time attribute. Yet, one preliminary definition is given.

Definition 5.4: Let x be either an atomic element or a set. Then $\text{set}(x)$ is defined as follows:

$$set(x) \equiv \begin{cases} x & | \text{ x is a set} \\ \{x\} & | \text{ otherwise} \end{cases}$$

U		
Name	Salary	Time
John	10k	d11
...
John	10k	d50
John	15k	d61
...
John	15k	d70
Alex	15k	d21
...
Alex	15k	d50
Alex	20k	d61
...
Alex	20k	d70

F		
Name	Salary	Time
John	10k	[d11, d50]
John	15k	[d61, d70]
Alex	15k	[d21, d50]
Alex	20k	[d61, d70]

(b)

(a)

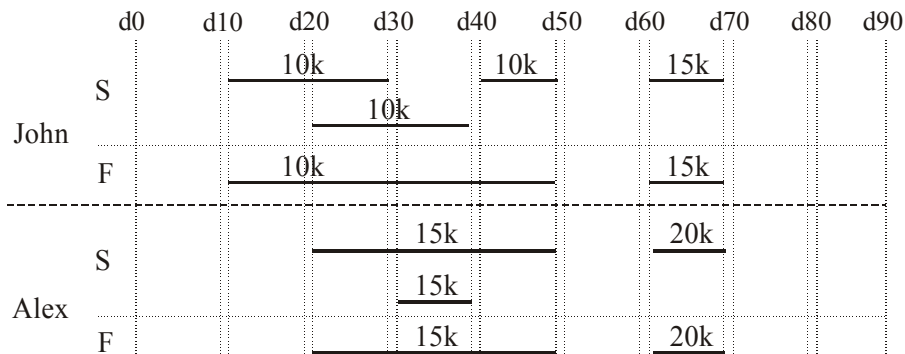


Figure 5.3: Result of operations *Unfold* and *Fold* on temporal data.

For an informal description of *Unfold*, let $R(\mathbf{A}, T)$ be the scheme of a relation. Let also (\mathbf{a}, t) be one of the tuples in R . When

$$Unfold[T](R)$$

is issued, (\mathbf{a}, t) yields in the result relation the set of tuples $\{(\mathbf{a}, t_i)\}$, where $\{t_i\}$ is the set of all time quanta in $set(t)$. The definition is as follows.

Definition 5.5: If R is a relation with scheme $R(\mathbf{A}, T)$ then relation

$$U = Unfold[T](R)$$

has scheme $U(\mathbf{A}, T \mid INSTANT)$ and extension

$$\{(\mathbf{a}, t_i) \mid t_i \in INSTANT \wedge set(t_i) \subseteq set(t) \wedge (\mathbf{a}, t) \in R\}.$$

Notice that there is only a minor difference between this definition and Definition 4.9, the use of *set*. This is due to the fact that temporal data modelling requires the use of time quanta, which are elements, whereas spatial data modelling requires the use of spatial quanta, which are sets.

As an example, if S is the relation in Figure 5.2 then

$$U = \text{Unfold}[\text{Time}](S)$$

is the relation in Figure 5.3(a), i.e., this operation yields the time instants at which a piece of data is valid.

One observation on relation S is that it contains duplicate data. For example, the fact that John's salary was 10k on date d25 has been recorded twice, once in the first and another in the second tuple (see also the relevant diagram). Contrary to this, one property of *Unfold*, a direct consequence of its definition, is that it eliminates data duplication. The difference is that the result relation has instants whereas the original relation may have periods.

By definition, *Unfold* can also be applied to a relation on an attribute of an instant type. It is then noted that $U = \text{Unfold}[\text{Time}](U)$, i.e. the result relation matches the original.

Now it is noticed in relation S (Figure 5.2) that John's salary was 10k on each of the dates in $[d21, d40]$ (second tuple) and also on each of the dates in $[d41, d50]$ (third tuple). Clearly, one would rather be tempted to see, in one tuple, that John's salary was 10k on each of the dates in the whole of $[d21, d50]$. Given also the data duplication in the first and second tuple that was discussed earlier, it is concluded that it is best for one to see that John's salary was 10k on each of the dates in $[d11, d50]$. Operation *Fold* achieves exactly this. For an informal description of this operation, let $R(\mathbf{A}, T)$ be the scheme of a relation. Let also $\{(\mathbf{a}, v_i)\}$ be a subset of R consisting of all the tuples with the same value for attributes \mathbf{A} . Then

$$\text{Fold}[T](R)$$

yields in the result relation the set of tuples $\{(\mathbf{a}, v_k)\}$, where all v_k are pair-wise non-adjacent periods with no points in common that consist of all the time instants in some of the above v_i . The operation is formalized as follows.

Definition 5.6: If $R(\mathbf{A}, T)$ is the scheme of a relation R then relation

$$F = \text{Fold}[T](R)$$

has scheme $F(\mathbf{A}, T \mid \text{PERIOD})$ and extension

$$\{(\mathbf{a}, v = \bigcup_{i=1}^n \text{set}(v_{r_i})) \mid (v \text{ is connected}) \wedge \\ ((\mathbf{a}, v_{r_i}) \in R, i=1, 2, \dots, n) \wedge \\ ((\exists(\mathbf{a}, v_{r_k}) \in R, k \neq 1, 2, \dots, n)(\text{set}(v) \cup \text{set}(v_{r_k}) \text{ is connected}))\}.$$

(Recall that the definition of connected set of time instants has been given in Definition 5.3.) Notice again that there is only a minor difference between this definition and that of *Fold* of spatial data (Definition 4.10).

As an example of the functionality of *Fold*, if R matches either relation S in Figure 5.2 or U in Figure 5.3(a) then the result of

$$F = \text{Fold}[\text{Time}](R)$$

is that one shown in Figure 5.3(b). It is noticed that F does not have two tuples with identical Name and Salary values that are associated either with adjacent periods or periods that have common points. At the same time, it is easy to see in F how an employee's salary *evolved* with respect to time, something not such obvious in relation S in Figure 5.2 (The issue of *data evolution with respect to time* is addressed in Section 5.6).

5.5 Application of Operations to Temporal Data

Now it is shown that the operations defined for the management of spatial data do have practical interest when they are applied to temporal relations. This is of particular interest, if it is noticed that, except the quantum operations, all the others have been specially defined for the management of spatial data. Although relations with an attribute of a PERIOD type are considered in the examples provided, the conclusions are the same if an INSTANT type is considered instead.

5.5.1 Application of Quantum Operations to Temporal Data

Let S_1 and S_2 be the two union compatible relations in Figure 5.4. Then the result of

$$F = S_1 \text{ } QUnion[T] \text{ } S_2$$

$$QE = S_1 \text{ } QExcept[T] \text{ } S_2$$

$$QI = S_1 \text{ } QIntersect[T] \text{ } S_2$$

is shown, respectively, in Figure 5.3(b), Figure 5.5(a), Figure 5.5(b).

Generally, let R_1 and R_2 be two union-compatible relations with respective scheme $R_1(\mathbf{A}, T \mid \text{PERIOD})$ and $R_2(\mathbf{A}, T \mid \text{PERIOD})$. Let also

$$S_1 = \{(\mathbf{a}, v_i) \mid i = 1, 2, \dots, p\} \quad (S_2 = \{(\mathbf{a}, v_j) \mid j = 1, 2, \dots, q\})$$

be a subset of R_1 (R_2), consisting of all the tuples with the same value for attributes \mathbf{A} . For these S_1 and S_2 ,

$$- R_1 \text{ } QUnion[T] \text{ } R_2$$

$$- R_1 \text{ } QExcept[T] \text{ } R_2$$

$$- R_1 \text{ } QIntersect[T] \text{ } R_2$$

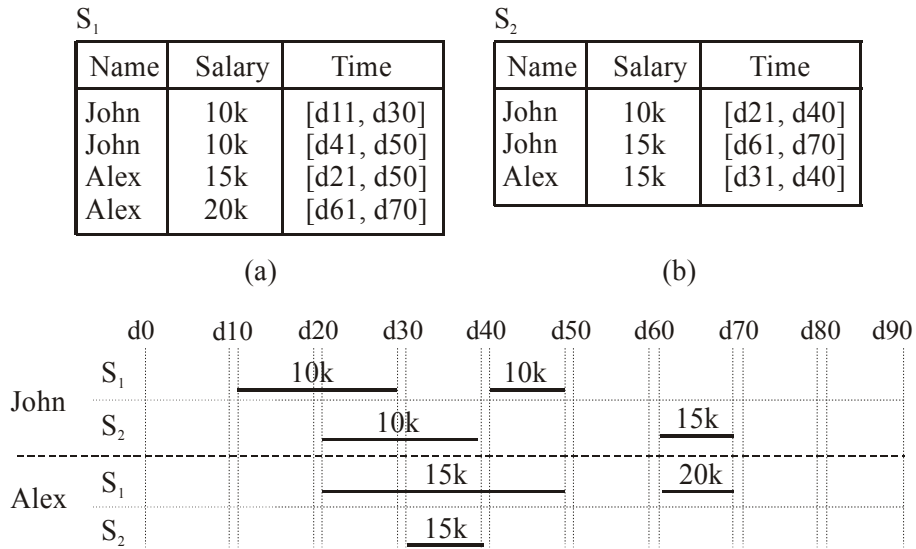


Figure 5.4: Temporal relations.

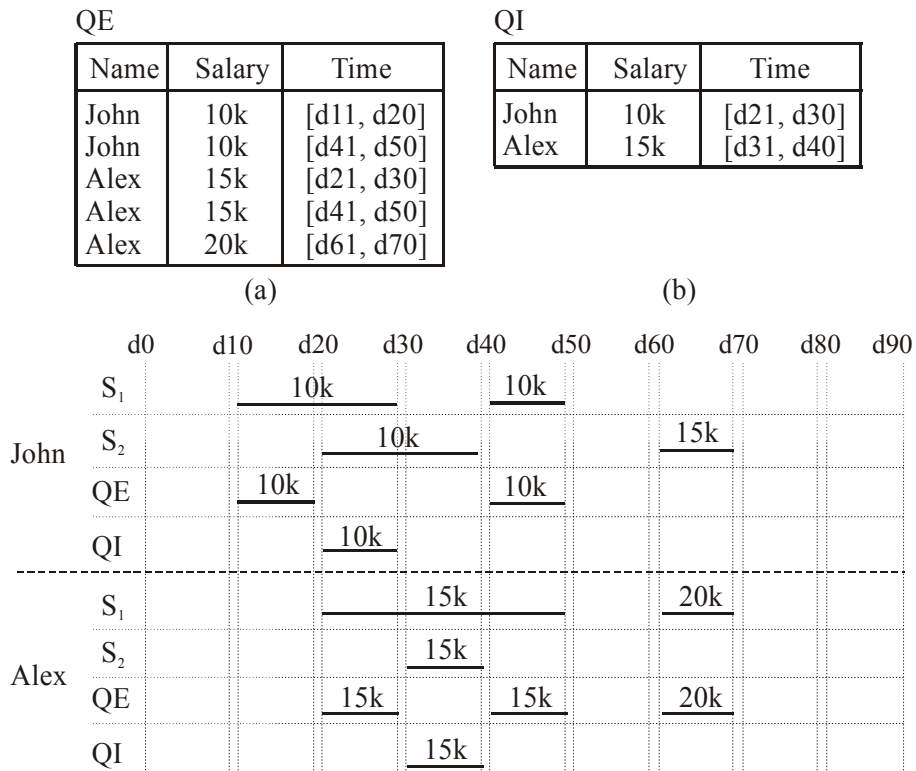


Figure 5.5: Result of operations *QExcept* and *QIntersect* on temporal data.

yields in the result relation the set of tuples $\{(a, v_k) \mid k = 1, 2, \dots, r\}$, where all v_k are pair-wise non-adjacent periods with no points in common that consist of all the time instants which are, respectively,

- in either some v_i or in some v_j
 - in some v_i but not in any v_j
 - in both some v_i and in some v_j ,
- $i = 1, 2, \dots, p, j = 1, 2, \dots, q$.

5.5.2 Application of Pair-Wise Operations to Temporal Data

Consider relations R and S in Figure 5.6(a) and Figure 5.6(b), respectively. Then the result of

- $WU = R \quad WUnion[T] \quad S$
- $WE = R \quad WExcept[T] \quad S$
- $WI = R \quad WIntersect[T] \quad S$

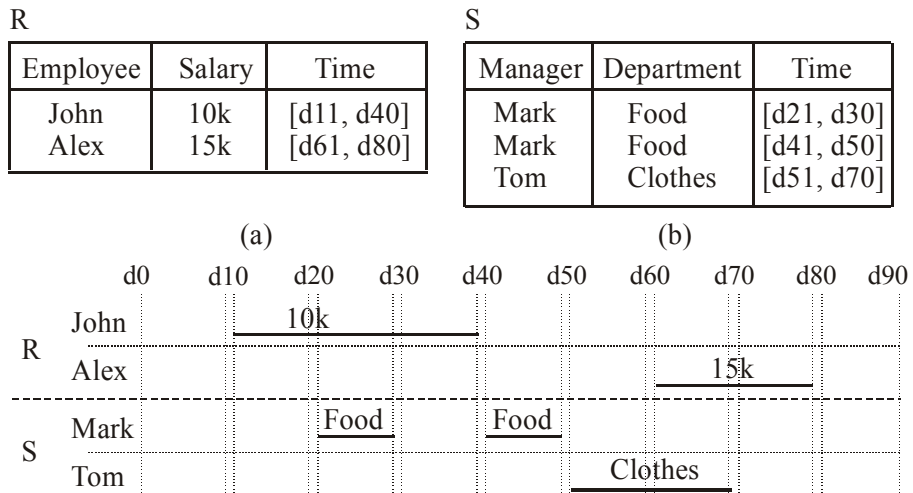


Figure 5.6: Temporal relations.

is shown in Figure 5.7. Generally, let R_1 and R_2 be two relations with respective scheme $R_1(A, T \mid PERIOD)$ and $R_2(B, T \mid PERIOD)$. Let also

$$S_1 = \{(a, v_i) \mid i = 1, 2, \dots, p\} \quad (S_2 = \{(b, v_j) \mid j = 1, 2, \dots, q\})$$

be a subset of R_1 (R_2), consisting of all the tuples with the same values for attributes $A(B)$. For these S_1 and S_2 ,

- $R_1 \quad WUnion[T] \quad R_2$
- $R_1 \quad WExcept[T] \quad R_2$
- $R_1 \quad WIntersect[T] \quad R_2$

WU

Employee	Salary	Manager	Department	Time
John	10k	Mark	Food	[d11, d50]
John	10k	Tom	Clothes	[d11, d40]
John	10k	Tom	Clothes	[d51, d70]
Alex	15k	Mark	Food	[d21, d30]
Alex	15k	Mark	Food	[d41, d50]
Alex	15k	Mark	Food	[d61, d80]
Alex	15k	Tom	Clothes	[d51, d80]

WE

Employee	Salary	Manager	Department	Time
John	10k	Mark	Food	[d11, d20]
John	10k	Mark	Food	[d31, d40]
John	10k	Tom	Clothes	[d11, d40]
Alex	15k	Mark	Food	[d61, d80]
Alex	15k	Tom	Clothes	[d71, d80]

WI

Employee	Salary	Manager	Department	Time
John	10k	Mark	Food	[d21, d30]
Alex	15k	Tom	Clothes	[d61, d70]

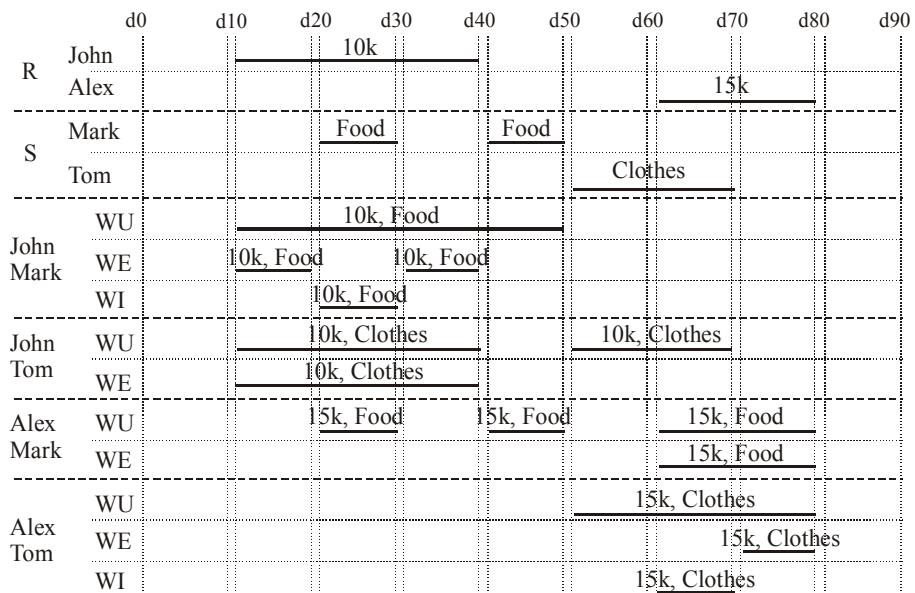


Figure 5.7: Result of pair-wise operations on temporal data.

yields in the result relation the set of tuples $\{(\mathbf{a}, \mathbf{b}, v_k) \mid k = 1, 2, \dots, r\}$, where all v_k are pair-wise non-adjacent periods with no points in common that consist of all the time instants which are, respectively,

- in either some v_i or in some v_j
- in some v_i but not in any v_j
- in both some v_i and in some v_j ,

$i = 1, 2, \dots, p, j = 1, 2, \dots, q$.

5.5.3 Application of Overlay Operations to Temporal Data

If R and S are the relations in Figure 5.6(a) and Figure 5.6(b), respectively, then the result of

$$FO = R \text{FOverlay}[\text{Time}] S$$

is shown in Figure 5.8. If the operation is

- *IOverlay* the result relation consists of the tuples (2) and (6) in FO.
- *LOverlay* the result relation consists of the tuples (1), (2), (3), (6), (7) in FO.
- *ROverlay* the result relation consists of the tuples (2), (4) and (5), (6) in FO.

Generally, let R_1 and R_2 be two relations with respective scheme $R_1(\mathbf{A}, T \mid \text{PERIOD})$ and $R_2(\mathbf{B}, T \mid \text{PERIOD})$. Let also

$$S_1 = \{(\mathbf{a}, v_i) \mid i = 1, 2, \dots, p\} \quad (S_2 = \{(\mathbf{b}, v_j) \mid j = 1, 2, \dots, q\})$$

be a subset of R_1 (R_2), consisting of all the tuples with the same values for attributes \mathbf{A} (\mathbf{B}). For these S_1 and S_2 :

- $IO = R_1 \text{IOverlay}[T] R_2$

consists of the set of tuples $\{(\mathbf{a}, \mathbf{b}, v_k) \mid k = 1, 2, \dots, r\}$, where all v_k are pair-wise non-adjacent periods with no points in common that consist of all the time instants which are both in some v_i and in some v_j .

- $LO = R_1 \text{LOverlay}[T] R_2$

consists of the set of tuples in IO and also all the tuples $\{(\mathbf{a}, \text{null}, v_k) \mid k = 1, 2, \dots, r\}$, where all v_k are pair-wise non-adjacent periods with no points in common that consist of all the time instants in some v_i but not in any v_j .

- $RO = R_1 \text{ROverlay}[T] R_2$

consists of the set of tuples in IO and also all the tuples $\{(\text{null}, \mathbf{b}, v_k) \mid k = 1, 2, \dots, r\}$, where all v_k are pair-wise non-adjacent periods with no points in common that consist of all the time instants in some v_j but not in any v_i .

- $FO = R_1 \text{FOverlay}[T] R_2$

consists of all the tuples in IO, LO and RO.

FO

Employee	Salary	Manager	Department	Time	
John	10k			[d11, d20]	(1)
John	10k	Mark	Food	[d21, d30]	(2)
John	10k			[d31, d40]	(3)
		Mark	Food	[d41, d50]	(4)
		Tom	Clothes	[d51, d60]	(5)
Alex	15k	Tom	Clothes	[d61, d70]	(6)
Alex	15k			[d71, d80]	(7)

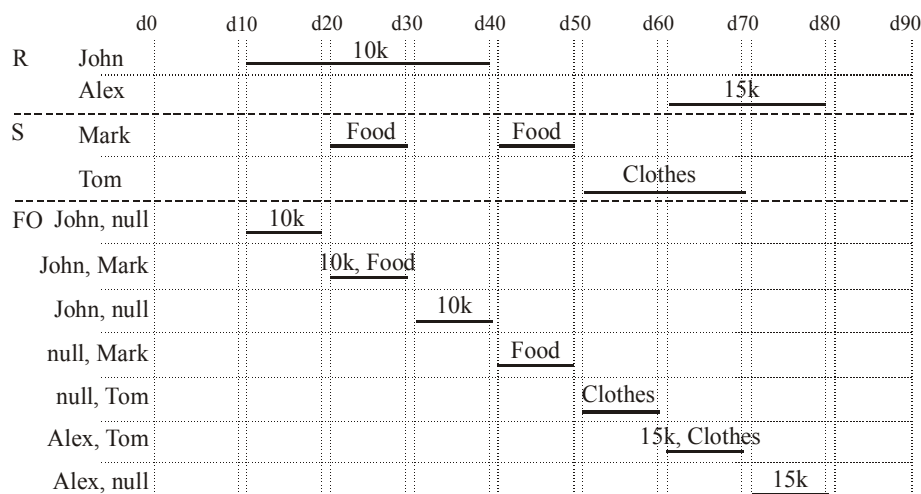


Figure 5.8: Result of overlay operations on temporal data.

5.5.4 Application of Other Operations to Temporal Data

If $t_{\text{TIME_ALL}} = [d0, d90]$ and S is the relation in Figure 5.2, then

$$\bar{C} = \text{Complementation}[T](S)$$

gives the relation in Figure 5.9. Generally, the complementation of a relation with scheme $R(\mathbf{A}, T)$ consists of tuples of the form (\mathbf{a}, v') , where v' is a greatest period during which \mathbf{a} is not valid, in the sense that none of the instants in v' are instants of some tuple $(\mathbf{a}, v) \in R$.

If S is the relation in Figure 5.10(a) then

$$B = \text{Buffer}[D, \text{Time}](S)$$

yields the relation in Figure 5.10(b). Generally, let $B = \text{Buffer}[\mathbf{A}, D, T](R)$ and let $(\mathbf{a}, d, [t_i, t_j])$ be a tuple obtained when R is normalized. This tuple of R yields, in the general case, a tuple $(\mathbf{a}, d, [t_{i-d}, t_{j+d}])$ (actually the period is $[t_{\max(0, i-d)}, t_{\min(90, i+d)}]$) in B , $[t_{i-d}, t_{j+d}] \subseteq t_{\text{TIME_ALL}}$, i.e., the period of validity of \mathbf{a} is extended by d instants to both the left and right.

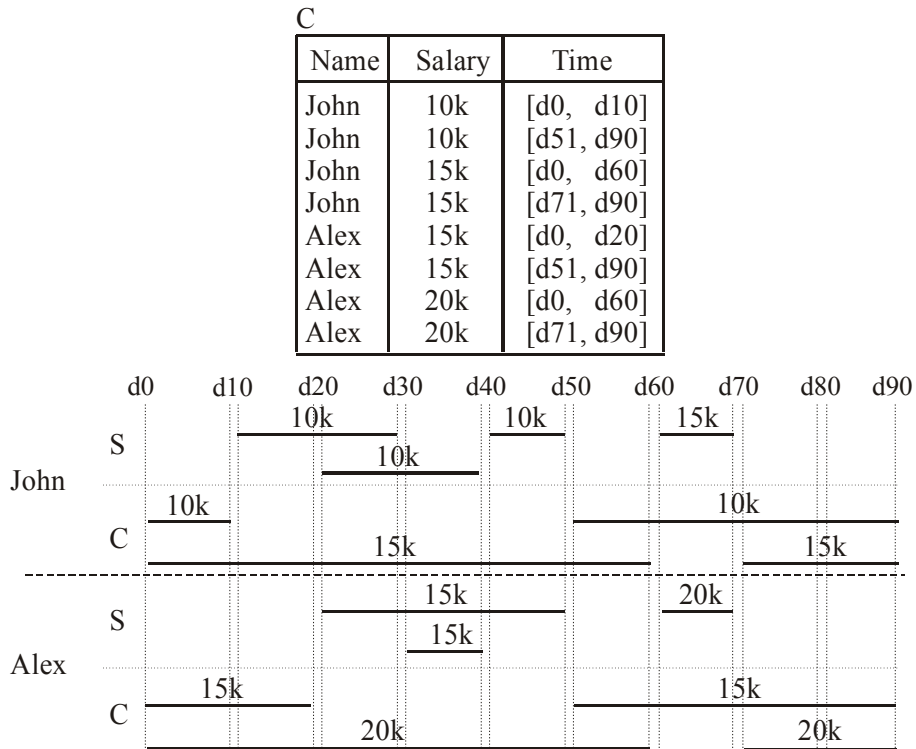


Figure 5.9: Result of *Complementation* on temporal data.

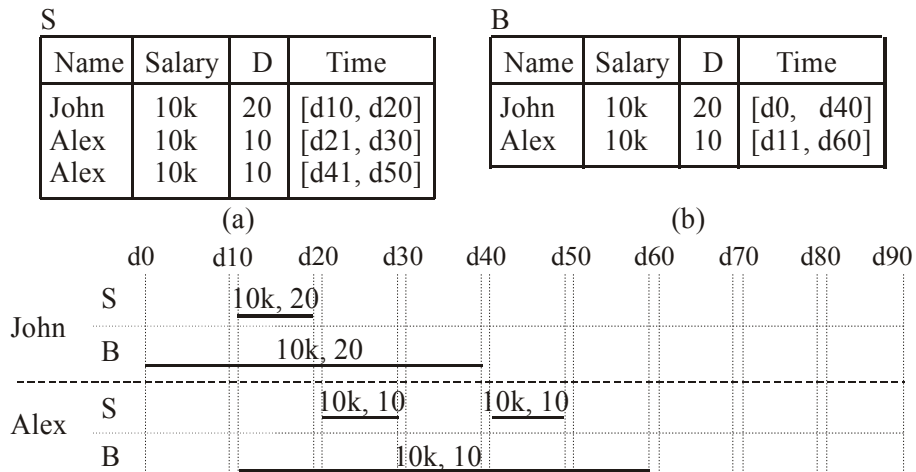


Figure 5.10: Result of *Buffer* on temporal data.

Regarding operation *Boundary*, it can easily be verified that when it is applied to a relation on a time attribute, it returns an empty relation. This is due to the

fact that the complementation of a period $[t_i, t_j]$ consists in general of two periods, $[t_0, t_{i-1}]$ and $[t_{j+1}, t_n]$ and neither of the two has common instants with $[t_i, t_j]$. This result matches fully the mathematical definition of boundary, when it is applied to a discrete set, and this is really a desirable result for the objectives of this thesis. Hence, it makes sense to apply *Boundary* to a relation on a time attribute, except that this does not really have some practical interest.

By its definition, a period does not have holes. Hence, it can also be verified that when operation *Envelope* is applied to a relation on a time attribute, it functions as operation *Fold*. Again, therefore, it makes sense to apply *Envelope* to a relation on a time attribute, except that this does not really have practical interest.

5.6 Evolution of Data with Respect to Time

The major data-modelling problem addressed in temporal databases is the definition of operations that yield relations, which show the *evolution of data with respect to time*. In general, the conventional relational algebra operations lack this capability. As an example, if TP is the temporal relation in Figure 5.11(a) then *Project*[A, T](TP) yields a relation R(A, T) with tuples

- (1, [d1, d2]),
- (1, [d2, d4]),
- (2, [d3, d5]),

Although R is temporal, it does not show precisely how its contents evolved, as opposed to relation TS in Figure 5.11(b). The same is also true if *Union* is applied to S_1 and S_2 , in Figure 5.4. Given that data evolution is an issue addressed in the next section, formalism is provided next.

Definition 5.7: Let P(A), Q(B) and R(C) be non-temporal relation schemes. Let also *Unary* (*Binary*) be an operation such that

$$S = \text{Unary}(P) \quad (S = Q \text{ Binary } R).$$

Consider the set $T = \{t_1, t_2, \dots, t_r\}$ of not necessarily successive instants. Assume also that the contents of P (Q and R)

- at time t_1 is $p_{t_1} (q_{t_1}, r_{t_1})$,
- at time t_2 is $p_{t_2} (q_{t_2}, r_{t_2})$,
- ...
- at time t_r is $p_{t_r} (q_{t_r}, r_{t_r})$.

Let the application of *Unary* (*Binary*) to P (Q and R) at these times return in S, respectively, $s_{t_1}, s_{t_2}, \dots, s_{t_r}$. Finally, let TP(A, T), TQ(B, T) and TR(C, T)

be temporal relations that contain exactly all the above data in P, Q and R, respectively, associated with the relevant instants. Finally, let

$$TS = TUnary(TP) \quad (TS = TQ \ TBinary \ TR).$$

If *TUnary* (*TBinary*) functions in such a way so that from the contents of TS it is always possible to identify exactly that the contents of S at time t_i was s_{t_i} for all $t_i \in T$, it is said that *TUnary* (*TBinary*) is the *evolution of Unary* (*Binary*) with respect to time.

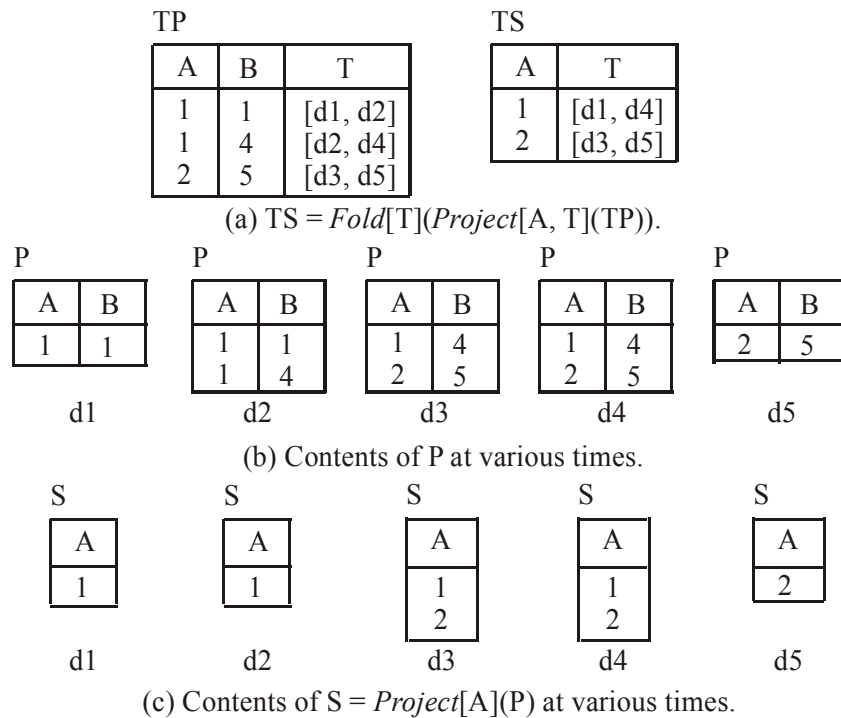


Figure 5.11: $Fold[Time] \circ Project[A, Time]$, is the evolution of $Project[A]$ with respect to time.

Example 5.1: Figure 5.11(b) shows that the contents of P(A, B) at each of the times d1 – d5 (i.e. at time d1 P contained the tuple (1, 1) or, equivalently, (1, 1) was valid at time d1 etc.). Figure 5.11(c) shows the result obtained in $S = Project[A](P)$ at each of these times. On the other hand, Figure 5.11(a) shows TP, a temporal relation, which contains all the data ever recorded in P, associated with the time during which that data was valid. Finally, TS, again in Figure 5.11(a), is another temporal relation that contains all the data ever recorded in S, associated again with the respective time. It is then noticed that

$$TS = Fold[T](Project[A, T](TP)).$$

Given that this property is always satisfied, it follows that $Fold[Time] \circ Project[A, Time]$ is the evolution of $Project[A]$ with respect to time, where $Operation_1 \circ Operation_2$ denotes composition of operations. In a similar manner, it can easily be verified that $QUnion[Time]$, $QExcept[Time]$ and $QIntersect[Time]$ is the evolution of $Union$, $Except$ and $Intersect$, respectively.

This issue of evolution is discussed separately in the next section, where the operations are applied to relations that contain spatio-temporal data.

5.7 Application to Spatio-temporal Data

As has been pointed out in Section 5.4, one or more attributes of a relation can be of some spatial or time type. For ease of discussion, a relation with at least one spatial and at least one time attribute is called *spatio-temporal*. Relation H in Figure 5.12, is such a relation, and it is used to record the *evolution* of a

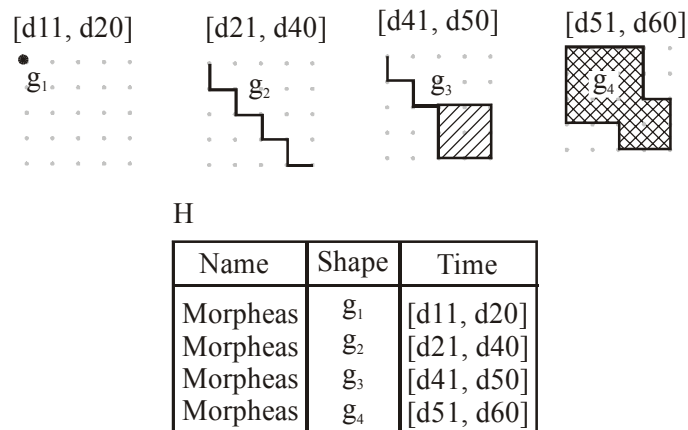
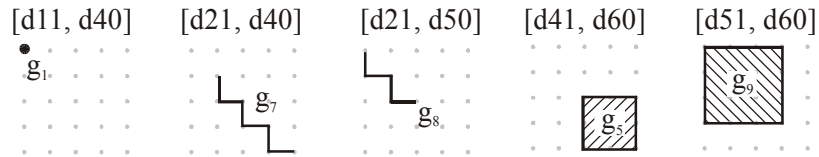


Figure 5.12: Spatio-temporal relation normalised on space and time.

hydrological event, called *Morpheas*. The geometric representation of the data recorded in attribute Shape is also shown in the same figure. Hence, the geometric representation of the object recorded in the first tuple shows that during [d11, d20] Morpheas was a spring (point). Then, during [d21, d40], it was a river (line). Next, during [d41, d50] it was a river pouring into a lake (hybrid surface) and, finally, during [d51, d60], it was a lake (pure surface). Two more spatio-temporal relations are shown in Figure 5.13. In all these relations it is noted that a 2-d spatial object is recorded in attribute Shape and time, recorded in the last attribute, represents a value of a 1-d space (Section

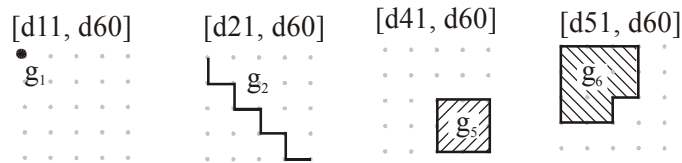
5.2). It then follows that a pair of (G, T) attributes of a spatio-temporal relation represents a 3-d space.



H_1

Name	Shape	Time
Morpheas	g_1	[d11, d40]
Morpheas	g_7	[d21, d40]
Morpheas	g_8	[d21, d50]
Morpheas	g_5	[d41, d60]
Morpheas	g_9	[d51, d60]

(a) Non-normalised spatio-temporal relation.



H_2

Name	Shape	Time
Morpheas	g_1	[d11, d60]
Morpheas	g_2	[d21, d60]
Morpheas	g_5	[d41, d60]
Morpheas	g_6	[d51, d60]

(b) Spatio-temporal relation normalized on time and space.

Figure 5.13: Spatio-temporal relations.

In the remainder of this chapter it is shown that the operations defined for the management of either spatial or temporal data suffice for the management of spatio-temporal data, i.e. there is no need to define new operations. Note however that these operations have now to be applied to pairs of (G, T) attributes, which form a 3-d space. Hence, the interpretation of the result relation must also be interpreted in this space. This is illustrated by the following example.

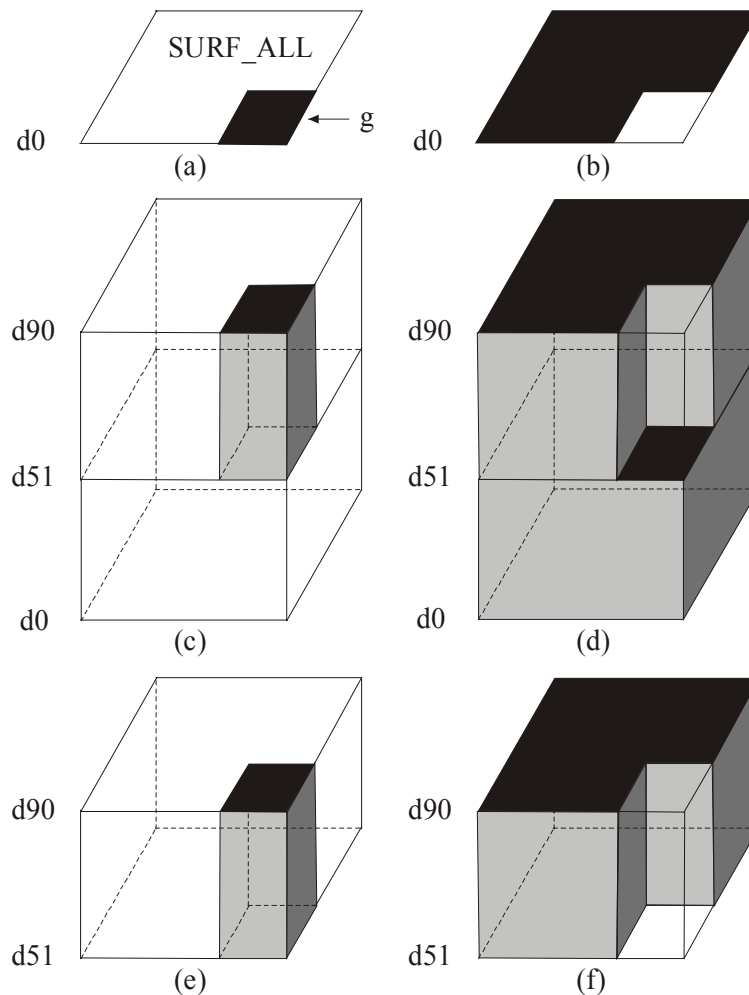


Figure 5.14: Complementation in a 3-d space.

Example 5.2: Consider a spatial relation $S(G)$, with one tuple (g) . Figure 5.14(a) shows the geometric representation of g , which is a subset of $SURF_ALL$. If *Complementation* is applied to S , then the geometric representation of the tuple in the result relation is the one shown in Figure 5.14(b). Since this result was obtained by the application of *Complementation* to a spatial, non-temporal relation, let, for ease of discussion, be said that this is a *spatial relational algebra operation*.

Now let a spatio-temporal relation be considered. Assume therefore that the contents of $SURF_ALL$ remains the same at all the times and recall that $TIME_ALL$ has a single tuple $[d0, d90]$. Consider also the spatiotemporal relation $ST(G, T)$ and let it contain one tuple, $(g, [d51, d90])$. This tuple

shows that the shape of object g remained the same during $[d51, d90]$. Hence, the geometric interpretation of the tuple in the 3-d space (SURF_ALL, TIME_ALL) is that one shown in Figure 5.14(c). It is then obvious that the complementation of the tuple in this space is the one shown in Figure 5.14(d). Indeed, the application of *Complementation* to relation ST (discussed later in Subsection 5.7.5) yields exactly this result. For ease of discussion, let it be said that this is a *spatio-temporal relational algebra operation*. In general, all the operations defined in Chapter 4 behave this way, when they are applied to relations, on pairs of (G, T) attributes.

In addition however to the above stated functionality of a *spatio-temporal relational algebra operation*, it also has practical interest to obtain the evolution of the *spatial relational algebra operation* with respect to time. The result of the evolution of operation *Complementation* is illustrated in Figures 5.14(e) and 5.14(f). The first of them shows how object g evolved with respect to time and the second shows how its spatial complementation evolved. Due to this observation, in conjunction with the discussion in the previous section, the spatio-temporal functionality of the operations is illustrated in the remainder of this section, but operations are also defined that represent the evolution of the respective spatial operations. For ease of discussion, a relational algebra operation is called *spatial (spatio-temporal)* if it is applied to a relation on a spatial attribute (a pair of a spatial and of a time attribute). From the illustration of the functionality of *Complementation* on a temporal and on a spatio-temporal relation, the validity of the next proposition is obvious.

Proposition 5.1: If *SOperation* is a spatial operation and *TSEOperation* is the relevant spatio-temporal then, in general, *TSEOperation* is not the evolution of *SOperation* with respect to time.

5.7.1 Spatio-temporal Unfold and Fold Operations

Definition 5.8: If A_1, A_2, \dots, A_n is a subset of the attributes of a relation R , the following are defined [LPS95]:

$$\begin{aligned} \text{Unfold}[A_1, A_2, \dots, A_n](R) &\equiv \text{Unfold}[A_n](\dots \text{Unfold}[A_2](\text{Unfold}[A_1](R))\dots) \\ \text{Fold}[A_1, A_2, \dots, A_n](R) &\equiv \text{Fold}[A_n](\dots \text{Fold}[A_2](\text{Fold}[A_1](R))\dots) \\ \text{Normalise}[A_1, A_2, \dots, A_n](R) &\equiv \text{Fold}[A_1, A_2, \dots, A_n](\text{Unfold}[A_1, A_2, \dots, A_n](R)) \end{aligned}$$

Since *Normalise* is defined in terms of *Fold*, it follows that

$$\text{Normalise}[A_1, A_2] \neq \text{Normalise}[A_2, A_1].$$

Some properties of these operations can be found in Appendix A, whereas a further set, associated with an efficient implementation, can be found in [LPS95, DDL03].

Name	Shape	Time
Morpheas	g ₁	d11
...
Morpheas	g ₁	d40
Morpheas	g ₇	d21
...
Morpheas	g ₇	d40
Morpheas	g ₈	d21
...
Morpheas	g ₈	d50
Morpheas	g ₅	d41
...
Morpheas	g ₅	d60
Morpheas	g ₉	d51
...
Morpheas	g ₉	d60

Name	Shape	Time
...
Morpheas	q ₁ g ₈	d50
Morpheas	q ₂ g ₈	d50
Morpheas	q ₃ g ₈	d50
.	.	.
.	.	.
.	.	.
Morpheas	q _n g ₈	d50
...

* {q₁g₈, q₂g₈, q₃g₈, ..., q_ng₈}
denotes the set of quanta of
space contained in g₈.

(a) *Unfold*[Time](H)
(b) *Unfold*[Shape](UH₁)

Figure 5.15: Examples of operation *Unfold*.

Although in [LM97] it has been argued that these operations can be applied to a relation on some attribute of any data type, for the objectives of this thesis only spatial and time attributes are considered. To illustrate now the functionality of these operations, consider relation H₁ in Figure 5.13(a). It is a spatio-temporal relation and, by a careful examination, it can be seen that it contains duplicate data. As an example, consider its fourth tuple and the geometric interpretation of object g₅. It is then noted that during the period [d51, d60] (which is a sub-period of the lifespan of g₅), object g₅ contains the top left quantum surface. From the geometric interpretation of object g₉, it is noted that during [d51, d60] this quantum surface is also contained in object g₉. Hence, a piece of surface has been recorded redundantly in two distinct tuples during [d51, d60]. A similar observation applies to objects g₇, g₈ for the period [d21, d40]. This data duplication can be eliminated by the use of *Unfold* (see Section 5.4) by applying the sequence of operations

$$UH_1 = \text{Unfold}[\text{Time}](H_1)$$

$$UH_2 = \text{Unfold}[\text{Shape}](UH_1)$$

UH₁ and UH₂ are depicted in Figure 5.15. Alternatively, UH₂ can be directly obtained from H₁ by applying

$$UH_2 = \text{Unfold}[\text{Shape}, \text{Time}](H_1).$$

Since UH₂ does not contain duplicate data, the same is also true for each of

$$H = Fold[Shape, Time](UH_2),$$

$$H_2 = Fold[Time, Shape](UH_2),$$

shown respectively in Figures 5.12 and 5.13(b). Moreover, a comparison of the contents of relations H and H₂ shows that the application of *Fold* first on a spatial and then on a time attribute yields a relation that enables identifying the evolution of the shape of spatial data with respect to time. Equivalently, H can be obtained directly from H₁ by applying

$$H = Normalise[Shape, Time](H_1).$$

Finally, H and H₂ can be obtained from each other by applying

$$H = Normalize[Shape, Time](H_2),$$

$$H_2 = Normalize[Time, Shape](H).$$

Relation \	[d11, d20]	[d21, d30]	[d31, d40]	[d41, d50]	[d51, d60]
LAND_USE F ≡ Forest C ≡ Cultivation I ≡ Industrial					
P_OWNER P ≡ Peter's parcel J ≡ John's parcel S ≡ Susan's parcel					

LAND_USE

Use	Shape	Time
Cultivation	g ₁	[d11, d20]
Forest	g ₂	[d11, d30]
Forest	g ₄	[d31, d50]
Industrial	g ₁	[d21, d30]
Industrial	g ₃	[d31, d50]

(a)

P_OWNER

Owner	Pid	Shape	Time
John	P2	g ₅	[d21, d40]
Peter	P1	g ₆	[d21, d30]
Peter	P1	g ₈	[d31, d40]
Peter	P3	g ₇	[d41, d60]
Susan	P3	g ₇	[d31, d40]
Susan	P1	g ₈	[d41, d60]
Susan	P2	g ₅	[d41, d60]

(b)

Figure 5.16: Spatio-temporal relations and their geometric representation.

To illustrate the functionality of the remainder operations on spaito-temporal relations, consider two relations. The first of them is LAND_USE in Figure 5.16(a), and it is used to record the evolution of the shape of land uses with respect to time. The geometric representation of this evolution is shown at the top of Figure 5.16. Consider also relation

$$OWNERSHIP(Owner, Pid, Time | PERIOD)$$

used to record the evolution of the ownership of land parcels with respect to time. Let also

LAND_PARCEL(Pid, Shape | PSURFACE, Time | PERIOD)

be a second relation used to record the evolution of the shape of land parcels with respect to time. It is then easy to apply relational algebra operations to OWNERSHIP and LAND_PARCEL and obtain another relation

P_OWNER(Owner, Pid, Shape | PSURFACE, Time | PERIOD),

where, for each time period, both the shape and the owner of a land parcel is shown. Let the extension of this relation be the one appearing in Figure 5.16(b). The geometric representation of this extension is also shown in Figure 5.16. Then the examples given in the remainder sections are primarily based on relations LAND_USE and P_OWNER.

5.7.2 Spatio-temporal Quantum Operations

Let I (Figure 5.17) be the relation obtained by the following sequence of operations.

$TR_1 = \text{Select}[\text{Use} = \text{'Industrial'}](\text{LAND_USE})$
 $I = \text{Project}[\text{Shape}, \text{Time}](TR_1)$

Its geometric representation is also shown in the same Figure 5.16. Notice that two plots of tuple $(g_3, [d31, 50])$ are shown, one for period $[d31, d40]$ and a second for $[d41, d50]$. This has been intentional, only to ease the demonstration of the functionality of the operations that follow. This principle is also adopted in the sequel.

Let also S (Figure 5.17) be the relation obtained by the following sequence of operations.

$TR_1 = \text{Select}[\text{Owner} = \text{'Susan'}](\text{P_OWNER})$
 $TR_2 = \text{Project}[\text{Shape}, \text{Time}](TR_1)$
 $S = \text{Normalize}[\text{Shape}, \text{Time}](TR_2)$

Its geometric representation is also shown in Figure 5.17. Then the result relations of operations

$QU = I \text{ } Q\text{Union}[\text{Shape}, \text{Time}] \text{ } S,$
 $QE = I \text{ } Q\text{Except}[\text{Shape}, \text{Time}] \text{ } S,$
 $QI = I \text{ } Q\text{Intersect}[\text{Shape}, \text{Time}] \text{ } S,$

and their geometric interpretation as well, are shown in Figure 5.17. The examples illustrate the following.

Proposition 5.2: $Q\text{Union}[G, T]$ ($Q\text{Except}[G, T]$, $Q\text{Intersect}[G, T]$) is the evolution with respect to time of $Q\text{Union}[G]$ ($Q\text{Except}[G]$, $Q\text{Intersect}[G]$).

The proof is omitted, as simple.

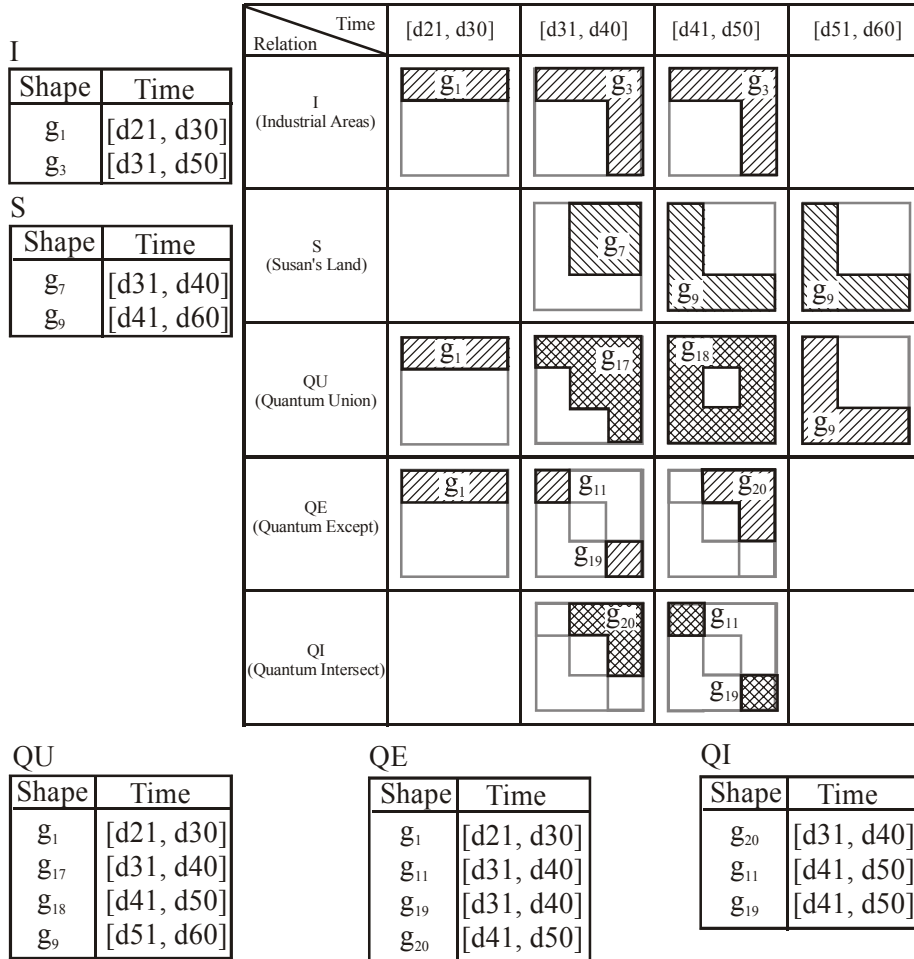


Figure 5.17: Result of spatio-temporal quantum operations.

5.7.3 Spatio-temporal Pair-Wise Operations

Consider again the relations in Figure 5.16 and the operations

$$\begin{aligned}
 TR_1 &= \text{Select}[\text{Use} = \text{'Industrial'}](\text{LAND_USE}) \\
 I &= \text{Project}[\text{Use}, \text{Shape}, \text{Time}](TR_1) \\
 TR_2 &= \text{Select}[\text{Owner} = \text{'Susan'} \text{ or } \text{Owner} = \text{'Peter'}](\text{P_OWNER}) \\
 TR_3 &= \text{Project}[\text{Owner}, \text{Shape}, \text{Time}](TR_2) \\
 S &= \text{Normalize}[\text{Shape}, \text{Time}](TR_3)
 \end{aligned}$$

The extension of I, S, and their geometric representation as well, is shown in Figure 5.18.

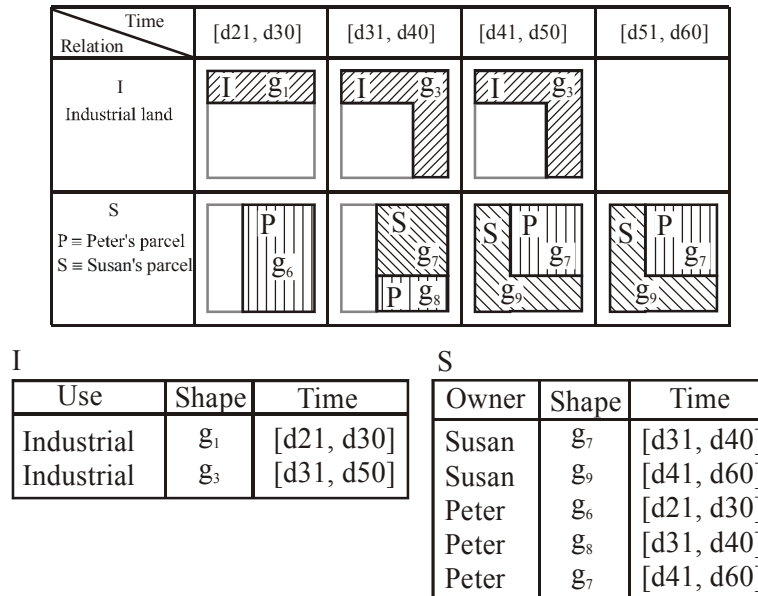


Figure 5.18: Input relations and geometric representation.

Then the result of

$$\begin{aligned} SWU &= I \text{ } WUnion[\text{Shape}, \text{Time}] \text{ } S, \\ SWE &= I \text{ } WExcept[\text{Shape}, \text{Time}] \text{ } S, \end{aligned}$$

is shown, respectively, in Figures 5.19, 5.20. As can be seen, $WUnion[G, T]$ and $WExcept[G, T]$ are not, respectively, the evolution of $WUnion[G]$ and $WExcept[G]$ with respect to time (Proposition 5.1). However, it is easy to see that this evolution can be implemented by the following sequence of relational algebra operations.

1. $TR_1 = Unfold[\text{Time}](I)$
2. $TR_2 = Unfold[\text{Time}](S)$
3. $TR_3 = TR_1(\text{Time}1 \leftarrow \text{Time}) \text{ } WUnion[G] \text{ } TR_2(\text{Time}2 \leftarrow \text{Time})$
4. $TR_4 = Select[\text{Time}1 = \text{Time}2](TR_3)$
5. $TR_5 = Project[\text{Use}, \text{Owner}, \text{Shape}, \text{Time}](TR_3(\text{Time} \leftarrow \text{Time}1))$
6. $W = Fold[\text{Time}](TR_5)$

To illustrate this functionality, an example relation W , and its geometric representation as well, is given in Figure 5.19. However, the definition of a relational algebra operation, in terms of the above sequence of operations, is avoided because a relevant syntax is provided in the SQL extension defined in the next chapter.





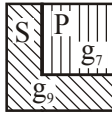
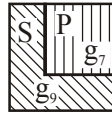

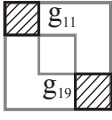
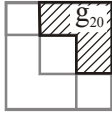

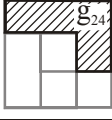
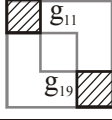
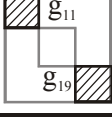
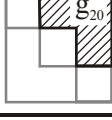
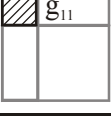
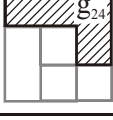
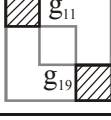
Time Relation		[d21, d30]	[d31, d40]	[d41, d50]	[d51, d60]
S P ≡ Peter's parcel S ≡ Susan's parcel					
(Spatio-temporal)	SWU				
	Industrial Peter				
(Evolution w.r.t.)	W				
	Industrial Peter				

SWU			
Use	Owner	Shape	Time
Industrial	Susan	g_1	[d21, d30]
Industrial	Susan	g_{17}	[d31, d40]
Industrial	Susan	g_{18}	[d41, d50]
Industrial	Susan	g_9	[d51, d60]
Industrial	Peter	g_{22}	[d21, d30]
Industrial	Peter	g_{23}	[d31, d40]
Industrial	Peter	g_{17}	[d41, d50]
Industrial	Peter	g_7	[d51, d60]

W			
Use	Owner	Shape	Time
Industrial	Susan	g_{17}	[d31, d40]
Industrial	Susan	g_{18}	[d41, d50]
Industrial	Peter	g_{22}	[d21, d30]
Industrial	Peter	g_{23}	[d31, d40]
Industrial	Peter	g_{17}	[d41, d50]

Figure 5.19: Spatio-temporal $WUnion$ and evolution of $WUnion$ with respect to time.

Similarly, a sequence of relational algebra operations that can implement the evolution of $WExcept[G]$ with respect to time is the one obtained if step 3 in the previous sequence is replaced by

Time Relation		[d21, d30]	[d31, d40]	[d41, d50]	[d51, d60]
					
I Industrial land					
S P ≡ Peter's parcel S ≡ Susan's parcel					
		S	S	S	S
(Spatio-temporal) SWE	Industrial Susan				
	Industrial Peter				
	Industrial Susan				
	Industrial Peter				

SWE				W			
Use	Owner	Shape	Time	Use	Owner	Shape	Time
Industrial	Susan	g ₁	[d21, d30]	Industrial	Susan	g ₁₁	[d31, d40]
Industrial	Susan	g ₁₁	[d31, d40]	Industrial	Susan	g ₁₉	[d31, d40]
Industrial	Susan	g ₁₉	[d31, d40]	Industrial	Susan	g ₂₀	[d41, d50]
Industrial	Susan	g ₂₀	[d41, d50]	Industrial	Peter	g ₁₁	[d21, d30]
Industrial	Peter	g ₁₁	[d21, d30]	Industrial	Peter	g ₂₄	[d31, d40]
Industrial	Peter	g ₂₄	[d31, d40]	Industrial	Peter	g ₁₁	[d41, d50]
Industrial	Peter	g ₁₁	[d41, d50]	Industrial	Peter	g ₁₉	[d41, d50]
Industrial	Peter	g ₁₉	[d41, d50]				

Figure 5.20: Spatio-temporal *WExcept* and evolution of *WExcept* with respect to time.

$$TR_3 = TR_1(\text{Time1} \leftarrow \text{Time}) \text{WExcept}[G] TR_2(\text{Time2} \leftarrow \text{Time})$$

As an example, this new sequence of operations yields relation *W* in Figure 5.20.

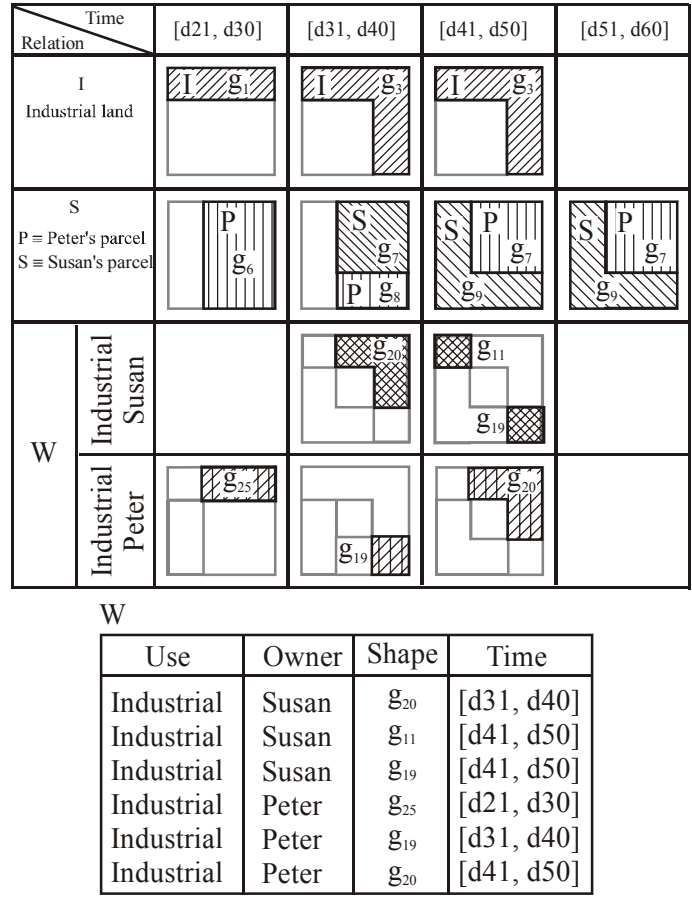


Figure 5.21: Example of spatio-temporal $WIntersect$.

Contrary to $WUnion[G, T]$ and $WExcept [G, T]$, the following proposition is satisfied.

Proposition 5.3: $WIntersect[G, T]$ is the evolution of $WIntersect[G]$ with respect to time.

The proof is again omitted, as simple. Note that, alternatively, the same result can be obtained if step 3 in the previous sequence of operations is replaced by

$$TR_3 = TR_1(\text{Time1} \leftarrow \text{Time}) WIntersect[G] TR_2(\text{Time2} \leftarrow \text{Time}).$$

An example relation W that illustrates this functionality is given in Figure 5.21.

5.7.4 Spatio-temporal Overlay Operations

Consider again the relations in Figure 5.16 and the sequence of operations

- $TR_1 = Select[Use = 'Industrial'](LAND_USE)$
- $I = Project[Use, Shape, Time](TR_1)$
- $TR_2 = Select[Owner = 'Susan'](P_OWNER)$
- $TR_3 = Project[Owner, Shape, Time](TR_2)$
- $S = Normalize[Shape, Time](TR_3)$

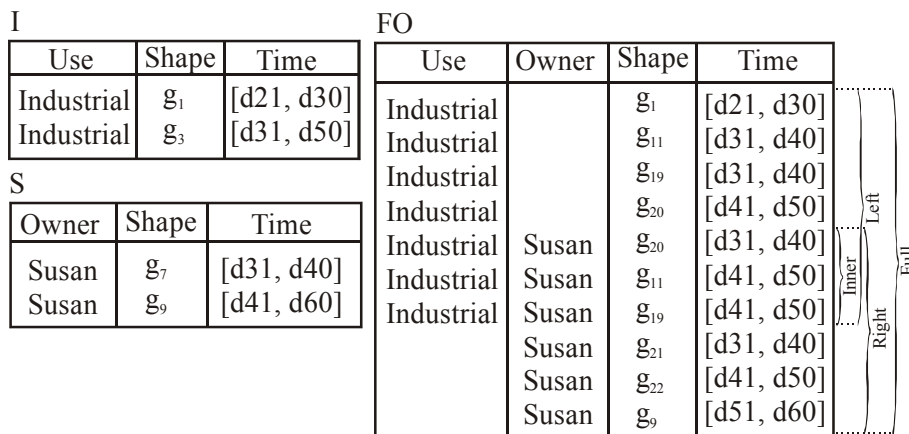
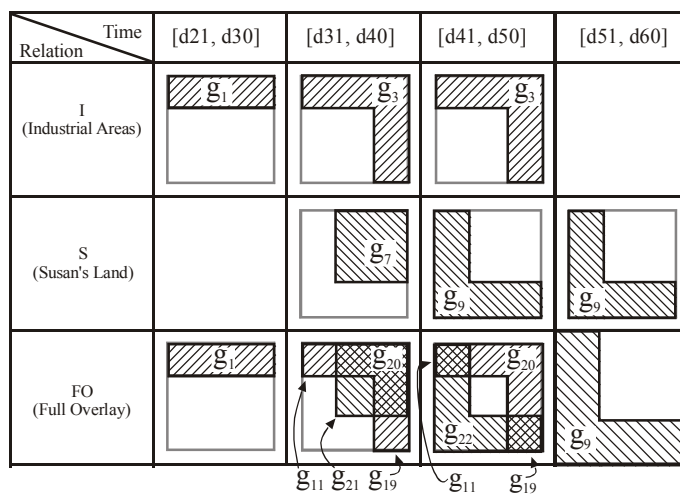


Figure 5.22: Result of spatio-temporal overlay operations.

The extension of both I, S and relevant geometric representation are shown in Figure 5.22. Then

$$FO = I \text{FOverlay}[G, T] S$$

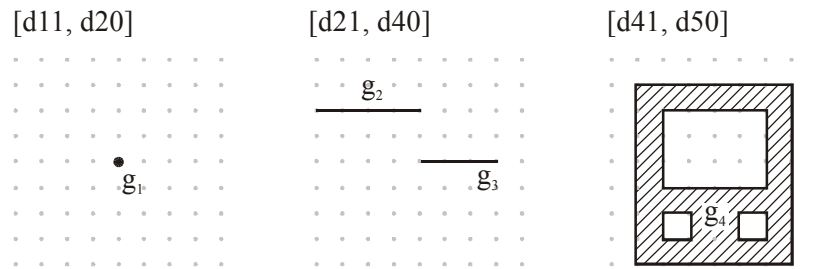
and its geometric representation is also shown in Figure 5.22. If, instead, $IOverlay (LOverlay, ROverlay)$ is applied then the result consists, respectively, of those tuples of FO that have been marked as Inner (Left, Right). It is easy to prove the following.

Proposition 5.4: $IOverlay[G, T] (LOverlay[G, T], ROverlay[G, T], FOverlay[G, T])$ is the evolution with respect to time of $IOverlay[G] (LOverlay[G], ROverlay[G], FOverlay[G])$.

The proof is again omitted, as simple

5.7.5 Other Interesting Spatio-temporal Operations

As has already been illustrated in Section 5.7, $Complementation[G, T]$ is *not* the evolution of $Complementation[G]$ with respect to time. However, the following general proposition for unary operations is satisfied.



H_3

Name	DG	DT	Shape	Time
Morpheas	4	10	g_1	[d11, d20]
Morpheas	1	20	g_2	[d21, d40]
Morpheas	1	20	g_3	[d21, d40]
Morpheas	1	10	g_4	[d41, d50]

Figure 5.23: Input spatio-temporal relation.

Proposition 5.5: If *Unary* is any of the *Complementation*, *Boundary*, *Envelope* and *Buffer* operations then

$$Fold[T] \circ Unary[G] \circ Unfold[T]$$

is the evolution of $Unary[G]$ with respect to time.

In addition, the following proposition is satisfied.

Proposition 5.6: $Boundary[G, T] (Envelope[G, T])$ is the evolution with respect to time of $Boundary[G] (Envelope[G])$.

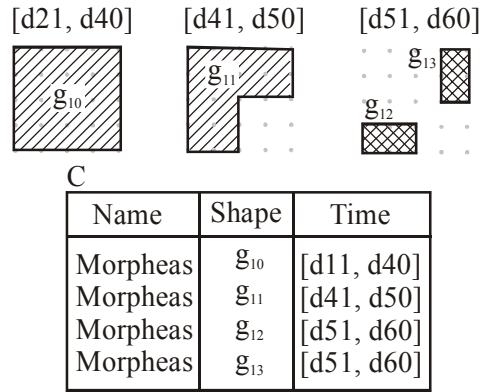


Figure 5.24: Evolution with respect to time of *Complementation*.

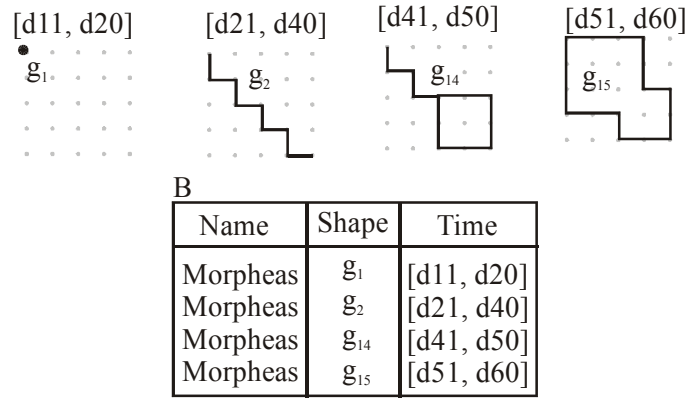


Figure 5.25: Result of operation *Boundary* on spatio-temporal data.

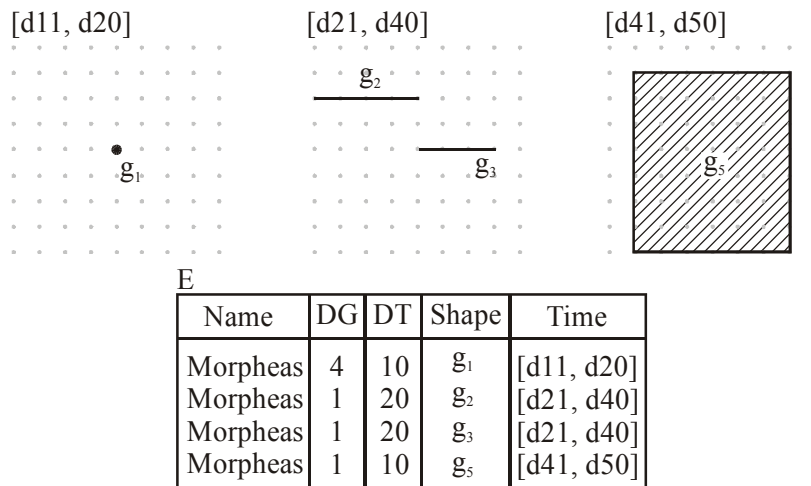
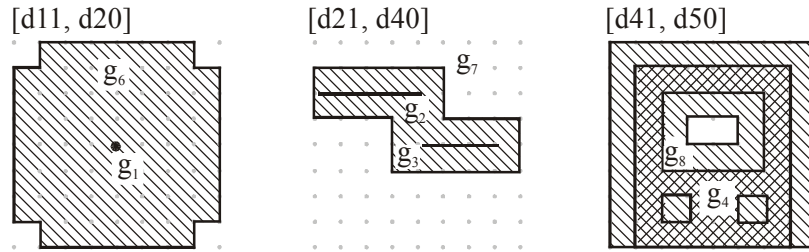


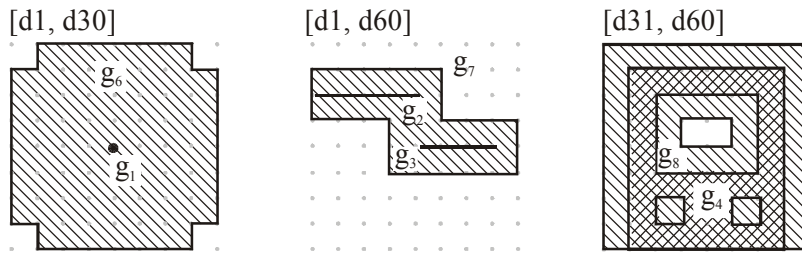
Figure 5.26: Result of operation *Envelope* on spatio-temporal data.



EB

Name	DG	DT	Shape	Time
Morpheas	4	10	g_6	[d11, d20]
Morpheas	1	20	g_7	[d21, d40]
Morpheas	1	10	g_8	[d41, d50]

(a) Evolution of *Buffer* with respect to time.



B

Name	DG	DT	Shape	Time
Morpheas	4	10	g_6	[d1, d30]
Morpheas	1	20	g_7	[d1, d60]
Morpheas	1	10	g_8	[d31, d60]

(b) Spatio-temporal *Buffer*.

Figure 5.27: Evolution of *Buffer* and spatio-temporal *Buffer*.

Again, the proofs are easy. To provide examples, consider relations H in Figure 5.12 and H_3 in Figure 5.23. Then, based on Proposition 5.5, the result of

$$C = Fold[Time](Complementation[Shape](Unfold[Time](H))),$$

shown in Figure 5.24, gives the evolution of the complementation of the data recorded in H with respect to time.

The result of

$$B = Boundary[Shape, Time](H),$$

shown in Figure 5.25, yields the evolution of the boundary of the data recorded in H with respect to time.

Similarly, the result of

$$E = \text{Envelope}[\text{Shape}, \text{Time}](H_3),$$

shown in Figure 5.26, yields the evolution of the envelope of the data recorded in H_3 with respect to time.

Finally, the result of

$$EB = \text{Fold}[\text{Time}](\text{Buffer}[\text{DG}, \text{Shape}](\text{Unfold}[\text{Time}](H_3))),$$

shown in Figure 5.27(a), yields the evolution of the buffer of the data recorded in H_3 with respect to time. For comparison reasons, the result of spatio-temporal *Buffer*

$$B = \text{Buffer}[\text{DG}, \text{DT}, \text{Shape}, \text{Time}](H_3)$$

is shown in Figure 5.27(b).

5.8 Conclusions

A formalism was provided for the management of temporal data. Based on the definition of time quanta, two generic time data types were defined, INSTANT and PERIOD. The definition of *Unfold* and *Fold* was also given, regarding their application to temporal relations, on a time attribute. Based on a developed formalism, a number of propositions were finally given, regarding the evolution of spatial data with respect to time. The characteristics of the model formalised in this chapter can be summarised as follows:

- The generic time types, INSTANT and PERIOD, match fully human perception.
- Temporal, spatial and spatio-temporal data are recorded in non-nested relations.
- No restriction is imposed on the number of spatial or time attributes of such relations.
- The definition of operations *Unfold* and *Fold*, when applied on attributes of a time type is a minor adjustment of the relevant definition of their application to a spatial attribute.
- No other operations had to be defined. Instead, it was shown that a unique set of operations enables the uniform management of spatial, temporal and spatio-temporal (and also *interval* [LM97]) data.

In conclusion, the spatio-temporal model defined in this thesis satisfies all the relevant properties specified in Section 2.8.

CHAPTER 6

SQL EXTENSION

6.1 Introduction

An SQL extension for the management of spatio-temporal data is defined, based on the relational algebra of the previous chapter. The syntax extends further IXSQL [LM97], defined for the management of interval data. It is assumed that the extension supports directly the data types for space and time that have been defined in Sections 3.3 and 5.2, respectively, as well as the relevant predicates and functions (some more can be found in Appendix B and in [LM97]). Note that only the primitives of the extension are presented in this chapter, but the full extension details are included in Appendix B. The extension to the SQL INSERT, DELETE and UPDATE manipulation operations is not considered here, since it matches the one given in [LM97]. The additional characteristics of SQL are given in **bold**. The remainder of this chapter is outlined as follows. In Section 6.2, the syntax of SQL:1999 <query specification> is extended by two new clauses, that enable the application of relational algebra operations *Unfold*, *Fold* and *Normalise*. An already proposed extension [LD96] that enables the formulation of queries that return data, which evolves with respect to time, is reviewed in Section 6.3. Section 6.4 presents the extension of the SQL:1999 <non-join query expression> that supports the application of the quantum and pair-wise operations of relational algebra. In Section 6.5 the SQL:1999 <joined table> is extended in a way that enables the incorporation of the various types of the overlay operations. A new type of <query expression>, namely <*unary query expression*>, is defined in Section 6.6. It enables the incorporation of the unary operations *Complementation*, *Boundary*, *Envelope* and *Buffer*. Syntactic rules for the proposed extension are given, wherever necessary. Finally, conclusions are drawn in the last section.

6.2 Query Specification

Two new optional clauses, <reformat clause> and <normalise clause>, have been added to the syntax of the SQL:1999 query specification, which is thus extended as follows [LM97]:

SELECT [<set quantifier>]	<select list>	(1)
FROM	<table ref list>	(2)
[WHERE	<search condition>]	(3)
[GROUP BY	<grouping column ref list>]	(4)
[HAVING	<search condition>]	(5)
[<reformat clause>]		(6)
[<normalise clause>]		(7)
[ORDER BY	<sort spec list>]	(8)

The BNF syntax of the new constructs is as follows:

```

<reformat clause> ::=
    REFORMAT AS <reformat item>

<reformat item> ::=
    FOLD [ALL] <reformat column list> [<reformat item>]
  | UNFOLD [ALL] <reformat column list> [<reformat item>]

<normalise clause> ::=
    NORMALISE ON [ALL] <reformat column list>

<reformat column list> ::=
    <reformat column> [{, <reformat column>}...]

<reformat column> ::=
    <column reference>| <unsigned integer>
  
```

The following syntactic rule applies:

Rule 6.1: The <reformat column list> must be a sub-list of the attributes that appear in <select list>.

(Note that in SQL:1999 line (8) is not actually part of <query specification>, but it has been included here for simplicity reasons.) The semantics and functionality of this extension is described below.

Reformat Clause

Lines (1)-(5) are executed as in SQL:1999 and next lines (6)-(8) are executed in this order. The <reformat clause> enables the introduction of a sequence of *Unfold* and *Fold* operations that are applied to the result produced by the execution of lines (1)-(5). Formally, let $TR_0(\mathbf{A})$ be the scheme of the relation

obtained by execution of lines (1)-(5), where **A** is a set of attributes of any data type. Let A_1, A_2, \dots, A_n be n subsets of **A**. Let also **XFOLD** denote either **UNFOLD** or **FOLD**. Then, the result obtained by

REFORMAT AS XFOLD $A_1, \text{XFOLD } A_2, \dots, \text{XFOLD } A_m$

is a relation TR_m that matches the result of the sequence of m relational algebra operations

$$TR_i = \text{XFold}[A_i](TR_{i-1}), i = 1, 2, \dots, m.$$

Hence, the following is a valid expression.

REFORMAT AS UNFOLD A_1, A_2, A_3
FOLD A_4, A_5, A_6
UNFOLD A_7
FOLD A_8, A_9

If **P_OWNER** is the relation in Figure 5.16(b), this functionality is illustrated by the following examples.

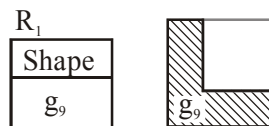


Figure 6.1: Illustration of **REFORMAT AS UNFOLD**.

Example 6.1: ‘Give the whole of the area that was owned either by John on date d25 or by Susan on date d45’.

```

SELECT Shape
FROM P_OWNER
WHERE (Owner = 'John' and Time cp [d25, d25]) or
        (Owner = 'Susan' and Time cp [d45, d45])
REFORMAT AS FOLD Shape

```

A *Fold* operation on space is applied to the relation returned by the execution of the first four lines, and yields relation R_1 in Figure 6.1.

R_2	
Owner	Time
John	[d21, d40]
Peter	[d21, d60]
Susan	[d31, d60]

Figure 6.2: Illustration of **REFORMAT AS FOLD**.

Example 6.2: ‘Give the ownership periods of every individual’.

```

SELECT Owner, Time
FROM P_OWNER
REFORMAT AS FOLD (Time)
    
```

A *Fold* on Time is applied to the result returned by the execution of the first two lines, and yields relation R₂ in Figure 6.2.

Normalise Clause

The <normalise clause> enables the application of a *Normalise* operation to the relation obtained by the execution of lines (1)-(6).

Formally, if TR is the relation obtained by the execution of lines (1)-(6) and A₁, A₂, ..., A_n is a sub-list of <select list>, then the SQL expression

```

NORMALISE ON A1, A2, ..., An
    
```

is equivalent to the relational algebra expression

$$Normalise[A_1, A_2, \dots, A_n](TR).$$

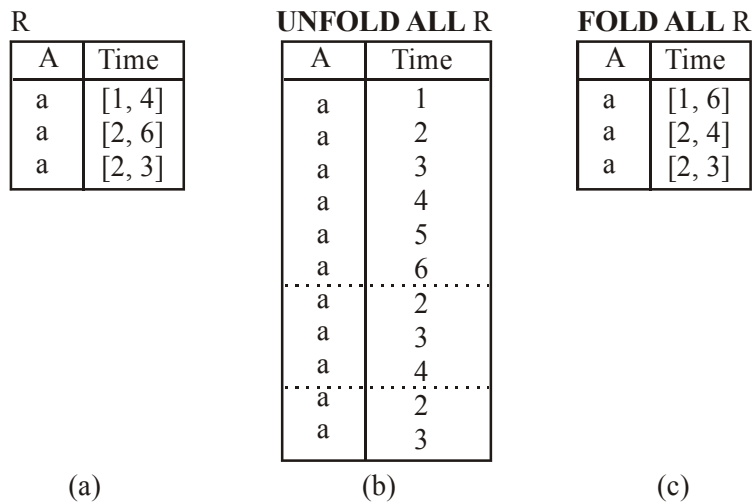


Figure 6.3 The [ALL] option in UNFOLD, FOLD and NORMALISE.

Example 6.3: Consider LAND_USE in Figure 5.16(a) and the query ‘Give the evolution of industrial land uses with respect to time’.

```

SELECT Use, Shape, Time
FROM LAND_USE
WHERE Use = 'Industrial'
NORMALISE ON Shape, Time
    
```

A *Normalise* operation on Shape and Time is applied to the result returned by the execution of the first three lines, and yields relation I in Figure 5.18.

Since duplicate tuples are allowed in SQL, a variation of **UNFOLD**, **FOLD** and **NORMALISE** is also provided below that enables obtaining relations with duplicate tuples. This functionality is included only for reasons of completeness and is illustrated by the example that follows, but it is not considered further.

Example 6.4: Consider the relation R in Figure 6.3(a). Then

```
SELECT Time
FROM R
REFORMAT AS UNFOLD ALL Time
```

yields the relation in Figure 6.3(b). If the last line is replaced by

- either ‘**REFORMAT AS FOLD ALL Time**’
- or ‘**NORMALISE ON ALL Time**’

then the relation in Figure 6.3(c) is obtained. The functionality of these options is similar when they are applied on more than one attribute.

The [ALL] option is incorporated in the remainder of the SQL extension but, its discussion is beyond the objectives of this thesis.

Sort rows on the basis of Space and Time columns

The total ordering defined for spatial objects (Section 3.4) and time periods (Section 5.3) enables incorporating references to attributes of these types after the key words ORDER BY. In addition, aggregate functions can be applied to attributes of these types.

6.3 Query Expression

In [LD96] an extension to the syntax of the SQL:1999 binary operations has been proposed, which enables the easy formulation of queries that return data, which evolves with respect to time. The proposed extension obeys the following algorithm.

Let $R(\mathbf{A})$, $S(\mathbf{B})$ be two non-temporal relations and let
 $R \text{ OPERATION}[\text{ALL}] S$

denote any SQL:1999 binary operation. Let also $TR(\mathbf{A}, T)$, $TS(\mathbf{B}, T)$ be the respective temporal relations. It is then defined that

$TR \text{ OPERATION} [\text{ALL}] \text{EXPANDING}(T) TS$

returns a relation with scheme and contents deduced by the execution of the following five steps:

- S1. Let $UR = TR \text{ UNFOLD} [\text{ALL}] (T)$, $US = TS \text{ UNFOLD} [\text{ALL}] (T)$.

S2. Let TIME be the relation returned by the expression

```
SELECT T FROM UR
UNION
SELECT T FROM US
```

S3. For every $t \in \text{TIME}$, let UR_t and US_t be the relations returned, respectively, by the expression

```
SELECT A FROM UR      SELECT B FROM US
WHERE T = 't'         WHERE T = 't'
```

S4. For every $t \in \text{TIME}$, let $\text{UP}_t(\mathbf{C})$ be the scheme of the relation obtained by the SQL:1999 binary operation $\text{UR}_t \text{ OPERATION } [\text{ALL}] \text{US}_t$.

S5. It is then defined that

$\text{TR OPERATION } [\text{ALL}] \text{EXPANDING}(\mathbf{T}) \text{TS}$
returns a relation $\text{P}(\mathbf{C}, \mathbf{T})$, where the domain of \mathbf{T} is of a period type. The rows of P are those obtained by steps S5.1 and S5.2 below.

S5.1 For every $t \in \text{TIME}$,
if \mathbf{c} is a row in UP_t
then add a row (\mathbf{c}, t) in $\text{UP}(\mathbf{C}, \mathbf{T})$.

S5.2 $\text{P} = \text{UP NORMALIZE } [\text{ALL}] (\mathbf{T})$.

By a similar argument, assume now that $\text{GR}(\mathbf{A}, \mathbf{G})$ and $\text{GS}(\mathbf{B}, \mathbf{G})$ are two spatial relations and let

$\text{GR SOPERATION } [\text{ALL}] \text{OF } (\mathbf{G}) \text{GS}$

be the syntax of one of the operations defined in Chapter 4. Let also $\text{TGR}(\mathbf{A}, \mathbf{G}, \mathbf{T})$, $\text{TGS}(\mathbf{B}, \mathbf{G}, \mathbf{T})$ be the respective spatio-temporal relations. By adopting the above algorithm, it then follows that

$\text{TGR SOPERATION } [\text{ALL}] \text{OF } (\mathbf{G}) \text{EXPANDING}(\mathbf{T}) \text{TGS}$

can be the syntax of the operation that gives the evolution of $\text{SOPERATION OF } (\mathbf{G})$ with respect to time. Indeed, this is the approach followed, in this thesis.

In SQL:1999 binary operations are involved in a $\langle \text{query expression} \rangle$, which can be either a $\langle \text{non-join query expression} \rangle$ or a $\langle \text{joined table} \rangle$. Here $\langle \text{query expression} \rangle$ is extended further, by the inclusion of a $\langle \text{unary query expression} \rangle$, as follows:

```
 $\langle \text{query expression} \rangle ::=$ 
   $\langle \text{IXSQL non-join query expression} \rangle$ 
  |  $\langle \text{IXSQL joined table} \rangle$ 
  |  $\langle \text{IXSQL unary query expression} \rangle$ 
```

The application of the above algorithm to each type of a query expression is addressed in one on the sections that follow.

6.4 Non-Join Query Expression

The syntax of the SQL:1999 non-join query expression, has been extended so as to support the quantum and pair-wise relational algebra operations.

Quantum Operations

In a simplified case (full syntax details are given in Appendix B), the syntax, to incorporate within SQL the quantum operations, is the following:

```

<non-join query expression> ::=
    <query exp 1> UNION [ALL]
        [EXPANDING (<reformat column list>)]
        <query exp 2>
    | <query exp 1> EXCEPT [ALL]
        [EXPANDING (<reformat column list>)]
        <query exp 2>
    | <query exp 1> INTERSECT [ALL]
        [EXPANDING(<reformat column list>)]
        <query exp 2>

```

The following syntactic rules apply:

Rule 6.2: <query exp 1> and <query exp 2> must return union compatible relations.

Rule 6.3: <reformat column list> must form a sub-list of the attributes of the relations returned by both <query exp 1> and <query exp 2>.

If the **EXPANDING** option is missing, then the above syntax matches that of the SQL <non-join query expression>. If this option is present then the algorithm of Section 6.3 is applied. If $R_1(\mathbf{A}, \mathbf{B})$, $R_2(\mathbf{A}, \mathbf{B})$ are the relations returned by <query exp 1> and <query exp 2>, respectively, then the result obtained by

```

<query exp 1> UNION      EXPANDING(<B>) <query exp 2>,
<query exp 1> EXCEPT  EXPANDING(<B>) <query exp 2>,
<query exp 1> INTERSECT EXPANDING(<B>) <query exp 2>,

```

matches, respectively, that obtained by the relational algebra operation

```

 $R_1 \ QUnion[\mathbf{B}] \ R_2,$ 
 $R_1 \ QExcept[\mathbf{B}] \ R_2,$ 
 $R_1 \ QIntersect[\mathbf{B}] \ R_2.$ 

```

The remainder examples of the chapter are based on relations LAND_USE and P_OWNER in Figure 5.16. The examples that follow apply quantum operations to a temporal, a spatial and a spatio-temporal relation. In a similar manner, any of the quantum operations can be applied to such relations.

R ₄
Time
[d21, d50]

Figure 6.4: Illustration of **UNION EXPANDING** (Time).

Example 6.5: ‘Give the periods of time during which there were industrial areas or John was the owner of some piece of land’.

```

SELECT Time
FROM   LAND_USE
WHERE  Use = 'Industrial'
UNION EXPANDING (Time)
SELECT Time
FROM   P_OWNER
WHERE  Owner = 'John'

```

The SQL expression selects two different sets of temporal data and it then applies the *Quantum Union* on time, yielding relation R₄ in Figure 6.4.

R ₅	
Use	Shape
Industrial	g ₁₀

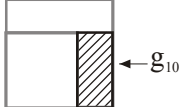


Figure 6.5: Illustration of **EXCEPT EXPANDING** (Shape).

Example 6.6: ‘Give the pieces of land on date d45 whose use was different on date d25’.

```

SELECT Use, Shape
FROM   LAND_USE
WHERE  Time_cp [d45, d45]
EXCEPT EXPANDING (Shape)
SELECT Use, Shape
FROM   LAND_USE
WHERE  Time_cp [d25, d25]

```

The SQL expression selects spatial data valid on two different dates and it then applies a *Quantum Except* on space, yielding relation R₅ in Figure 6.5.

Example 6.7: ‘Give the pieces of land that were industrial or were owned by Susan, and the respective periods as well’.

```

SELECT Shape, Time
FROM LAND_USE
WHERE Use = 'Industrial'
UNION EXPANDING (Shape, Time)
SELECT Shape, Time
FROM P_OWNER
WHERE Owner = 'Susan'

```

The SQL expression selects two different sets of spatio-temporal data and it then applies *Quantum Union* on space and time, yielding relation QU in Figure 5.17.

Pair-Wise Operations

In a simplified case (full syntax details are given in Appendix B), the syntax, to incorporate within SQL the pair-wise operations, is the following:

```

<non-join query expression> ::=
    <query exp 1> WUNION [ALL] OF (<ref col list 1>)
        [EXPANDING (<ref col list 2>)]
        <query exp 2>
    | <query exp 1> WEXCEPT [ALL] OF (<ref col list 1>)
        [EXPANDING (<ref col list 2>)]
        <query exp 2>
    | <query exp 1> WINTERSECT [ALL] OF (<ref col list 1>)
        [EXPANDING(<ref col list 2>)]
        <query exp 2>

```

The following syntactic rules apply:

Rule 6.4: <ref col list 1> and <ref col list 2> must form sub-lists of the list of attributes of the relations returned by both <query exp 1> and <query exp 2>.

Rule 6.5: <ref col list 1> and <ref col list 2> may not have attributes in common.

Note that sets of columns returned by <query exp 1> and <query exp 2> do not have to be disjoint because SQL:1999 does not impose such a restriction. If $R_1(\mathbf{A}, \mathbf{C})$, $R_2(\mathbf{B}, \mathbf{C})$ are the relations returned by <query exp 1> and <query exp 2>, respectively, then the result obtained by

```

<query exp 1> WUNION      OF(C) <query exp 2>,
<query exp 1> WEXCEPT  OF(C) <query exp 2>,
<query exp 1> WINTERSECT OF(C) <query exp 2>,

```

matches, respectively, the one obtained by the relational algebra operation

```

 $R_1 \text{ } WUnion[\mathbf{C}] \text{ } R_2,$ 
 $R_1 \text{ } WExcept[\mathbf{C}] \text{ } R_2,$ 
 $R_1 \text{ } WIntersect[\mathbf{C}] \text{ } R_2.$ 

```

If $R_1(A, G, T)$, $R_2(B, G, T)$ are two spatio-temporal relations, it is recalled that, in the general case, the result obtained by operation $R_1 \text{ } WUnion[G, T] R_2$ does not match the evolution with respect to time of the result obtained by operation $R_1 \text{ } WUnion[G] R_2$. The same observation also applies to operations $R_1 \text{ } WExcept[G, T] R_2$ and $R_1 \text{ } WExcept[G] R_2$. However, this evolution can be obtained by the use of the **EXPANDING** option and the application of the algorithm in Section 6.3. Therefore, the expressions

TABLE R_1 **WUNION** OF (G) **EXPANDING**(T) TABLE R_2 ,

TABLE R_1 **WEXCEPT** OF (G) **EXPANDING**(T) TABLE R_2 ,

TABLE R_1 **WINTERSECT** OF (G) **EXPANDING**(T) TABLE R_2 ,

yield, respectively, the evolution with respect to time of the data returned by operations

$R_1 \text{ } WUnion[G] R_2$,

$R_1 \text{ } WExcept[G] R_2$,

$R_1 \text{ } WIntersect[G] R_2$.

R ₆		
Owner	Use	Shape
John	Industrial	g ₁₁
John	Forest	g ₁₂

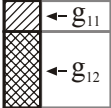


Figure 6.6: Illustration of **WINTERSECT OF (Shape)**.

Example 6.8: ‘For each owner, give the land he/she owned on date d25 and the use of this land on that date’.

```

SELECT Owner, Shape
FROM P_OWNER
WHERE Time_cp [d25, d25]
WINTERSECT OF (Shape)
SELECT Use, Shape
FROM LAND_USE
WHERE Time_cp [d25, d25]

```

The SQL expression selects two sets of spatial data and it then applies *Pair-Wise Intersect* on space. Relation R_6 in Figure 6.6 shows the result only for John.

Example 6.9: ‘Give the boundaries between Peter’s and Susan’s parcels on date d45’.

```

SELECT Pid AS Peter, Shape
FROM P_OWNER
WHERE Owner = 'Peter' and Time cp [d45, d45]
WINTERSECT OF (Shape)
SELECT Pid AS Susan, Shape
FROM P_OWNER
WHERE Owner = 'Susan' and Time cp [d45, d45]

```

As can be seen in relation R_7 in Figure 6.7, the spatial intersection yields lines.

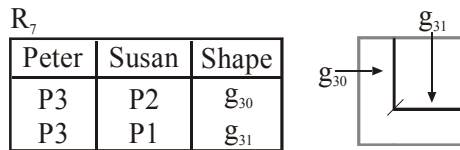


Figure 6.7: Illustration of **WINTERSECT OF (Shape)** yielding pure lines.

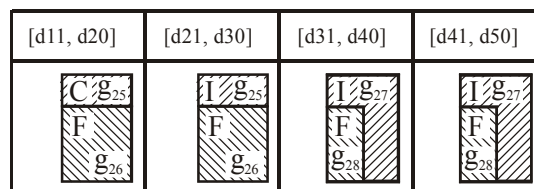
Example 6.10: ‘Give the evolution with respect to time of the land use of the area that parcel P1 occupied on date d25’.

```

SELECT Use, Shape, Time
FROM LAND_USE
WINTERSECT OF (Shape)
SELECT Shape
FROM P_OWNER
WHERE Pid = 'P1' and Time cp [d25, d25]

```

The SQL expression yields the evolution with respect to time of spatial data in a given area (R_8 in Figure 6.8).



R_8

Use	Shape	Time
Cultivation	g_{25}	[d11, d20]
Forest	g_{26}	[d11, d30]
Forest	g_{28}	[d31, d50]
Industrial	g_{25}	[d21, d30]
Industrial	g_{27}	[d31, d50]

Figure 6.8: Illustration of the obtaining of evolution of spatial data with respect to time in a given area.

R_9

Use	Time
Cultivation	[d11, d20]
Industrial	[d21, d30]
Forest	[d11, d30]

Figure 6.9: Illustration of **WEXCEPT OF** (Time).

Example 6.11: ‘Give the life spans of land uses outside the lifespan of parcel P3’.

```
SELECT Use, Time
FROM LAND_USE
WEXCEPT OF (Time)
SELECT Time
FROM OWNERSHIP
WHERE Pid = 'P3'
```

The SQL expression selects two sets of temporal data and then it applies *Pair-Wise Except* on time, yielding relation R_9 in Figure 6.9.

Example 6.12: ‘Give the land use of each piece of land while it was not owned by Peter and while it was not owned by Susan. Give also the respective time and the name of these two persons who did not own this piece of land’.

```
SELECT Use, Shape, Time
FROM LAND_USE
WEXCEPT OF (Shape, Time)
SELECT Owner, Shape, Time
FROM P_OWNER
WHERE Owner = 'Peter' or Owner = 'Susan'
```

Relation SWE in Figure 5.20 contains part of the result, concerning pieces of industrial use.

Example 6.13: ‘Give the spatial union of the pieces of land of every land use with every land owner that were valid at the same time. Give also the respective valid time’.

```
SELECT Use, Shape, Time
FROM LAND_USE
WUNION OF (Shape) EXPANDING (Time)
SELECT Owner, Shape, Time
FROM P_OWNER
```

Relation W in Figure 5.19 contains part of the result, concerning pieces of industrial use and owners Peter and Susan.

6.5 Joined Table

Beyond the extension by the **EXPANDING** option of the various types of the SQL:1999 join operations [LD96], a syntax has also been provided for joined tables, that enables the explicit application of the various types of overlay operations. This syntax is given below.

```
<IXSQL overlay> ::=
    <table ref 1> [NATURAL][<overlay type>] OVERLAY [ALL]
    [OF (<ref col list 1>)]
    [EXPANDING (<ref col list 2>)]
    <table ref 2>
```

```
<overlay type> ::=
    INNER
    | {LEFT | RIGHT | FULL} [OUTER]
```

The following rule applies:

Rule 6.6: Exactly one of the options, either **NATURAL** or **OF (<ref col list 1>)** must be specified.

If the option **<overlay type>** is not specified then **INNER** is assumed by default. The option **NATURAL** is equivalent to the option **OF (<ref col list 1>)**, where **<ref col list 1>** is the list of all attributes in the result of both **<query exp 1>** and **<query exp 2>**. Recall that **<ref col list 1>** and **<ref col list 2>** must be disjoint sub-lists of the attributes of the relations returned by **<query exp 1>** and **<query exp 2>** (Rules 6.4 and 6.5). If $R_1(\mathbf{A}, \mathbf{C})$, $R_2(\mathbf{B}, \mathbf{C})$ are the relations returned by **<query exp 1>** and **<query exp 2>**, respectively, then the result obtained by

```
<table ref 1> INNER OVERLAY OF(C) <table ref 2>,
<table ref 1> LEFT OVERLAY OF(C) <table ref 2>,
<table ref 1> RIGHT OVERLAY OF(C) <table ref 2>,
<table ref 1> FULL OVERLAY OF(C) <table ref 2>,
```

matches, respectively, the one obtained by the relational algebra operation

```
 $R_1 \text{ } I\text{Overlay}[\mathbf{C}] \text{ } R_2,$ 
 $R_1 \text{ } L\text{Overlay}[\mathbf{C}] \text{ } R_2,$ 
 $R_1 \text{ } R\text{Overlay}[\mathbf{C}] \text{ } R_2,$ 
 $R_1 \text{ } F\text{Overlay}[\mathbf{C}] \text{ } R_2.$ 
```

If $R_1(\mathbf{A}, \mathbf{G}, \mathbf{T})$, $R_2(\mathbf{B}, \mathbf{G}, \mathbf{T})$ are two spatio-temporal relations, it is recalled (Proposition 5.4) that operation $R_1 \text{ } F\text{Overlay}[\mathbf{G}, \mathbf{T}] \text{ } R_2$ yields the evolution with respect to time of operation $R_1 \text{ } F\text{Overlay}[\mathbf{G}] \text{ } R_2$. The same observation

applies to all the other types of overlay operations. It is therefore deduced that the following two expressions are equivalent:

R_1 **FULL OVERLAY OF**(G, T) R_2 .
 R_1 **FULL OVERLAY OF**(G) **EXPANDING**(T) R_2 .

It is also easy to realise that the first of them is equivalent to

R_1 **FULL JOIN EXPANDING**(G, T) R_2 **ON** (1=1)

Therefore, all three of them are equivalent. The same observation applies to all the other types of overlay operations.

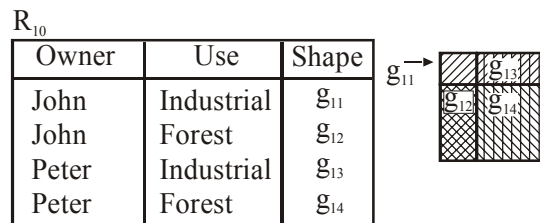


Figure 6.10: Illustration of **FULL OVERLAY EXPANDING** (Shape).

Example 6.14: ‘Give the uses of all the areas and their owner on date d25’.

```
(SELECT Owner, Shape
FROM P_OWNER
WHERE Time cp [d25, d25])
FULL OVERLAY OF (Shape)
(SELECT Use, Shape
FROM LAND_USE
WHERE Time cp [d25, d25])
```

The result relation can be seen in Figure 6.10.

Example 6.15: ‘For all the recorded times, associate industrial land with land owned by Susan’.

```
(SELECT Use, Shape, Time
FROM LAND_USE
WHERE Use = 'Industrial')
FULL OVERLAY OF (Shape, Time)
(SELECT Owner, Shape, Time
FROM P_OWNER
WHERE Owner = 'Susan')
```

This yields relation FO in Figure 5.22. In a similar manner, it is also possible to apply any of the other overlay operations.

6.6 Unary Query Expression

The definition of <unary query expression> enables incorporating in SQL the functionality of the relational algebra operations *Complementation*, *Boundary*, *Envelope* and *Buffer*. An obvious revision of the algorithm given in Section 6.3 is also considered, which enables defining the evolution of these operations with respect to time. Hence, the syntax is as follows:

```
<unary query expression> ::=
  (<unary query expression>
  | <table ref> { COMPLEMENTATION
                 | BOUNDARY
                 | ENVELOPE [ALL]}
                 OF (<ref col list 1>)
                 [EXPANDING (<ref col list 2>)]
  | <table ref> BUFFER [ALL]
                 OF (<ref col list 1>)
                 WITHIN DISTANCE (<ref col list 3>)
                 [EXPANDING (<ref col list 2>)]
```

The following rules apply:

Rule 6.7: <ref col list 1>, <ref col list 2> and <ref col list 3> must form disjoint sub-lists of the attributes of the relations returned by <table ref>.

Rule 6.8: The data type of attributes in <ref col list 3> must be numeric.

Rule 6.9: The number of attributes in <ref col list 2> must be identical to the number of attributes in <ref col list 3>.

If R is the relation obtained by <table ref>, then the result obtained by
 <table ref> **COMPLEMENTATION OF** (<ref col list 1>),
 <table ref> **BOUNDARY OF** (<ref col list 1>),
 <table ref> **ENVELOPE OF** (<ref col list 1>),
 <table ref> **BUFFER OF** (<ref col list 1>

WITHIN DISTANCE (<ref col list 3>),

matches, respectively, that obtained by the relational algebra operation

Complementation[<ref col list 1>](<table ref>),

Boundary[<ref col list 1>](<table ref>),

Envelope[<ref col list 1>](<table ref>),

Buffer[<ref col list 3>, <ref col list 1>](<table ref>).

Recall that *Boundary*[G, T] (*Envelope*[G, T]) yields the evolution with respect to time of operation *Boundary*[G] (*Envelope*[G]). Hence,

R **BOUNDARY OF** (G, T) and

R ENVELOPE OF (G, T)
 are, respectively, equivalent to
R BOUNDARY OF (G) EXPANDING (T)
R ENVELOPE OF (G) EXPANDING (T)

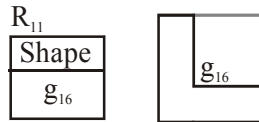


Figure 6.11: Illustration of **BOUNDARY OF (Shape)**.

Example 6.16: ‘Give the boundary of the area owned by either John or Peter on date d35’.

```
(SELECT Shape
  FROM P_OWNER
  WHERE Time cp [d35, d35] and
        (Owner = 'John' or Owner = 'Peter'))
  BOUNDARY OF (Shape)
```

The result is relation R_{11} in Figure 6.11.

Example 6.17: ‘Give the boundary of the land owned by Susan on various dates’.

```
(SELECT Shape, Time
  FROM P_OWNER
  WHERE Owner = 'Susan')
  BOUNDARY OF (Shape) EXPANDING (Time)
```

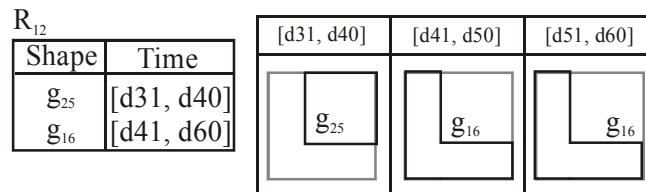


Figure 6.12: Illustration of **BOUNDARY OF (Shape) EXPANDING (Time)**.

The result is relation R_{12} in Figure 6.12. In a similar manner, it is also possible to apply any of the unary operations.

Note that the proposed syntax enables the application of a sequence of operations *Complementation*, *Boundary*, *Envelope*, *Buffer*, to an input relation. Thus, the SQL expression of Example 6.17 can be extended as follows.

Example 6.18: ‘Give a buffer area of 10 around the boundary of the land owned by Susan on various dates’.

```
((SELECT 10 AS D, Shape, Time
  FROM P_OWNER
  WHERE Owner = 'Susan')
  BOUNDARY OF (Shape) EXPANDING (Time))
  BUFFER OF (Shape) WITHIN DISTANCE (D) EXPANDING (Time)
```

6.7 Conclusions

An SQL extension for the management of spatial and spatio-temporal data was presented. The work is a further extension, and also fully compatible, to another extension [LM97, LD96], which aimed at the management of temporal and, more generally, interval data. The extension is fully compatible with the ISO SQL:1999 standard [Iso99] and enables incorporating the whole power of the relational algebra operations, defined in Chapters 4 and 5, within SQL.

CHAPTER 7

COMPARISON WITH OTHER APPROACHES

7.1 Introduction

The model proposed in this thesis is now compared with known spatial and spatio-temporal approaches. To achieve this, the properties identified in Chapter 2 are considered as a yardstick. Hence, both the various spatial and spatio-temporal approaches and also the model proposed in this thesis are evaluated with respect to them. This way, it is shown that this model inherits the functionality of almost all other approaches. The evaluation with respect to some property is marked as follows:

- Y (Yes): The property is satisfied.
- N (No): The property is not satisfied.
- P (Partially): The property is satisfied partially.
- N/A (Not Applicable): The property is not applicable.
- E (Estimated) : It is not explicitly stated in the literature that the property is satisfied, but it is estimated so.
- ? (Not clear): It is not clear from the literature whether the property is satisfied.

Effort has been made for the evaluation to be objective. Note that this has not been an easy task, since many approaches lack formalism.

The remainder of the chapter is organized as follows: In Section 7.2 each of the spatial approaches reviewed in Section 2.2 are classified according to the characteristics analysed in Section 2.4 and are evaluated with respect to the properties specified in Section 2.8. In Section 7.3 each of the spatio-temporal approaches reviewed in Section 2.3 are classified according to the characteristics analysed in Section 2.6 and are evaluated with respect to the properties specified in Section 2.8. Note that the use of arcs in vector based spatial approaches has been neglected in the evaluation of user-friendly data types. In Section 7.4, the model formalised in this thesis is also classified and evaluated. In Section 7.5 the proposed model is evaluated with respect to the

functionality of other approaches. Additional characteristics of the model are presented in Section 7.6. Similarities with other approaches are identified in Section 7.7. Finally, conclusions are drawn in the last section.

7.2 Classification and Evaluation of Spatial Approaches

Short explanations are given, wherever this is estimated to be necessary, in the classification and evaluation of each model.

7.2.1 ROSE Algebra

References: [GS93, GS95]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Definition of a set of spatial data types and operations between them.

C3. Database or GIS Centric: Data types integrated in Database-centric architecture.

P2.1 Point, Pure Line, Pure Surface: Set-valued spatial objects allowed.

P4. Spatial Data Validation Mechanisms: It is not satisfied. For example, the domain of *G_city* in *CAPITAL_CITIES(city_name, country_name, G_city)* has to be of a set of points data type. Clearly, however, more than one spatial point can be recorded for a given city name.

P8. No Data Loss in Operations: Note that distinct operations have to be applied to obtain the surface, line and point parts of the spatial intersection of two surfaces.

P9.2 Fold: Supported by combination of *Fusion* and *Decompose*. Applied only to spatial objects of the same type.

P9.4-P9.13 Quantum, Pair-Wise and Overlay Operations: All are supported indirectly by functions for spatial union, difference and intersection. Limitations are identified: Spatial union and difference must be applied to spatial objects of the same data type. *Overlay* is applied only to pure surfaces and gives again only pure surfaces.

	ROSE	TMA	ESSPM	HTGIS	ARCINFO	GEOMEDIA	MAPINFO	GEO GRAPHICS	GRID GIS	GEO SABRINA	ESSQL	PSQL	SVRA	GNTRM
Model Classification														
C1	D	C	D	D	D	D	D	D	C	D	D	D	D	D
C2	V	N/A	N/A	V	V	V	V	V	R	V	V/R	V/R	V	R
C3	D			G	G	G	G	G	G	D	D	D	D	D
Model Evaluation														
P1 Formalism for Spatial Types and Operations														
P1(a)	Y	N	Y	N	N/A	N/A	N/A	N/A	N/A	N	N	N	N	Y
P1(b)	Y	N	Y	P	N/A	N/A	N/A	N/A	N/A	N	N	N	N	Y
P2 Spatial Data Types Matching Human Perception														
P2.1(a)	N	N	N	Y	Y	N	N	N	N	N	?	?	N	N
P2.1(b)	N	N	N	Y	N	N	N	N	N	N	?	?	N	N
P2.1(c)	N	N	Y	Y	N	N	N	N	Y	N	?	?	N	N
P2.2 Compatibility between Spatial Data Types														
P2.2.3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	?	Y
P2.3 Set-Theoretically Closed Spatial Objects														
P2.4	N	Y	Y	Y	N	Y	?	Y	Y	N	N	?	N	N
P2.4 Empty Set not a Valid Spatial Object														
P3 Support of Hybrid Surfaces														
P4	N	Y	N	P	N	N	N	N	N	N	?	?	N	N
P4 Spatial Data Validation Mechanisms														
P5	N	N/A	N/A	Y	N	N	?	N	N/A	N	?	?	N	N/A
P5 Non-Connected Spatial Objects not Required														
P6	Y	N/A	N/A	N	Y	Y	Y	N	N/A	Y	Y	Y	Y	Y
P6 Complex Data Structures not Required														
P7	Y	N/A	N/A	N	P	P	P	N	N/A	Y	Y	Y	Y	Y
P7 No Limitations on Data Structures														
P8	Y	Y	N	N	N	Y	N/A	N	N	?	N/A	N	N	N
P8 No Data Loss in Operations														

Figure 7.1(a): Classification and evaluation of spatial approaches.

	ROSE	TMA	ESSPM	HTGIS	ARCINFO	GEOMEDIA	MAPINFO	GEO GRAPHICS	GRID GIS	GEO SABRINA	ESSQL	PSQL	SVRA	GNTRM
Model Evaluation														
P9 Functionality of Spatial Operations														
P9.1 Basic Relational	N/A	N/A	N/A	P	Y	Y	Y	Y	N/A	Y	Y	Y	Y	Y
P9.2 Fold	P	P	P	P	P	Y	N	N	P	Y	N	N	N	P
P9.3 Unfold														
P9.4 Quantum Union	P	N/A	N/A	P	P	Y	N	N	N/A	Y	N	N	N	P
P9.5 Quantum Except	P	N/A	N/A	P	N	N	N	N	N/A	Y	N	N	N	P
P9.6 Quantum Intersect	Y	N/A	N/A	P	P	Y	N	P	N/A	Y	N	P	P	P
P9.7 Pair-Wise Union	P	N/A	N/A	P	P	Y	N	N	N/A	Y	N	N	N	P
P9.8 Pair-Wise Except	P	N/A	N/A	P	N	N	N	N	N/A	Y	N	N	N	P
P9.9 Pair-Wise Intersect	Y	N/A	N/A	P	P	Y	N	P	N/A	Y	N	P	P	P
P9.10 Inner Overlay	Y	Y	P	P	P	Y	N	P	P	Y	N	P	P	P
P9.11 Left Overlay	P	Y	P	P	P	Y	N	P	P	Y	N	N	N	P
P9.12 Right Overlay	P	Y	P	P	P	Y	N	P	P	Y	N	N	N	P
P9.13 Full Overlay	P	Y	P	P	P	Y	N	P	P	Y	N	N	N	P
P9.14 Complementatation	N	N/A	N/A	?	N	N	N	N	N/A	N	?	N	N	P
P9.15 Boundary	Y	N/A	N/A	?	?	N	N	N	N/A	N	P	N	Y	N
P9.16 Envelope	Y	N	N	N	Y	Y	N	N	N	N	N	N	N	Y
P9.17 Buffer	N	Y	N	Y	Y	Y	N	N	N	N	N	N	N	P
P10 Limited Number of Kernel Operations	N	N	Y	Y	N	N	N	N	N	N	N	N	N	Y
P11 Dimension Independent Spatial Types and Operations	N	E	E	E	N	N	N	N	N	N	N	N	N	E
P12 Implementation Available	Y	Y	N	?	Y	Y	Y	Y	Y	Y	?	Y	Y	?

Figure 7.1(b): Classification and evaluation of spatial approaches.

	GEOSAL	GRAL	QLG	DEDALE	CALG	RCM	SVTMM	ISOSQLMM	OGISSQL	ORACLE	INFORMIX	DB2	POSTGRESQL	GEO++	GEUS	OOGDM	
Model Classification																	
C1	D	D	C	C	D	D	D	D	D	D	D	D	D	D	D	D	D
C2	V	V	N/A	N/A	N/A	N/A	N/A	V	V	V	V	V	V	V	V	V	V
C3	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
Model Evaluation																	
P1	Y	Y	Y	Y	Y	Y	Y	N	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
P1(a)	N	N	N	Y	Y	Y	Y	N	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
P1(b)	N	N	N	Y	Y	Y	Y	N	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N
P2	Y	Y	N	N	N	Y	N	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
P2.1(a)	P	P	Y	N	N	N	N	P	P	N	P	P	P	P	P	P	P
P2.1(b)	P	P	Y	N	N	N	N	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
P2.1(c)	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	Y
P2.2	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
P2.3	Y	Y	Y	Y	Y	Y	N	N	N	P	N	N	Y	Y	Y	Y	N
P2.4	N	N	P	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	N
P3	Y	Y	Y	N	N	N	N	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
P4	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N
P5	Y	Y	Y	N	N	Y	N	N	N	N	N	N	Y	Y	Y	Y	N
P6	Y	Y	N	Y	Y	N	N	Y	Y	Y	Y	Y	N	N	N	N	Y
P7	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
P8	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N/A

Figure 7.1(c): Classification and evaluation of spatial approaches.

	GEOSAL	GRAL	QLC	DEDATE	CAIG	RCM	SVTMM	ISOSQLMM	OGISSQL	ORACLE	INFORMIX	DB2	POSTGRESQL	GEO+	GEUS	OGDM
Model Evaluation																
P9 Functionality of Spatial Operations																
P9.1 Basic Relational	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
P9.2 Fold	P	N	Y	Y	Y	P	P	N	N	N	P	N	N	N	N	N
P9.3 Unfold																
P9.4 Quantum Union	P	N	Y	Y	Y	P	P	P	P	P	P	P	N	N	N	N
P9.5 Quantum Except	P	N	N	Y	Y	P	P	P	P	P	P	P	N	N	N	N
P9.6 Quantum Intersect	P	P	Y	Y	Y	P	P	Y	Y	Y	P	P	P	P	N	P
P9.7 Pair-Wise Union	P	N	Y	Y	Y	P	P	P	P	P	P	P	N	N	N	N
P9.8 Pair-Wise Except	P	N	N	Y	Y	P	P	P	P	P	P	P	N	N	N	N
P9.9 Pair-Wise Intersect	P	P	Y	Y	Y	P	P	Y	Y	Y	P	P	P	P	N	P
P9.10 Inner Overlay	P	P	Y	Y	Y	P	P	Y	Y	Y	P	P	P	P	N	P
P9.11 Left Overlay	P	N	N	Y	Y	P	P	N	N	N	P	N	N	N	N	N
P9.12 Right Overlay	P	N	N	Y	Y	P	P	N	N	N	P	N	N	N	N	N
P9.13 Full Overlay	P	N	N	Y	Y	P	P	N	N	N	P	N	N	N	N	N
P9.14 Complementation	N	N	N	Y	N	P	P	N	N	N	N	N	N	N	N	N
P9.15 Boundary	P	N	Y	Y	Y	N	P	P	P	N	P	P	P	P	N	P
P9.16 Envelope	Y	N	Y	N	N	P	N	Y	Y	Y	Y	Y	N	N	N	N
P9.17 Buffer	Y	N	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N	N	N	Y
P10 Limited Number of Kernel Operations	N	N	N	Y	Y	P	P	N	N	N	N	N	N	N	N	N
P11 Dimension Independent Spatial Types and Operations	N	N	N	Y	Y	E	E	N	N	N	N	N	N	N	N	N
P12 Implementation Available	Y	Y	Y	Y	Y	N	?	N/A	N/A	Y	Y	Y	Y	Y	Y	Y

Figure 7.1(d): Classification and evaluation of spatial approaches.

7.2.2 Tomlin's Map Algebra (TMA)

References: [To90, To91, To94, To97] *Classification-Evaluation Tables:*
 Figures 7.1(a-b)

Objective: Abstract description of a classification of operations for the management of maps of any type.

C1. Discrete or Continuous Change in Space: The set of operations enables the management of data that changes continuously in space.

C2. Vector or Raster-Based: Both vector and raster representations are possible at implementation level.

C3. Database or GIS Centric: The approach considers operations on maps.

P2.1 Point, Pure Line, Pure Surface: Only one type of zone (set of points) is supported in maps.

P9.2 Fold: Points with the same thematic value are automatically coalesced into the same zone.

P9.4-P9.9 Quantum and Pair-Wise Operations: This model aims at defining operations on maps. Therefore, these properties are not applicable.

P9.10-P9.13 Overlay Operations: Supported by operation *LocalCombination*.

P10. Limited Number of Kernel Operations: The set of operations is too long and does not intend to be complete.

P11. Dimension Independent Spatial Types and Operations: It is estimated that the model can easily be generalized to an n-d space.

7.2.3 Erwig and Schneider's Spatial Partition Model (ESSPM)

References: [ES97, ES00] *Classification-Evaluation Tables:*
 Figures 7.1(a-b)

Objective: Formalism for *spatial partitions* (thematic maps) and operations.

C2. Vector or Raster-Based: Conceptual model.

C3. Database or GIS Centric: The approach considers operations on partitions.

P2.1(a), P2.1(b) Point, Pure Line: Spatial Partitions contain only pure surfaces.

P2.2 Compatibility between Spatial Data Types: N/A, due to previous remark.

P3. Support of Hybrid Surfaces: Not satisfied, due to previous remark.

P8. No Data Loss in Operations: The spatial intersection of pure surfaces (achieved by operation *Intersection* on *spatial partitions*) yields only pure surfaces.

P9.2 Fold: Points and pure lines are not supported.

P9.4-P9.9 Quantum and Pair-Wise Operations: They are not applicable because they are operations on relations whereas the approach defines a map model.

P9.10-P9.13 Overlay Operations: Only pure surfaces are supported.

P9.14, P9.15 Complementation and Boundary: See Properties P9.4-P9.9.

7.2.4 Hadzilacos and Tryfona's GIS Model (HTGIS)

References: [HT96, DHT94, HT92]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Combination of the map model defined in [DHT94] with the relational model.

P1. Formalism for Spatial Types and Operations: Only the concept of layer and some operations are formalized.

P3. Support of Hybrid Surfaces: Overlapping is not allowed between spatial objects of the same layer. Thus, a hybrid surface can be represented indirectly as a set of pure lines and pure surfaces.

P7. No Limitations on Data Structures: Only one spatial attribute is allowed in a layer (recall that a *layer* is a relation with only one spatial attribute).

P6. Complex Data Structures not Required: Different data structures are used for the modelling of spatial and conventional data.

P8. No Data Loss in Operations: The spatial intersection of two pure surfaces yields only pure surfaces.

P9.1 Basic Relational: They are supported for conventional data but they cannot be applied to layers.

P9.2 Fold: It can be applied only to layers of either pure lines or of pure surfaces.

P9.4-P9.9 Quantum and Pair-Wise Operations: Spatial union, difference and intersection are applied to spatial objects of the same data type. Conventional attributes of input layers are not maintained in the result.

P9.10-P9.13 Overlay Operations: The *overlay* of two layers of pure surfaces yields only pure surfaces. Similar observation applies to other spatial data types.

7.2.5 ESRI ArcInfo 8

References: [Esri99, Esri00, Ze99, Ma99, Tu00, MSW99, Esri02a, Esri02b] *Classification-Evaluation Tables:* Figures 7.1(a-b)

Objective: Providing functionality for the development of GIS applications.

P2.1(b, c) Pure Line, Pure Surface: Actually sets of pure lines and pure surfaces are supported.

P4. Spatial Data Validation Mechanisms: Not supported due to the previous remark.

P6. Complex Data Structures not Required: Same set of tools for conventional and spatial structures.

P7. No Limitations on Data Structures: Only one spatial attribute at a time is considered in spatial operations.

P8. No Data Loss in Operations: The spatial intersection of two pure surfaces yields only pure surfaces.

P9.2 Fold: Operation *Dissolve* applies explicitly either to pure surfaces or to pure lines.

P9.4, P9.7 Quantum and Pair-Wise Union: Achieved with operation *Dissolve*, but the operation applies explicitly to either pure surfaces or pure lines.

P9.5, P9.8 Quantum and Pair-Wise Except: Operation *Erase* does not enable achieving Quantum or Pair-Wise Except.

P9.6, P9.9 Quantum and Pair-Wise Intersect, Overlay: Only pure surfaces must be recorded in the second feature class and data loss was identified in Property P8.

7.2.6 Intergraph Geomedia 5

References: [LH98, Int02]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Providing functionality for the development of GIS applications.

The behaviour of this approach is close to the one of the previous subsection, therefore, only main differences are addressed below.

P2.1(a) Point: Sets of points are supported.

P2.2 Compatibility between Spatial Data Types: Data type COMPOUND enables recording spatial objects of any type in the same relation.

P3. Support of Hybrid Surfaces: Objects HS in Figures 2.12(c-d) are valid.

P8. No Data Loss in Operations: Satisfied.

P9.2, P9.4, P9.7 Fold, Quantum and Pair-Wise Union: Supported.

P9.6, P9.9-P9.13 Quantum and Pair-Wise Intersect and Overlay Operations: Supported.

7.2.7 MapInfo Professional 7

References: [Mapi01, Mapi02]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Providing functionality for the development of GIS applications.

The behaviour of this approach is close to the one of the previous subsection, therefore, only main differences are addressed below.

P2.1 Point, Pure Line, Pure Surface: Only one spatial data type is supported.

P2.4 Empty Set not a Valid Spatial Object: Not clear.

P5. Non-Connected Spatial Objects not Required: Not clear.

P8. No Data Loss in Operations: Relevant operations not provided.

P9. Functionality of Spatial Operations: Relevant operations not provided.

7.2.8 Bentley Microstation Geographics 7.2

References: [Bent01]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Providing functionality for the development of GIS applications.

The behaviour of this approach is close to the one in Subsection 7.2.6, therefore, only main differences are addressed below.

P6. Complex Data Structures not Required: Conventional data is stored in relations whereas spatial data is stored in CAD files.

P7. No Limitations on Data Structures: Only one attribute to record spatial data is allowed.

P9.6, P9.9-P9.13 Quantum and Pair-Wise Intersect and Overlay Operations: They are supported only for pure surfaces.

7.2.9 Grid Based Commercial GIS Tools (GRIDGIS)

References: [Esri01, MJ01, Keig02, Gras02, Lo00, BK01, RHS01] *Classification-Evaluation Tables:* Figures 7.1(a-b)

Objective: Providing functionality for the management of maps with properties that change continuously in space.

P2.1(a), P2.1(b) Point and Pure Line: Only pure surfaces are supported.

P8. No Data Loss in Operations: Points and pure lines are not supported.

P9.2 Fold: Conceptually, cells with identical value are automatically coalesced into zones.

P9.10-P9.13 Overlay Operations: The *Combine* operation supports overlay for maps of pixels.

7.2.10 Geosabrina

References: [LPV93, CVL+94, YC94a] *Classification-Evaluation Tables:* Figures 7.1(a-b)

Objective: Description of the characteristics of a spatial extension of a relational DBMS (Formalism not provided).

P2.1 Point, Pure Line, Pure Surface: Although the set of data types is extensible, only one spatial data type, GEOMETRY, is supported.

P2.4 Empty Set not a Valid Spatial Object: It is explicitly declared that spatial objects are “sets of elements that are either points, connected lines or connected regions accepting holes”. One predicate checks whether the result of a spatial intersection is the empty set [LPV93].

P3. Support of Hybrid Surfaces: Objects HS in Figures 2.12(c-d) are valid.

P8. No Data Loss in Operations: Operations are not defined.

P9.2, P9.4-P9.13 Fold, Quantum, Pair-Wise and Overlay Operations: Spatial union, difference and intersection are supported as functions. Spatial union and intersection are also supported as aggregate functions but they are not formalised.

7.2.11 Egenhofer's Spatial SQL (ESSQL)

References: [Eg94, Eg89, EF91]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Description of requirements for a spatial SQL, with particular interest to spatial data presentation issues. It does not aim at defining a set of spatial data types and operations.

P2.1, P2.2 Point, Pure Line, Pure Surface and Compatibility: Not clear, since spatial data types are not defined.

P2.4 Empty Set not a Valid Spatial Object: The boundary of a point is the empty set [Eg89].

P8. No Data Loss in Operations: Operations are not defined.

P9.2-P9.13 Fold, Quantum, Pair-Wise and Overlay Operations: Not mentioned.

P9.15 Boundary: The boundary of a pure line is a set of points and the boundary of a point is the empty set.

Further Observations: The boundary of a point is the empty set and that of a pure line is a set of points. Note however that in mathematics, the boundary of a point is the point itself and the boundary of a pure line is the pure line itself.

7.2.12 Pictorial SQL (PSQL)

References: [RFS88]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Efficient management of spatial data by the use of R^+ trees rather than the formalization of spatial data types and operations.

Only the main differences from the previous approach are described here.

C2. Vector or Raster-Based: Both representations are supported internally.

P8. No Data Loss in Operations: Spatial intersection of two lines gives only points.

P9.6, P9.9, P9.10 Quantum Intersect, Pair-Wise Intersect and Inner Overlay: Only one function is provided that computes the intersection points of two pure lines.

7.2.13 Scholl and Voisard's Relational Approach (SVRA)

References: [SV92]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Reporting the experience of implementing a relational-like model in an object-oriented DBMS.

P2.1 Point, Pure Line, Pure Surface: Set-valued spatial objects supported for the three primitive data types.

P2.3 Set Theoretically Closed Spatial Objects: The support of open or closed spatial objects is left open in the approach.

P8. No Data Loss in Operations: The function for spatial intersection of two pure surfaces returns only pure surfaces.

P9.4, P9.5, P9.7, P9.8 Quantum and Pair-Wise Union and Except: Functionality for spatial union and difference is supported.

P9.9, P9.10 Pair-Wise Intersect and Inner Overlay: Functions for spatial intersection have limitations: (i) they cannot be applied to data of any two spatial data types and (ii) data loss is identified (see Property P8).

P9.11-P9.13 Left, Right and Full Overlay: Spatial difference is not supported.

7.2.14 Gargano, Nardelli and Talamo's Relational Model (GNTRM)

References: [GNT91a, GNT91b]

Classification-Evaluation Tables:
Figures 7.1(a-b)

Objective: Definition of a relational model for spatial data management.

P2.1 Point, Pure Line, Pure Surface: Only pure surfaces are supported, as sets of raster pixels.

P5. Non-Connected Spatial Objects not Required: Operations *G-Compose* and *G-Fusion* produce non-connected pure surfaces.

P8. No Data Loss in Operations: Only pure surfaces are supported.

P9. Functionality of Spatial Operations: Boundary is not supported, since pure lines are not supported either. Basic relational operations are fully

supported. The remainder are partially supported, since they are applied only to pure surfaces.

P10. Limited Number of Kernel Operations: Only three primitive operations.

7.2.15 GeoSAL

References: [SH91, HSH92, HS93]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Definition of a many sorted algebra for the manipulation of spatial objects.

C1. Discrete or Continuous Change in Space: Operations for continuous change are defined between raster grids.

C2. Vector or Raster-Based: Although raster grids are included in the model, their representation is vector-based.

P2.1(b), P2.1(c) Pure Line and Pure Surface: Lines must be simple and surfaces may not have holes.

P8. No Data Loss in Operations: The spatial intersection of two pure surfaces yields only pure surfaces. However, the intersection of two pure lines yields only isolated intersection points.

P9.2 Fold: It is supported only for lines or polygons.

P9.4 – P9.13 Quantum, Pair-Wise and Overlay Operations: They are supported only between spatial objects of the same data type, either LINE or POLYGON.

P9.15 Boundary: Supported only for polygons without holes.

7.2.16 Geo Relational Algebra (GRAL)

References: [Gu88, Gu89]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Definition of a many sorted algebra for the manipulation of spatial objects.

Only main differences from previous approach are described here.

P9.2 Fold: It is not supported.

P9.6, P9.9, P9.10 Quantum and Pair-Wise Intersect and Inner Overlay: Operation *Intersection* resembles *Pair-Wise Intersection*. Limitations are: (i)

It cannot be applied to any two spatial types and (ii) Data loss has been identified.

P9.15 Boundary: Not supported.

7.2.17 QLG

References: [CZ96, CW96]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Definition of data types and operations that enable the management of spatial data in a *Nested Relational* model.

P3. Support of Hybrid Surfaces: Spatial object HS in Figures 2.12(c-d) is valid. However, after the application of spatial union and intersection redundant data can be eliminated.

P6. Complex Data Structures not Required: Set-valued attributes are required for some operations such as spatial intersection.

7.2.18 Dedale

References: [GRSS97, GRS98a,
GRS98b, GRS00,
RSSG02]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Definition of a nested constraint-based model and query language for the management of spatial and spatio-temporal data.

C2. Vector or Raster-Based: Spatial representation is based on linear constraints.

P2.1 Point, Pure Line, Pure Surface: Only one spatial data type is supported.

P2.3 Set Theoretically Closed Spatial Objects: The use of “=” and “≤” in linear constraints disallows open sets.

P2.4 Empty Set not a Valid Spatial Object: After an operation, tuples containing an empty linear constraint relation are discarded.

P5. Non-Connected Spatial Objects not Required: Spatial union of two spatial objects may yield a non-connected spatial object. The same observation applies to spatial difference and spatial intersection.

P6. Complex Data Structures not Required: Spatial data management is based on the complexity of the spatial objects, represented by linear constraint relations.

P9. Functionality of Spatial Operations: They are all supported. The only difference is that non-connected spatial objects may be produced.

7.2.19 CALG

References: [KRSS98]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Definition of a nested constraint-based model and query language for the management of spatial and spatio-temporal data.

The behaviour of this approach is very close to that of the previous, hence only main differences are given.

P2.4 Empty Set not a Valid Spatial Object: The empty set is a valid linear constraint relation in this approach.

7.2.20 van Roessel's Conceptual Model (RCM)

References: [Ro93, Ro94]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Defining a conceptual model to support the *Overlay* operation in both raster and vector implementations.

C2. Vector or Raster-Based: Spatial objects are sets of R^2 points.

P2.1(b), P2.1(c) Pure Line and Pure Surface: Set of points includes both pure lines and surfaces.

P2.3 Set Theoretically Closed Spatial Objects: Sets of points can be opened.

P6. Complex Data Structures not Required: Complex data structures are required for the definition of operation *Fold*.

P9. Functionality of Spatial Operations: They are all indirectly supported, based on *Fold* and *Unfold*. However, non-closed spatial objects may be obtained in the result. This property disallows the support of operation *Boundary*.

7.2.21 Scholl and Voisard's Thematic Map Model (SVTMM)

References: [SV89]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Defining a complex object model for the manipulation of maps containing 2-d spatial objects.

C2. Vector or Raster-Based: Spatial objects are subsets of R^2 .

P2.1 Point, Pure Line and Pure Surface: Only one data type of subsets of R^2 is supported.

P2.3 Set Theoretically Closed Spatial Objects: Open subsets of R^2 are supported.

P5. Non-Connected Spatial Objects not Required: Spatial union may yield non-connected spatial objects. Same applies to spatial difference and to spatial intersection.

P6. Complex Data Structures not Required: The support of operation *Fold* is based *Nest* operation for nested models.

P9. Functionality of Spatial Operations: They all are supported. However, non-closed spatial objects may be obtained.

7.2.22 ISO SQL Multimedia Standard: Spatial (ISOSQLMM)

References: [ME01, Iso02]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Specification of a Standard set of SQL:1999 object types for the manipulation of 2-d spatial objects.

P2.1(b) Pure Line: Pure lines have just one pair of end points.

P2.2 Compatibility between Spatial Data Types: Data type ST_GEOMETRY enables storing spatial values of any spatial data type in the same column.

P3. Support of Hybrid Surfaces: Spatial object HS in Figures 2.12(c-d) is valid.

P6. Complex Data Structures not Required: Spatial data manipulation is based on complex spatial objects.

P9.4, P9.5, P9.7, P9.8, P9.11-P9.13 Quantum and Pair-Wise Union and Except and Left, Right and Full Overlay: The lack of the support of operation *Fold* does not allow achieving the full functionality of these operations.

P9.15 Boundary: The boundary of a pure line is a set of points and the boundary of a point is the empty set.

Further Observations: The boundary of a point is the empty set whereas that of a pure line is a set of points. Note however that in mathematics the boundary of a point is the point itself and the boundary of a pure line is the pure line itself.

7.2.23 OpenGis Simple Features Specification for SQL (OGISSQL)

References: [Ogis99, Ogis01a]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Specification of a standard spatial SQL extension.

The functionality is identical with that of the previous approach.

7.2.24 Oracle8i Spatial Cartridge (ORACLE)

References: [ORA00]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Developing a spatial extension of the object relational DBMS ORACLE.

P2.1 Point, Pure Line, Pure Surface: Only one spatial data type, SDO_GEOMETRY, is supported.

P2.4 Empty Set not a Valid Spatial Object: The empty set is not a valid spatial object, but if the result of some function is the empty set the null value is returned.

P3. Support of Hybrid Surfaces: Spatial objects HS in Figures 2.12(c-d) valid.

P6. Complex Data Structures not Required: Although the model supports complex data structures, the manipulation of spatial data is based on the complexity of spatial objects.

P9. Functionality of Spatial Operations: The functionality of the operations is close to that of the approach in Subsection 7.2.22. The difference is that function *boundary* is not supported here.

7.2.25 IBM Informix Spatial DataBlade (INFORMIX)

References: [Inf01]

Classification-Evaluation Tables:
Figures 7.1(c-d)

The functionality of this system is very close to that of the approach in Subsection 7.2.22, hence only main differences are addressed below.

Objective: Developing a spatial extension of the object relational DBMS INFORMIX.

P8. No Data Loss in Operations: The spatial intersection of a line with a surface yields only lines. A similar observation also applies to other data types, for operations spatial union and spatial difference.

P9.2 Fold: Aggregate function *st_dissolve* cannot be applied to spatial objects of any data type and non-connected spatial objects may appear in the result.

P9.4 - P9.13 Quantum, Pair-Wise and Overlay Operation: They are supported partly, due to the partial support of operation *Fold* and of the partial support of spatial union, difference and intersection. However, two limitations are identified: (i) Data loss, which has already been identified above. (ii) Spatial union and spatial difference have to be applied to spatial objects of the same data type. In addition, non-connected spatial objects may appear in the result.

7.2.26 IBM DB2 Spatial Extender (DB2)

References: [Ibm01]

Classification-Evaluation Tables:
Figures 7.1(c-d)

The functionality is very close to that of previous, hence only main differences are addressed.

Objective: Developing a spatial extension of the object relational DBMS DB2.

P9.2, P9.11-P9.13 Fold, Left, Right and Full Overlay: Operation *st_dissolve* is not provided in DB2.

7.2.27 PostgreSQL

References: [Post01, SR86, RS87,
SRH90]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Developing an open-Source Object Relational DBMS with spatial capabilities.

P2.1(b) Pure Line: Pure lines have just one pair of end points.

P2.1(c) Pure Surface: Only pure surfaces without holes are supported.

P6. Complex Data Structures not Required: Complex object relational data structures are required to support the manipulation of complex non-connected spatial objects.

P8. No Data Loss in Operations: The intersection of two rectangles is either the empty set or one rectangle.

P9.6, P9.9, P9.10 Quantum and Pair-Wise Intersect and Inner Overlay: Only the spatial intersection of either two line segments or of two rectangles is directly supported.

P9.15 Boundary: It is supported only for pure surfaces without holes.

7.2.28 GEO++

References: [VO92]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Developing a GIS on top of the Object Relational DBMS Postgres.

The classification and evaluation matches that of PostgreSQL (Subsection 7.2.27).

7.2.29 GEUS

References: [PLL+98]

Classification-Evaluation Tables:
Figures 7.1(c-d)

Objective: Extending a commercial object relational DBMS with new spatial data types, operations and spatial indexes (R*-tree).

The classification and evaluation is very close to that of the previous approach. Only one difference is described below.

P9. Functionality of Spatial Operations: Functions for the computation of spatial union, spatial difference, spatial intersection and boundary are not provided.

7.2.30 Object-Oriented Geographic Data Model (OOGDM)

References: [Vo97, BVH96, DBVH97, VBH97] *Classification-Evaluation Tables:*
 Figures 7.1(c-d)

Objective: Design and implementation of an object-oriented database kernel for spatio-temporal data management.

C1. Discrete or Continuous Change in Space: Both supported by vector and raster data types, respectively.

P2.1(b) Pure Line: Pure lines have just one pair of end points.

P3. Support of Hybrid Surfaces: Spatial object HS in Figures 2.12(c-d) is valid.

P6. Complex Data Structures not Required: The formalization of spatial operations is based on the definition of complex spatial types, *features sets*.

P8. No Data Loss in Operations: If the spatial intersection of two pure surfaces contains pure surfaces, pure lines and points, only the pure surfaces can be obtained. The same observation applies to the spatial intersection of two pure lines.

P9.6, P9.9, P9.10 Quantum and Pair-Wise Intersect and Inner Overlay: Three different methods are defined to compute the spatial intersection of spatial objects. Data loss has been identified in Property P8.

P9.15 Boundary: The boundary of a point is the empty set and the boundary of a pure line is the set of its end points.

Further Observations: The boundary of a point is the empty set whereas that of a pure line is a set of points. Note however that in mathematics, the boundary of a point is the point itself and the boundary of a pure line is the pure line itself.

7.3 Classification and Evaluation of Spatio-temporal Approaches

Again, short explanations are given, wherever this is estimated to be necessary, in the classification and evaluation of each model

	G+STM	MRASTM	WSTM	OPTM	KKTGIS	STSQ	SQLST	GEODETIC	ORPARADB
Model Classification									
C1	D	D	D	D	D	D	D	D	D
C2	V/R	?	V	N/A	V	?	V	V	?
C3	D	D	D	?	G	D	D	D	D
Model Evaluation									
P1	Formalism for Spatial Types and Operations								
P1(a)	Y	N	N	Y	N	N	Y	N/A	N
P1(b)	Y	N	N	Y	N	N	Y	N/A	N
P2	Spatial Data Types Matching Human Perception								
P2.1(a)	Y	?	N	N	Y	?	Y	Y	N
P2.1(b)	N	?	N	Y	P	?	N	P	N
P2.1(c)	N	?	N	Y	Y	?	N	Y	N
P2.2	N	?	Y	N	N	?	Y	Y	Y
P2.3	Y	?	Y	Y	Y	?	Y	Y	?
P2.4	N	?	N	Y	Y	?	Y	Y	E
P3	N	?	Y	N	N	?	Y	N	?
P4	N	?	N	Y	Y	?	N	Y	N
P5	N	?	N	N/A	Y	Y	Y	N/A	N
P6	Y	?	Y	N/A	N	N	Y	N/A	Y
P7	Y	Y	Y	N/A	N	P	Y	Y	Y
P8	N	N/A	Y	N/A	N/A	Y	Y	N/A	Y

Figure 7.2(a): Classification and evaluation of spatio-temporal approaches.

Model Evaluation		G+STM	MRASTM	WSTM	OPTM	KKTGIS	STSQL	SQLST	GEODETIC	ORPARADB
P9	Functionality of Operations for Space	N/A	N/A	N/A	N/A	Y	Y	Y	Y	P
P9.1	<i>Basic-Relational</i>	P	N	N	P	N	Y	P	N	P
P9.2	<i>Fold</i>									
P9.3	<i>Unfold</i>									
P9.4	<i>Quantum Union</i>	P	N	P	N	N	P	N	N	Y
P9.5	<i>Quantum Except</i>	P	N	P	N	N	P	N	N	Y
P9.6	<i>Quantum Intersect</i>	Y	N	P	N	N	P	Y	N	Y
P9.7	<i>Pair-Wise Union</i>	P	N	P	N	N	P	N	N	Y
P9.8	<i>Pair-Wise Except</i>	P	N	P	N	N	P	N	N	Y
P9.9	<i>Pair-Wise Intersect</i>	Y	N	P	N	N	P	Y	N	Y
P9.10	<i>Inner Overlay</i>	Y	N	P	N	N	P	Y	N	Y
P9.11	<i>Left Overlay</i>	P	N	N	N	N	P	Y	N	Y
P9.12	<i>Right Overlay</i>	P	N	N	N	N	P	Y	N	Y
P9.13	<i>Full Overlay</i>	P	N	N	N	N	Y	Y	N	Y
P9.14	<i>Complementation</i>	N	N	N	N	N	N	N	N	N
P9.15	<i>Boundary</i>	Y	N	?	N	N	N	N	N	N
P9.16	<i>Envelope</i>	Y	N	N	N	N	N	N	N	N
P9.17	<i>Buffer</i>	N	N	N	N	N	N	Y	N	N
P10	Limited Number of Kernel Operations	N	N	N	N	N	Y	N	N	Y
P11	Dimension Independent Spatial Types and Operations	N	N	N	E	N	E	N	N	E
P12	Implementation Available	Y	N	E	N	Y	N	N	Y	N

Figure 7.2(b): Classification and evaluation of spatio-temporal approaches.

	G+STM	MRASTM	WSTM	ESSTPM	OPTM	THTGIS	KKTGIS	STSQL	SQLST	DEDALE	YCSTM	GEODETIC	ORPARADB	TOOGDM	TRIPOD	MOST
Model Classification																
C1	C	C	D	C	D	D	D	D	D	C	C	D	D	D	D	C
C2	V	V	B	V	V	B	B	B	V	V	?	V	V	B	B	V
C3	A	A	A	T	T	T	T	T	T	A	A	A	A	TA	TA	A
Model Evaluation																
P1	Formalism for Time Types and Operations															
P1(a)	Y	N	N	Y	Y	N	N	N	Y	Y	N	N/A	N	N	Y	N
P1(b)	Y	N	P	Y	Y	N	N	N	Y	Y	N	N/A	N	N	Y	N
P2	Time Types Matching Human Perception															
P2.1	N	N	N	P	P	P	P	N	P	N	N	N	N	Y	Y	N
P2.2	N	?	N	Y	Y	Y	Y	?	N/A	Y	Y	Y	Y	Y	N	Y
P3	N	?	N	N	N	Y	Y	E	Y	Y	N	N	E	N	N	N
P4	N	N	N	N/A	N/A	N/A	N/A	Y	Y	Y	Y	N/A	N	N/A	Y	N/A
P5	Y	Y	Y	N/A	N/A	N/A	N/A	N	Y	Y	N	N/A	Y	N/A	N	N/A
P6	Y	?	N	N	N	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y
P7	Y	Y	Y	N/A	N/A	N	N	P	N	Y	Y	Y	Y	Y	Y	Y
P8	Y	Y	Y	N/A	N/A	Y	Y	Y	Y	Y	N	Y	N	Y	Y	Y
P9	Y	N	Y	Y	Y	N	N	Y	Y	Y	N	N	Y	N	Y	N
P10	N	N	N	Y	N	N/A	N/A	Y	Y	Y	Y	N	Y	N	N	N/A
P11	P	N	P	N	N	N	N	N	N	Y	?	Y	N	Y	N	Y

Figure 7.2(c): Classification and evaluation of spatio-temporal approaches.

7.3.1 Güting et al Spatio-temporal Model (G+STM)

References: [GBE+00, FGNS00, CFG01]

Classification-Evaluation Tables: Figures 7.2(a-c)

Objective: Formalization of a set of data types and operations for the management of moving objects.

Classification and Evaluation With Respect to Spatial Properties

The behaviour with respect to the spatial properties is almost identical to that of the ROSE algebra, evaluated in Subsection 7.2.1. Only differences are given below.

C2. Vector or Raster-Based: Spatial representation at conceptual level.

P2.1(a) Point: It is supported.

Classification and Evaluation With Respect to Spatio-temporal Properties

P3. Tuple or Attribute Level of Time Recording: Time and space integrated in the definition of spatio-temporal data types.

P2.1 User-friendly Time Data Types: Non-connected time periods are valid.

P3. Support of Various Granularities of Time: Only one domain for time, isomorphic to the set of real numbers.

P4. Spatio-temporal Data Types Not Required: Data types defined for moving objects.

P9. Evolution with Respect to Time of Spatial Operations: Supported by the functionality of *Lifting* spatial operations.

P11. Implementation Available: Partial results are discussed in [FGNS00, CFG01].

7.3.2 Moreira, Ribeiro and Abdessalem's Spatio-temporal Model (MRASTM)

References: [MRA00, MSR99]

Classification-Evaluation Tables: Figures 7.2(a-c)

Objective: Classification of the set of operations required for the manipulation of moving objects.

In general, no formalism is provided.

7.3.3 Worboys's Spatio-temporal Model (WSTM)

References: [Wo94]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Definition of a data model for the representation and manipulation of spatial objects that change discretely with respect to time.

Classification and Evaluation With Respect to Spatial Properties

P2.1(a), P2.1(b), P2.1(c) Point, Pure Line, Pure Surface: Only one spatial data type is supported.

P2.4 Empty Set not a Valid Spatial Object: Not supported as this is implied by the definition of a spatial object as a collection of points, straight line segments and triangles.

P5. Non-Connected Spatial Objects not Required: Spatial union of two spatial objects may yield a non-connected spatial object.

P9.2 Fold: It is not defined.

P9.4 Quantum and Pair-Wise Operations: In spite of the support of functions for spatial union, difference and intersection, the lack of operation *Fold* disallows achieving the full functionality of these operations.

P9.10 Inner Overlay: Same observation as above.

P9.11 Left, Right and Full Overlay: It is not possible to subtract from a spatial object the set of spatial objects stored in one relation. Therefore, these operations are not supported.

P12. Implementation Available: Reported as part of current work.

Classification and Evaluation With Respect to Spatio-temporal Properties

The approach is very close to that in Subsection 7.3.1, therefore, only main differences are discussed below.

P1. Formalism for Time Types and Operations: Only the spatio-temporal operations are formalized.

P6. Generic Support of Temporal Data: Only spatio-temporal formalism.

P11. Implementation Available: Reported as part of current work.

7.3.4 Erwig and Schneider's Spatio-temporal Partition Model (ESSTPM)

References: [ES99]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Formalism for *spatio-temporal partitions* (temporal thematic maps) and operations.

Classification and Evaluation With Respect to Spatial Properties

See Subsection 7.2.3.

Classification and Evaluation With Respect to Spatio-temporal Properties

C3. Tuple or Attribute Level of Time Recording: Mapping from a time domain to the set of spatial partitions.

P2. Time Types Matching Human Perception: Only time instants.

P3. Support of Various Granularities of Time: One domain for time is supported, isomorphic to the set of reals.

P6. Generic Support of Temporal Data: Only evolution of spatial partitions is covered.

7.3.5 d'Onofrio and Pourabbas's Temporal Map Model (OPTMM)

References: [OP01]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Formalism for *temporal thematic maps* and operations between them.

The functionality is very close to that of the previous approach, therefore, only major differences are discussed below.

Classification and Evaluation With Respect to Spatial Properties

P2.1(b) Pure Line: Thematic maps of lines are also supported.

P2.2 Compatibility between Spatial Data Types: Lines and surfaces may not coexist in the same map.

P8. No Data Loss in Operations: Only *Fold* is supported.

P9. Functionality of Spatial Operations: Only *Fold* is supported when applied to either surfaces or lines.

Classification and Evaluation With Respect to Spatio-temporal Properties

C1. Discrete or Continuous Change in Time: Discrete changes are considered.

7.3.6 Tryfona and Hadzilacos's Temporal GIS Approach (THTGIS)

References: [TH98]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Generalization of a GIS approach, so as to apply to spatio-temporal data.

Classification and Evaluation With Respect to Spatial Properties

See Subsection 7.2.4.

Classification and Evaluation With Respect to Spatio-temporal Properties

P2.1 User Friendly Time Data Types: Only period of time is supported.

P4. Spatio-temporal Data Types not Required: The evolution of spatial operations is not supported.

P7. No Limitations on Data Structures: A relation may have at most one valid time attribute and one transaction time attribute.

7.3.7 Kemp and Kowalczyk's Temporal GIS Approach (KKTGIS)

References: [KK94]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Design and implementation of a temporal GIS on top of the object-relational DBMS.

The functionality is very close to that of the previous approach, therefore, only major differences are addressed below.

Classification and Evaluation With Respect to Spatial Properties

P2.1(b-c) Pure Line: Pure lines restrict to straight line segments.

P2.2 Compatibility between Spatial Data Types: A combination of points, lines and surfaces is not a valid spatial object.

P8. No Data Loss in Operations: Operations, relevant to the work in the present thesis, are not provided.

P9. Functionality of Spatial Operations: Operations, relevant to the work in the present thesis, are not provided.

Classification and Evaluation With Respect to Spatio-temporal Properties

Identical to the previous approach.

7.3.8 STSQL

References: [BJS98, BJ96]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: SQL extension for the management of spatio-temporal data.

Classification and Evaluation With Respect to Spatial Properties

The approach is very close to that in Subsection 7.2.20.

P1. Formalism for Spatial Types and Operations: Spatial data representation and operations are not formalized.

P6. Complex Data Structures not Required: Spatial explicit and implicit attributes are considered.

P7. No Limitations on Data Structures: Spatial explicit and implicit attributes are considered.

Classification and Evaluation With Respect to Spatio-temporal Properties

P2.1 User-friendly Time Data Types: Only one type, PERIOD, is provided.

P5. Complex Data Structures not Required: Implicit and explicit attributes are considered.

P7. No Limitations on Data Structures: Time data types are implicit.

P9. Evolution with Respect to Time of Spatial Operations: Supported by explicit *expanding* of tuples by the use of the keyword REDUCIBLE.

7.3.9 SQLST

References: [CZ00, CZ99]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: SQL extension for the management of spatio-temporal data.

Classification and Evaluation With Respect to Spatial Properties

P2.1(b-c) Pure Line, Pure Surface: Data type LINE also includes points, and data type REGION also includes lines and points.

P9. Functionality of Spatial Operations: Relevant to the spatial functionality proposed in the present thesis, only one spatial function is provided, *intersection*. In conjunction with *Fold* that that is involved in insertions, it enables achieving the functionality of *Quantum Intersection*, *Pair-Wise Intersection* and *Inner Overlay*.

Classification and Evaluation With Respect to Spatio-temporal Properties

P2.1 User-friendly Time Data Types: Only the time instant data type is supported.

P2.2 Empty set not a Valid Time Period: Time period data type is not defined.

P7. No Limitations on Data Structures: At most one valid time attribute is allowed in a relation.

P9. Evolution with Respect to Time of Spatial Operations: It is supported by the internal use of automatic expanding of tuples.

7.3.10 Dedale

References: [GRS98b, GRS00]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Formalism for a spatio-temporal model, based on constraints.

Classification and Evaluation With Respect to Spatial Properties

Given in Subsection 7.2.18.

Classification and Evaluation With Respect to Spatio-temporal Properties

P2.1 User-friendly Time Data Types: Only one data type, *set of time instants*, is supported.

P2.2 Empty set not a Valid Time Period: Tuples containing empty linear constraint relations are automatically discarded when produced by some operation.

P4. Spatio-temporal Data Types not Required: At the user's level, spatio-temporal data appears in a relation-valued attribute $R(x, y, t)$, where x and y are spatial coordinates and t is the time coordinate.

7.3.11 Yeh and de Cambray's Spatio-temporal Model (YCSTM)

References: [YC95, YC94b]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Presentation of a data model to manage highly variable spatio-temporal data.

Classification and Evaluation With Respect to Spatial Properties

See Subsection 7.2.10.

Classification and Evaluation With Respect to Spatio-temporal Properties

C2. Only Valid Time (V) or Bitemporal (BT): It is not clear.

P2.1 User-friendly Time Data Types: Only one period data type is supported.

P3. Support of Various Granularities of Time: Only one domain for time is supported, isomorphic to the set of real numbers.

P5. Complex Data Structures not Required: A spatio-temporal object is recorded as a set of tuples in a complex object model.

P8. No Need to Redefine Conventional Operations: Operations *Union*, *Except* and *Intersect* are redefined for the management of temporal data.

P9. Evolution with Respect to Time of Spatial Operations: Automatic expanding is used only for time periods and not for spatial objects.

7.3.12 Informix Geodetic DataBlade (GEODETTIC)

References: [Inf00]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Extending a commercial DBMS by providing spatio-temporal data types and operations.

Classification and Evaluation With Respect to Spatial Properties

The approach is very close to that in Subsection 7.2.25. Functions for spatial union, difference and intersection are not supported.

P9. Functionality of Spatial Operations: They are not supported.

Classification and Evaluation With Respect to Spatio-temporal Properties

P2.1 User-friendly Time Data Types: Only one period data type is supported.

P3. Support of Various Granularities of Time: Time ranges are defined only for the DATE type.

P6. Generic Support of Temporal Data: Spatio-temporal specific datablade.

7.3.13 ORParaDB

References: [CG94, CGN93]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Definition of an object-relational data model and pattern matching query language for the management of spatial data.

Classification and Evaluation With Respect to Spatial Properties

C2. Vector or Raster-Based: Spatial representation is not specified.

P2.1(a-c) Point, Pure Line, Pure Surface: Only one spatial domain.

P2.4 Empty Set not a Valid Spatial Object: Spatial values are closed under operations spatial union, spatial difference and spatial intersection. However, it is estimated that tuples with a spatial object that matches the empty set are not recorded.

P9.1 Basic Relational: The functionality of operations Union, Except and Intersect resemble those of the respective operation Quantum operations.

P9.2, P9.4-P9.13 Fold, Quantum, Pair-wise and Overlay Operations: They are supported, due to the implicit use of *Fold*.

P9.14, P9.15 Complementation and Boundary: Not defined.

P11. Dimension Independent Spatial Types and operations: It is estimated that a generalization to n-d is not difficult.

Note that one additional limitation in [CG94] is that conventional algebra operations have been redefined for the management of spatial data.

Classification and Evaluation With Respect to Spatio-temporal Properties

P2.1 User-friendly Time Data Types: Only one data type is supported whose elements are sets of time instants.

P4. Spatio-temporal Data Types not Required: Operations set union, set difference and set intersection are applied to spatio-temporal objects.

P8. No Need to Redefine Conventional Operations: Operations *Union*, *Except* and *Intersect* have been redefined for the management of temporal data.

P9. Evolution with Respect to Time of Spatial Operations: It is supported due to the automatic expanding.

7.3.14 Temporal Object-Oriented Geographic Data Model (TOOGDM)

References: [Vo97, BVH96]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Design and implementation of an object-oriented database kernel for spatio-temporal data management.

Classification and Evaluation With Respect to Spatial Properties

See Subsection 7.2.30.

Classification and Evaluation With Respect to Spatio-temporal Properties

C3. Tuple or Attribute Level of Time Recording: Valid time is used at the attribute level and transaction time is used at the tuple level.

Attribute level for valid time and tuple level for transaction time.

P3. Support of Various Granularities of Time: Only one domain for time is considered.

P9. Evolution with Respect to Time of Spatial Operations: Only selection is supported, and it incorporates temporal predicates.

7.3.15 Tripod

References: [GFDP01, GFP+01a,
GFP+01b]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Definition of data types and operations for the management of spatio-temporal data in an Object-oriented model.

Classification and Evaluation With Respect to Spatial Properties

See Subsection 7.2.1.

Classification and Evaluation With Respect to Spatio-temporal Properties

C3. Tuple or Attribute Level of Time Recording: Both levels are allowed.

P2.2 Empty set not a Valid Time Period: The intersection of two time periods may be the empty set.

P3. Support of Various Granularities of Time: Only one time domain is supported.

P5. Complex Data Structures Not Required: The management of transaction time is based on the manipulation of complex structures, called *Histories*.

7.3.16 MOST

References: [SWCD97, WXCJ98]

Classification-Evaluation Tables:
Figures 7.2(a-c)

Objective: Defining a data model for the management of the future evolution of spatial objects.

Classification and Evaluation With Respect to Spatial Properties

Spatial data types and operations are not defined.

Classification and Evaluation With Respect to Spatio-temporal Properties

P2.1 User-friendly Time Data Types: Only time periods are supported which are open to the right up the infinite future.

P3. Support of Various Granularities of Time: Only one domain for time is supported.

P9. Evolution with Respect to Time of Spatial Operations: Only selection is supported, and it incorporates temporal predicates.

7.4 Classification and Evaluation of the Proposed Model

In this section the model formalized in the present thesis is classified according to the characteristics analysed in Sections 2.4 and 2.6 and is evaluated with respect to the properties specified in Section 2.8.

7.4.1 Classification and Evaluation with Respect to Spatial Properties

C1. Discrete Change in Space: The model has been defined for the support of properties that change discretely in space. However, the indication is that it can also be used for the management of data that changes continuously in space [VL03b]. This issue is discussed further in Section 8.3.5.

C2. Raster-like Representation of Space: In terms of the formalization undertaken, the proposed model fits better the *raster-based* representation (Section 3.2). Note however that beyond surfaces, which are defined in such approaches, the proposed model also considers additional spatial objects such as lines and hybrid surfaces. Note also that the formalism undertaken does not necessarily imply that it is only a raster-based implementation that has to be undertaken. This issue is discussed in more detail in Section 8.3.2

C3. Database-centric Spatial Data Management: This is also obvious from the formalization followed (Chapter 4).

P1. Formalism for Spatial Types and Operations: This is true for both data types and operations (Chapters 3 and 4).

P2. Spatial Data Types Matching Human Perception: It is satisfied in that *Point*, *Line* and *Surface* are defined as primitive spatial types and they are *spatially* compatible. Also, all the spatial objects are set-theoretically closed and the empty set is not included in them (Section 3.3).

P3. Support of Hybrid Surfaces: Satisfied (Section 3.3).

P4. Spatial Data Validation Mechanisms: They are inherent in the definition of the spatial data types (Section 3.3).

P5. Non-Connected Spatial Objects not Required: Satisfied (Section 3.3).

P6. Complex Data Structures not Required: The model considers only non-nested relations (Section 4.2).

P7. No Limitations on Data Structures: The scheme of a relation may have arbitrarily many spatial attributes (Section 4.2).

P8. No Data Loss in Operations: It is also satisfied from the way the operations have been defined. This has also been demonstrated by examples (Section 4.3).

P9. Full Functionality of all Spatial Operations: It is satisfied because all the operations can be applied to all types of spatial data, point, line, surface and combinations of them, as has also been shown in Section 4.3.

P10. Limited Number of Kernel Operations: Indeed, the kernel consists of the operations of the non-nested relational model plus operations *Unfold* and *Fold*. Note that these operations had been defined in previous work on temporal databases [LM97]. Within this thesis it was only necessary to define how they can be applied to spatial data. All the remainder operations have been defined in terms the kernel operations. Definitely, more non-kernel operations can also be defined (Section 4.3).

P11. Dimension Independent Spatial Types and Operations: The thesis is restricted to the management of only 2-d spatial data. However the management of n-d data is straightforward. It suffices to consider n-d spatial data types (Section 3.3) and specify the functionality of *Unfold* and *Fold* (Subsection 4.3.2): Both of them are simple. As an example, a 3-d space requires defining 3-d quanta, i.e. 3-d quantum points, lines, surfaces and cubes. The application of *Unfold* to 3-d objects is trivial. Consequently, it is argued that this property is also satisfied.

P12. Implementation Available: Initially, it has to be taken into consideration that an implementation of the proposed model was beyond the objectives of this thesis. However, it is argued that its implementation is possible. It is noted in particular that, beyond the conventional operations of the relational model, all the others, defined in this thesis, can be expressed in terms of operations *Unfold* and *Fold*. Predicate *conductive* is also used in the definition of some of these operations. For a prototype implementation therefore, it suffices to implement *Unfold*, *Fold* and *conductive*. Indeed, pseudo code, to implement them all, is provided in Appendix C.

7.4.2 Classification and Evaluation with Respect to Spatio-temporal Properties

C1. Discrete Change in Time: The model has been designed to support the modelling of spatial data that changes discretely with respect to time. However, it is estimated that it can also be used in applications where spatial data changes continuously with respect to time. Related issues are addressed in Section 8.3.5.

C2. Valid Time Semantics: The semantics for the time elements in the model are assumed to be those of valid time (Section 5.4). However, transaction time data and bitemporal data can also be handled [LM03] by the same set of operations.

C3. Tuple Level of Time Recording: Time is recorded explicitly at the level of tuple in non-nested relations (Section 5.4).

P1. Formalism for Time Types and Operations: Formalism has been developed for both time types and operations (Section 5.4).

P2. Time Types Matching Human Perception: It is satisfied, in that *time instants* and *time periods* are supported as distinct data types. Besides, time periods are defined as connected. Finally, the empty set is not a valid time period (Section 5.2).

P3. Support of Various Granularities of Time: Various granularities of time are supported as specific instances of the generic types INSTANT and PERIOD (Section 5.2).

P4. Spatio-temporal Data Types not Required: Spatio-temporal operations are defined without the need of defining spatio-temporal data types (Section 5.7). Instead, only spatial and time types have been defined (Sections 3.3 and 5.2).

P5. Complex Data Structures not Required: Only non-nested relations are required for the storage and management of time data (Section 5.4).

P6. Generic Support of Temporal Data: The operations are generic in that they can also be applied to other types of data [LM97].

P7. No Limitations on Data Structures: A relation may have arbitrarily many attributes of some time type (Section 5.4).

P8. No Need to Redefine Conventional Operations: No operations had to be redefined. The full functionality has been achieved by the definition of two more operations *Unfold* and *Fold* (Section 5.4).

P9. Evolution with Respect to Time of Spatial Operations: It is supported, as it has been shown in Section 5.7.

P10. Limited Number of Kernel Operations: As in the case of spatial data, the kernel consists of operations of the non-nested relational model and operations *Unfold* and *Fold* (Section 5.4).

P11. Implementation Available: In the proposed model, spatial objects and time are recorded in distinct attributes. The implementation of the temporal component has already been addressed in previous research. In particular, [LM97] reports on an implementation on a SUN workstation, with SUNOS 4.1.3. One time period data type is supported, DATEINTERVAL, whose format is [yyyy-mm-dd, yyyy-mm-dd). DATEINTERVAL, as well as a series of predicates and functions for periods have been implemented directly within the kernel of INGRES, release 6.4. Optimisation has also been incorporated in the implementation. In particular the operations, which are theoretically defined in terms of *Unfold*, incorporate internally a user-transparent operation, *split* [LPS95, LPS94], which reduces drastically the execution time. Main storage structures are also used, to reduce disk I/O operations [LM94]. Finally, certain *Unfold* or *Fold* operations, explicitly issued by the user, are internally eliminated, if they are determined to be redundant for the derivation of the result relation. Details of this implementation are reported in [VLGM94]. Further optimisation solutions are also proposed in [LM97] and, moreover, an implementation that also supports transaction and concurrency control is reported in [VLG98]. It is concluded therefore that the temporal aspect of the model has been implemented.

7.5 Comparison with Functionality of Other Approaches

In this section an inverse evaluation is undertaken, in that the model formalised in the present thesis is evaluated with respect to the functionality proposed in other approaches.

7.5.1 Predicates and Functions

It is recalled (Section 3.4, Section 3.5) that it was beyond the objective of this thesis to define a *full* set of predicates and functions. Relevant to those defined, however, the following have to be noticed.

- (i) Predicate *has_holes* (Section 3.4) can be applied to spatial objects of any data type, hence it is more general than the one defined in [Inf01, Ibm01, Ogis99, Iso02].
- (ii) None of the predicates defined in other approaches has a functionality equivalent to that of *conductive* and *surrounds*, defined in Section 3.4. It is estimated, however, that these two predicates have practical interest.
- (iii) To the best of this author's knowledge, defining comparison predicates $<$, $<=$, $>$, $>=$ (Section 3.4) between spatial objects has not been defined in any other approach. However, their definition now enables using them uniformly in comparisons between any two elements of the same data type. Moreover, this use has practical interest, as is shown in Section 7.5.2 and in [VL03b].

7.5.2 Relational Algebra Operations

Functionality defined in other approaches, which is not supported in the model proposed in this thesis, includes the aggregate functions *mbr* [Inf01] and *intersection* [SH91, GRS98a, KRSS98] and the operations *simple_path* [CZ96], *distance_along_path* [CF93], *Delaunay triangulation* [VO92], *projection in space* [CZ96, Inf01], *move* [SH91, MO86], *rotate* [SH91], *scale* [SH91] and *convex_hull* [Eg94, VO92, Gu88, Ora00, Inf01, Ibm01, Ogis99, Iso02]. The justification is that the above functionality can be useful to few specialized applications. In this author's opinion, however, a data model must have a high degree of abstraction. As such, it must consist of few, general-purpose operations, independent of some specific application, that enable facing a wide range of requirements. At the other end, it can be noticed that Arc/Info supports too many operations but only few of them are widely used.

Finally, it is noticed that, no matter how many operations can be defined, programming will always be unavoidable.

FUNCTIONALITY	APPROACHES
<i>Holes</i>	CZ96, Vo97
<i>EndPoints</i>	SH91, Inf01, Ibm01, Ogis99, Iso02, Cz96, CF93, PLL+98
<i>Split</i>	CZ96, SH91, Esri00
<i>Nearest Neighbors</i>	Gu88, Ora00, Inf01, RFS88, GS95
<i>Voronoi</i>	Gu88, CZ96, Vo92, SH91, Esri01, Keig02, Gras02, Lo00, Bk01

Figure 7.3: Additional functionality proposed in other approaches.

On the other hand, the operations defined in this thesis are satisfactorily powerful, in that they enable formulating queries that have practical interest. This is illustrated by a number of examples, which are provided in the remainder of this section. The illustration also includes expressions that enable achieving the functionality defined in other approaches, in particular those provided in Figure 7.3. In what follows, therefore, an informal description of some functionality is initially given. Next, it is shown how the same functionality can be expressed by the use of the SQL extension that was defined in Chapter 6. In some cases it is pointed out that the functionality achieved in the present model is more general than that originally defined in some other approach. Note that SQL is used only for ease of presentation, since a set of equivalent relational algebra operations could be lengthy.

Pure Surface Parts: *Get the pure surface parts of a set of spatial objects.*

Input: $R(\mathbf{A}, G)$.

Output: A relation with attributes \mathbf{A}, G .

Description: For each set of tuples $\{(\mathbf{a}, g_i)\}$ in R , the result relation consists of a set of tuples $\{(\mathbf{a}, g_j)\}$, where g_j is one of the elements of the spatial union of all the pure quantum surfaces that are subsets of some g_i . As an example, if R consists of all the spatial objects depicted in Figure 7.4(a) then the result relation consists of all the spatial objects depicted in Figure 7.4(b).

SQL Statement:

```

WITH pure_surface_parts( $\mathbf{A}, G$ ) AS (1)
  (SELECT  $\mathbf{A}, G$  (2)
   FROM (SELECT  $\mathbf{A}, G$  (3)
        FROM  $R$  (4)
        REFORMAT AS UNFOLD  $G$ ) (5)
   WHERE  $is\_pure\_surface(G)$  (6)
   REFORMAT AS FOLD  $G$ ) (7)
SELECT  $\mathbf{A}, G$  (8)
FROM pure_surface_parts (9)

```

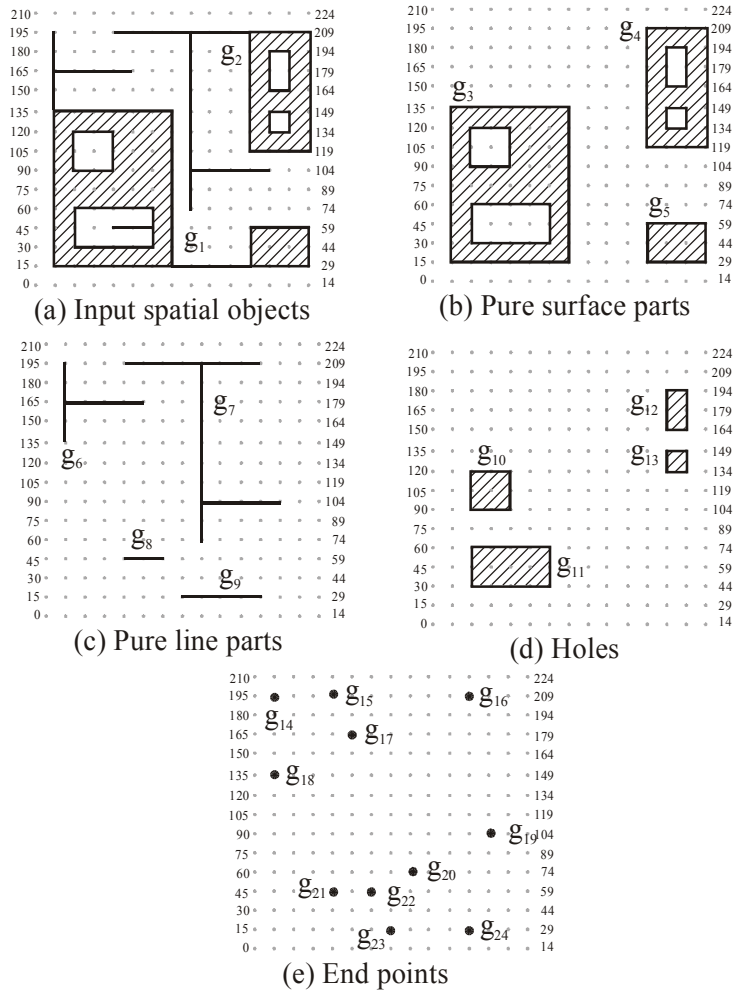


Figure 7.4: Illustration of additional functionality.

Explanation: The sub-query of lines (3)-(5) retrieves all the quanta in R. Next, the WHERE clause of line (6) restricts to the quanta of a PSURFACE type. Finally, line (7) folds on attribute G to obtain the final result. The use of the WITH clause simplifies the formulation of subsequent statements that follow below.

Remark: The above functionality is generic, in that it can be applied to a relation that contains any type of spatial data.

Pure Line Parts: *Get the pure line parts of a set of spatial objects.*

Input: R(A, G).

Output: A relation with attributes **A**, **G**.

Description: For each set of tuples $\{(a, g_i)\}$ in **R**, the result relation consists of a set of tuples $\{(a, g_j)\}$, where g_j is one of the elements of the spatial union of all the pure quantum lines that are subsets of some g_i and not contained in some pure quantum surface contained in some g_i . As an example, if **R** consists of all the spatial objects depicted in Figure 7.4(a) then the result relation consists of all the spatial objects depicted in Figure 7.4(c).

SQL Statement:

```

SELECT A, G                (1)
FROM (SELECT A, G          (2)
      FROM R                (3)
      REFORMAT AS UNFOLD G) (4)
WHERE is_pure_line(G)      (5)
EXCEPT EXPANDING (G)    (6)
SELECT A, G                (7)
FROM pure_surface_parts    (8)

```

Explanation: Similarly as in the previous query, lines (1)-(5) retrieve all the (a, q_k) where q_k are pure line quanta. Finally, the *QExcept* operation of line (6) subtracts the pure surface parts retrieved in lines (7)-(8) (see previous query for derivation of pure-surface-parts) from the quantum lines retrieved in lines (1)-(5).

Remark: The previous functionality is generic in that it can be applied to a relation that contains any type of spatial data.

Holes: *Get the holes of a set of pure surfaces.*

Input: **R(A, G)**.

Output: A relation with attributes **A**, **G**.

Description: For each set of tuples $\{(a, g_i)\}$ in **R**, the result relation has one or more tuples of the form (a, g_j) where g_j is a hole of some spatial object in the spatial union of $\{g_i\}$. As an example, if **R** consists of all the spatial objects depicted in Figure 7.4(b) then the result relation consists of all the spatial objects depicted in Figure 7.4(d).

SQL statement:

```

SELECT A, G                (1)
FROM pure_surface_parts    (2)
  ENVELOPE OF (G)          (3)
EXCEPT EXPANDING (G)    (4)
SELECT A, G                (5)
FROM R                     (6)

```

Explanation: In line (4), the spatial objects in R, which are retrieved in lines (5), (6) are subtracted from their spatial envelope, which is retrieved in lines (1)-(3). The expression, which retrieves the pure surface parts of the spatial objects in R, was given earlier.

Remark: The functionality is more general than that in [CZ96, Vo97] because it can be applied to spatial objects of any type.

End Points: Give the end points of a set of pure lines.

Input: R(A, G)

Output: A relation with attributes A, G.

Description: For each set of tuples $\{(a, g_i)\}$ in R, the result relation has one or more tuples of the form (a, g_j) where g_j is an end point of some pure line in the spatial union of $\{g_i\}$. As an example, if R consists of all the spatial objects depicted in Figure 7.4(c) then the result relation consists of all the spatial objects depicted in Figure 7.4(e).

SQL statement:

```

WITH UR(A, G) AS (1)
  (SELECT A, G (2)
   FROM R (3)
   REFORMAT AS UNFOLD G) (4)

SELECT UR1.A, UR1.G (5)
FROM UR AS UR1 (6)
WHERE is_point(G) AND (7)
      1 = (SELECT count(*) (8)
          FROM UR AS UR2 (9)
          WHERE is_pure_line(UR2.G) AND (10)
                UR2.A = UR1.A AND (11)
                UR2.G cp UR1.G) (12)

```

Explanation: The query of lines (2)-(4) retrieve all the quanta in R. Next, the conditions in lines (7)-(12) restricts to those quanta of type POINT that intersect with just one quantum line.

Remark: The above expression is general in that it can be applied to any kind of pure lines. Contrary to this, only simple pure lines can be considered in [SH91, Inf01, Ibm01, Ogis99, Iso02, CZ96, CF93, PLL+98].

Split: Split each set of spatial objects recorded in a relation $R_1(\mathbf{A}, G)$ with respect to each set of objects recorded in a relation $R_2(\mathbf{B}, G)$.

Input: $R_1(\mathbf{A}, G)$, $R_2(\mathbf{B}, G)$

Output: A relation with attributes A, B, G.

Description: For each set of tuples $\{(\mathbf{a}, g_i)\}$ in R_1 and for each set of tuples $\{(\mathbf{b}, g_j)\}$ in R_2 , the result relation contains a set of tuples $\{(\mathbf{a}, \mathbf{b}, g_k)\}$, where for each pair of quanta q_1, q_2 in some g_k , which is contained in some g_i , and for each g_j , the predicate $conductive(q_1, q_2, g_i, g_j)$ evaluates to true. As an example, if R_1 consists of all the objects in Figure 7.5(a) that are subscripted by A_i and R_2 consists of all the objects in Figure 7.5(a) that are subscripted by B_j , then the result relation contains all the spatial objects depicted in Figure 7.5(b).

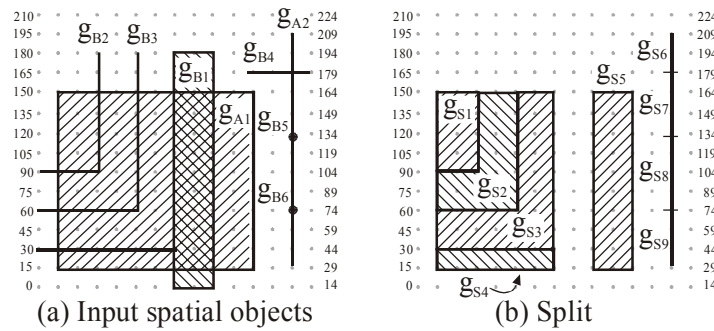


Figure 7.5: Illustration of the functionality of *Split*.

SQL statement:

```

WITH R(A, B, G1, G2) AS (1)
  (SELECT TR.A, TR.B, TR.G1, TR.G2 (2)
   FROM (3)
    (SELECT A, B, R1.G AS G, R1.G AS G1, R1.G AS G2 (4)
     FROM R1, R2 (5)
     REFORMAT AS UNFOLD G1, G2) AS TR (6)
   WHERE NOT EXISTS (7)
    (SELECT R2.G (8)
     FROM R2 (9)
     WHERE NOT conductive(TR.G1, TR.G2, TR.G, R2.G) (10)
      AND TR.B = R2.B) (11)
    REFORMAT AS FOLD G1, G2) (12)
  SELECT A, B, G1 as G (13)
  FROM R (14)

```

Explanation: The sub-query of lines (4)-(6) retrieves a relation with attributes $A, B, G, G1, G2$, containing tuples of the form $(\mathbf{a}, \mathbf{b}, g, q_i, q_j)$, where (\mathbf{a}, g) belongs to R_1 , the pairs (q_i, q_j) yield the combinations of every quantum in g with all the quanta in g and there exists some (\mathbf{b}, g_r) in R_2 . The condition expressed in lines (7)-(11) restricts to those tuples $(\mathbf{a}, \mathbf{b}, g, q_i, q_j)$ for which there does not exist some tuple (\mathbf{b}, g_r) in R_2 for which $conductive(q_i, q_j, g, g_r)$ evaluates to false. In line (12) the result of the previous selection is folded on attributes $G1, G2$, in order to retrieve the spatial objects of the result. Finally,

lines (13)-(14) project out attribute G2, since it contains the same data as attribute G1.

Remark: The above statement is general in that it can be applied to any kind of spatial data. Contrary to this, the relevant functionality in [SH91] splits with respect to a pure line either a simple pure line or a pure surface without holes. The relevant functionality in [CZ96] splits either a pure surface with respect to a set of pure lines or a pure line with respect to a set of points.

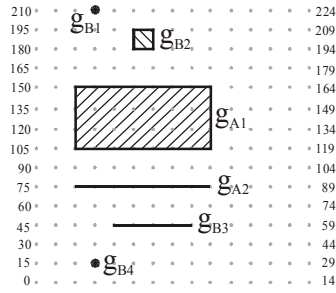


Figure 7.6: Illustration of functionality of *Nearest Neighbours*.

Nearest Neighbours: For each set $\{(a, g_i)\}$ in $R_1(A, G)$ retrieve from $R_2(B, G)$ all the tuples $\{(b, g_j)\}$ for which the distance between g_i and g_j is minimum.

Input: $R_1(A, G), R_2(B, G)$

Output: A relation with attributes **A, B, G₁, G₂**.

Description: For each set of tuples $\{(a, g_i)\}$ in R_1 the result relation contains a set of tuples (a, b, g_i, g_j) such that the distance between g_i and each g_j matches the minimum distance between any pair of g_i, g_r , where (b_r, g_r) belongs to R_2 . As an example, if R_1 consists of the set of tuples $\{(a, g_{A1}), (a, g_{A2})\}$ and R_2 consists of the set of tuples $\{(b_1, g_{B1}), (b_2, g_{B2}), (b_3, g_{B3}), (b_4, g_{B4})\}$, where g_{A_i} and g_{B_j} are depicted in Figure 7.6, then the result relation contains the tuples $\{(a, b_2, g_{A1}, g_{B2}), (a, b_3, g_{A2}, g_{B3})\}$.

SQL statement:

```

SELECT TR1.A, TR2.B, TR1.G AS G1, TR2.G AS G2           (1)
FROM   R1 AS TR1, R2 AS TR2                           (2)
WHERE  distance(TR1.G, TR2.G) =                          (3)
        (SELECT min(distance(TR3.G, TR4.G))                (4)
         FROM   R1 AS TR3, R2 AS TR4                    (5)
         WHERE  TR1.A = TR3.A)                             (6)

```

Explanation: The from clause in line (2) retrieves the Cartesian product of objects in R1 and R2. Then the condition in lines (3)-(6) restricts to those combinations for which the distance between the objects matches the minimum distance between each object in R1 and every object in R2.

Remark: The above statement is general in that it can be executed against any kind of spatial data. Contrary to this, the relevant functionality in [Gu88, RFS88] retrieves the points which are nearest to a given point. Also, the relevant functionality in [GS95, Ora00, Inf01] considers minimum distances only between one spatial object and a set of spatial objects.

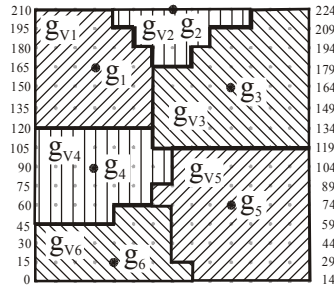


Figure 7.7: Illustration of the functionality of *Voronoi*.

Voronoi: For each spatial point g_i in some tuple of $R_1(\mathbf{A}, G)$, retrieve from $R_2(G)$ the pure surface that consists of all the quantum surfaces which are closest to g_i rather than to any other point g_j in some other tuple of $R_1(\mathbf{A}, G)$.

Input: $R_1(\mathbf{A}, G)$, $R_2(G)$, where the data type of $R_1.G$ is POINT and the data type of $R_2.G$ is PSURFACE.

Output: A relation with attributes \mathbf{A}, G, G_1

Description: For tuple (\mathbf{a}, g_i) in R_1 the result relation contains a tuple (\mathbf{a}, g_{v_i}) where g_{v_i} consists of all those quantum surfaces in some tuple of R_2 that are closer to g_i rather than to any other g_j in R_1 . In case that a quantum surface q has the same distance from both some g_i and some g_j , it is assigned to the minimum of them, according to the total ordering defined in Section 3.4. Note that the pure surfaces in the result relation are either adjacent or disjoint. As an example, if R contains the points g_i in Figure 7.7, then the result relation contains the pure surfaces g_{v_i} in the same figure.

SQL statement:

```

SELECT A, G1 AS G, G2 AS G1                (1)
FROM   (SELECT A, R1.G AS G1, R2.G AS G2   (2)
        FROM R1, R2                        (3)
        REFORMAT AS UNFOLD G2) AS TR      (4)
WHERE  distance(G1, G2) =                 (5)
        (SELECT min(distance(R1.G1, TR.G2)) (6)
         FROM R1) AND                      (7)
G1 = (SELECT min(G)                        (8)
      FROM R1                              (9)
      WHERE distance(TR.G1, TR.G2) =      (10)
            distance(R1.G, TR.G2))      (11)
REFORMAT AS FOLD G2

```

Explanation: The sub-query in lines (2)-(4) retrieves, for each point in R1, all the quantum surfaces in R2. Then, the condition expressed in lines (5)-(7) restricts to those quanta whose distance from the relevant point matches the minimum distance between the same quantum and any point in R1. The condition expressed in lines (8)-(11) disallows assigning one quantum q to more than one point. From all the points with the same distance from quantum q the minimum is chosen.

7.6 Additional Characteristics

Additional characteristics of the proposed model can be summarized as follows:

1. With reference to the management of spatial data, it is estimated that the model integrates fully spatial data within the relational model and a map can be seen as a (base or derived) relation. Hence, the model combines fully the flexibility of *GIS-Centric* approaches, dedicated to the manipulation of maps, with the advantages of database technology. This combination includes the incorporation of relevant optimisation strategies.
2. A simple relational structure, that of non-nested relations, suffices for the representation of spatial, temporal and spatio-temporal data. The management of spatial data actually reduces to the management of relations. Hence, a unique set of operations suffices for the management of all the above types of data. Recall in addition that operations originally defined for the management of spatial data proved to also have practical interest when applied to temporal data, as has been shown in Section 5.3. As has also been shown in [LM97] the same set of operations has practical interest when applied to any other type of data. Hence, all types of data can be represented and be manipulated in a uniform way. To this author's knowledge, no other approach satisfies this property. Moreover, it is estimated that the model is user-friendly because applications, in their majority, consider non-nested relations.
3. It has been shown (Section 7.6) that some operations, defined in other approaches, can also be expressed in the model proposed in this thesis, by the use of its kernel operations. Moreover, it has been shown that the functionality achieved in the proposed model is more general than that of the operations defined in the original approaches. Hence, the estimation is that the model is not open-

ended, in that the set of its kernel operations does not have to be increased.

4. The use of the *Replicate* operation (Subsection 4.3.7) has enabled a further increase of the functionality of the proposed model.
5. Given that there are few kernel operations, the estimation is that optimisation techniques can easily be incorporated.
6. Generally, nested relational structures are more powerful than the relevant non-nested. However, an initial investigation has shown that all the work undertaken in this thesis can straightforwardly be integrated within a nested relational model, the one that has been proposed in [LD98].
7. Finally, initial investigation has shown that the model can also be applied for the management of data related to continuous changes in space and in time, as is briefly discussed in Section 8.3.5.

7.7 Similarities with Other Approaches

Similarities with the data types proposed in this thesis can be identified in the areas of Computer Graphics, Image Processing and Spatial Data Structures. For example, the term *pure quantum surface* of the present model could be associated to what is commonly called *pixel* [Pa82, Sa90c]. However, pixels have been considered only for image processing and spatial data plotting purposes. Contrary to this the various types of spatial quanta of the present work aimed at defining spatial data types and a spatial relational algebra. To this author's knowledge, spatial data types have not been defined in terms of quanta in any other data modelling approach.

Spatial objects that resemble the *pure quantum surfaces* of the present work are incorporated in [Wi98, WF99, WF00]. However, the objective of [Wi98, WF99, WF00] is the development of a *hybrid-raster* representation that incorporates the 9-intersection model [EH92].

As already reported, operations *Unfold* and *Fold* were originally defined for the management of temporal and interval data [LJ88a, LM97]. One first approach, to incorporate them in spatial data modelling, has been reported in [Ro93, Ro94] (see Subsection 2.2.21). However, the characteristics of [Ro93, Ro94], show that the approach differs radically from that in the present thesis: Firstly, five different spatial data types are considered, one of a *point* and four of the form *sets of points*, but the latter have nothing in common with the line and surface types defined in this thesis. Secondly, *Unfold* returns an infinite set of tuples. Thirdly, two types of a *Fold* operation are defined, which return

elements of some *set of points* type. Finally, the description concerns a conceptual model, as the author also admits. The reason is that *Unfold* returns an infinite number of tuples.

The approach undertaken in [GNT91a] (Subsection 2.2.14) considers two operations, *Decompose* and *Compose*, which have similarities with *Unfold* and *Fold*, respectively, defined in this thesis. However, spatial quanta are not considered in [GNT91a] and only one spatial data type is defined, GEOMETRY(S), whose elements are sets of raster cells. Consequently, operation *Decompose* decomposes each *tuple* of a relation into so many tuples as the number of raster cells that are contained in the relevant GEOMETRY(S) value. The second operation performs the inverse, it merges *tuples* with matching values on a given conventional attribute.

Finally, similarities can also be identified with the QLG approach [CZ96, CW96, CN97] (Subsection 2.2.17) but important differences are the following: A nested data model is considered. All the operations in QLG are primitive. Only informal descriptions of the operations are given. Hybrid surfaces are supported only partially (Section 7.2.17). Spatial difference is not defined. Finally, a vector-based representation is considered.

7.8 Conclusions

A classification and evaluation of various spatial and spatio-temporal approaches, as well as of the model defined in this thesis, has been undertaken with respect to the classification and properties specified in Chapter 2. This has not been an easy task, due to the lack of formalism in various approaches. The conclusion is that the proposed model inherits a number of advantages.

It has also been shown that the proposed model supports the majority of the functionality provided in other approaches and, in most cases, it provides a more general functionality. Finally, the model enables the uniform representation and management of any type of data.

CHAPTER 8

SUMMARY AND FUTURE WORK

8.1 Introduction

The findings of the research undertaken in the present thesis are summarized in Section 8.2 and topics for further research are outlined in Section 8.3.

8.2 Summary

In this thesis a data model has been formalised for the management of spatial and spatio-temporal data. Its characteristics can be summarised as follows:

1. It considers discrete change in space and in time.
2. Regarding the management of space, it is closer to raster-based approaches and it is database-centric.
3. Regarding the management of time it considers valid time at the level of tuple.

As opposed to other approaches, the model satisfies the following properties:

1. The definition of spatial types has been based on the prior definition of *spatial quanta*.
2. The spatial types are user-friendly, in that:
 - (i) They match the ordinary *point*, *line* and *surface* that are used in daily practice.
 - (ii) They are *spatially compatible*, in that a line can be seen as a degenerate surface and a point can be seen as either a degenerate surface or a degenerate line. Due to this, spatial objects whose geometry is either a point or a line or a surface can be recorded under the same attribute of a relation.
 - (iii) They are set-theoretically *closed*, in that a line with *missing* points or a surface with either *missing* lines or *missing* points are invalid spatial objects.

- (iv) A *hybrid surface*, a connected spatial object composed of surfaces and lines, is a valid spatial object.
- (v) The empty set is not a valid spatial object.
- 3. Time data types are also user-friendly in that:
 - (i) They consist of *instant* and *period* types.
 - (ii) The empty set is not a valid time period.
- 4. By definition, various granularities for time are supported.
- 5. There was no need to define distinct spatio-temporal data types.
- 6. Spatial, temporal and spatio-temporal data can be recorded and be manipulated in the simple structures of non-nested relations.
- 7. No limitations are enforced by the relation scheme, in that it may have more than one attribute of either a spatial or time type.
- 8. A map can be seen as one or more relations that contain spatial data.
- 9. Formalism is provided for the relational algebra operations.
- 10. A set of few *kernel* operations achieves full functionality. This set consists of the known relational algebra operations and two more, *Unfold* and *Fold*. All the remainder operations have been defined in terms of those in the kernel set.
- 11. Operations on spatial data reduce to operations on relations.
- 12. Spatial data loss, identified in other approaches when operations are applied to spatial data, has been alleviated.
- 13. The full functionality of all the operations has been achieved, in that they all can be applied to pieces of spatial data of any data type.
- 14. The model achieves, and generalizes further, the functionality of other spatial models.
- 15. Regarding the management of spatio-temporal data, the model supports the evolution of spatial operations with respect to time.
- 16. The model does not restrict to the management of spatial or spatio-temporal data, but it can also be applied to the management of pure temporal data. Moreover, it has been shown that operations, originally defined for the management of spatial data, can also be applied to temporal data and, in some cases, this application has practical interest.
- 17. Regarding the management of temporal data, it did not necessitate to redefine the functionality of conventional algebra operations.
- 18. The model enables the management of 2-d spatial data but its extension to n-d data is straightforward.
- 19. In conjunction with previous research [LM97], the model enables the uniform management of any type of data.

20. Although discrete spatial changes have been considered, the indication is that the model can also be used for the management of continuous spatial changes (Subsection 8.3.5).
21. The indication is that the model can be the basis for the definition of a general-purpose nested relational model.
22. Based on an earlier extension, a further extension of SQL:1999 was proposed that achieves the full functionality of the proposed relational algebra.
23. It has been shown that the model can be implemented.

8.3 Future Work

Topics for further research are outlined in the following subsections.

8.3.1 Definition of Predicates and Functions

The definition of a *satisfactory* set of predicates and functions was beyond the objectives of this thesis. Only few were defined and some more can be found in Appendix A. However, a basic set of spatial predicates can enable testing topological relationships between two spatial objects. One first effort, called the *four intersection model* (4IM), has been defined in [Eg89, EF91]. This work has been extended further in [HT92]. Another extension of the 4IM, called the *nine intersection model* (9IM), has been defined in [EH92]. Further extensions of the 4IM and the 9IM are the *dimension extended 4IM* (DE4IM) [CFO93] and the *dimension extended 9IM* (DE9IM) [CF94]. In [CF94], a *calculus based method* (CBM) is developed and it is proved that it can express the set of relationships defined in the DE9IM. The CBM has also been slightly modified in [CF96], so as to support pure surfaces with holes and sets of spatial objects.

It is estimated that further research can combine the results of the above approaches so as to be applied to the data types defined in this thesis, especially to hybrid surfaces. It is also estimated that the total ordering of spatial quanta, which has been formalized in this thesis, can provide an alternative approach in the definition of directional predicates [PZ87, TPS96]. Finally, it is estimated that the definition of functions is also an issue of further research

8.3.2 Investigation of Efficient Storage Structures

Vector-based and raster-based approaches consider a tight coupling between the logical and the physical level, in that the operations defined at the logical level depend fully on the internal representation of spatial data. However, in the area of databases such a view contradicts data independence. The model proposed in this thesis is estimated to overcome many problems of spatial data modelling at the logical level. However, it is also estimated that an implementation of it should not necessarily be based on a raster-based approach, to which it is closer. Some initial investigation has already been undertaken in a vector-based implementation. Moreover, alternative solutions have to be considered, related to the use of efficient storage structures for the data types and the operations defined in this thesis.

8.3.3 Investigation of Optimisation Techniques

The major objective of this thesis has been the modelling of spatial and of spatio-temporal data. Consequently, the study of optimisation techniques, related to an efficient implementation, has fully been neglected. Regarding the optimisation of operations that can be applied to temporal relations, research work has been reported in [VLGM94] and in [LM97] and part of it has also been considered in an implementation. Two facts of major importance are the following: Firstly, in [LM97] it has been shown that, in many cases, the execution of an explicitly issued *Unfold* operation can be avoided. Secondly, operations whose definition has been based on *Unfold*, have actually been implemented in terms of an internal *Split* operation [LPS94, LPS95], which reduces drastically the execution time. In this thesis, however, additional operations have been defined in terms of *Unfold*. Hence, further research is worth undertaking both for these operations and also for the application of operation *Split* to spatial data.

8.3.4 Implementation

Based on the research results of the two previous sections, it is worth undertaking an efficient implementation, in a way similar to that for temporal data [VLGM94]. Alternative solutions have also to be investigated, related to the way the data types and operations can be integrated within a DBMS.

8.3.5 Modelling Continuous Change in Space and in Time

The spatial data model formalized in this thesis aimed at overcoming problems related to applications that require discrete spatial changes, such as cartography and cadastral systems (Section 2.8 and Subsection 7.4.1). In (Subsection 2.4.1) it has also been pointed out that the functionality required for the management of data, which changes discretely in space, is different than that for the management of data that changes continuously in space. As has however been reported (Subsection 7.4.1), an initial investigation [VL03b] has shown that operations originally defined for the management of continuous changes in space can be expressed in the SQL extension defined in Chapter 6. To express some of these operations, advantage is also taken of the recursive capabilities of SQL:1999. However, it is estimated that further research has to be undertaken related to the following issues:

- Exhaustive investigation, to determine whether new fundamental operations are really needed or not.
- Development of new storage structures and optimisation techniques that fit better continuous changes in space.

Regarding the management of spatio-temporal (and temporal data as well), discrete time has also been considered (Section 2.8 and Subsection 7.4.2), as is the common case in temporal models. However, initial investigation has shown that the combination of the proposed model with interpolation functions enables the modelling of moving objects. Again, however, it is estimated that the above research issues for continuous changes in space have also to be investigated for continuous changes in time.

REFERENCES

- [AB95] S. Abiteboul, C. Beeri, "The Power of Languages for the Manipulation of Complex Values". *VLDB Journal* 4(4), pp. 727-794, 1995.
- [Al83] J.F. Allen, "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM* 26(11), pp. 832-843, 1983.
- [Ar86] G. Ariav, "A temporally oriented data model", *ACM Transactions on Database Systems* 11(4), pp. 499-527, 1986.
- [AS93] K. K. Al-Taha, R. T. Snodgrass, M. D. Soo, "Bibliography on Spatiotemporal Databases." *SIGMOD Record* 22(1), pp. 59-67, 1993.
- [BBC97] O. Balovnev, M. Breunig, A.B. Cremers, "From GeoStore to GeoToolKit: The Second Step", *Proc. 5th International Symposium on Large Spatial Databases (SSD'97)*, Berlin, Germany, July 15-18 1997, M. Scholl, A. Voisard (eds.), *Advances in Spatial Databases*, Lecture Notes in Computer Science 1262, Springer-Verlag, pp. 223-237, 1997.
- [Be82] J. Ben-Zvi, *The time relational model*, Ph.D. Dissertation, Department of Computer Science, University of California, 1982.
- [Bent01] *MicroStation GeoGraphics User's Guide*, Version 7.2, Bentley Systems Inc., 2001.
- [BJ96] M.H. Böhlen, C.S. Jensen, "Seamless Integration of Time into SQL", Technical Report R-96-49, Department of Computer Science, Aalborg University, 1996.
- [BJS98] M.H. Böhlen, C.S. Jensen, B. Skjellaug, "Spatio-temporal database support for legacy applications", *Proc. of the 1998 ACM symposium on Applied Computing (SAC'98)*, February 27 - March 1 1998, pp. 226-234.
- [BK01] J.K. Berry, J. Kensinger, "Academic MapCalc: Educational Materials for Instruction in Grid-Based Map Analysis", *Proc. 15th Annual Conference on Geographic Information Systems*, Vancouver, British Columbia, Canada, February 19-22, 2001.

- [BM98] P.A. Burrough, R.A. McDonnell, *Principles of Geographical Information Systems*, Spatial Information Systems Series, Oxford University Press, 1998.
- [BVH96] L. Becker, A. Voigtmann, K. Hinrichs, "Temporal Support for Geo-Data in Object-Oriented Databases", *Proc. 7th International Conference Database and Expert Systems Applications (DEXA'96)*, Zurich, Switzerland, September 9-13 1996, R. Wagner, H. Thoma (eds.): *Database and Expert Systems Applications*, Lecture Notes in Computer Science 1134, pp. 79-93, 1996.
- [CC93] J. Clifford, A. Croker, "The historical relational data model (HRDM) revisited", A. Tansel, J. Clifford, S. Gadia, A. Segev, R. Snodgrass (eds.), *Temporal Databases: Theory, Design and Implementation*, Benjamin / Cummings, pp. 6-27, 1993.
- [CCF+96] G. Camara, M.A. Casanova, U.M. Freitas, J.P.C. Cordeiro, L. Hara, "A Presentation Language For Gis Cadastral Data", *Proc. 4th ACM workshop on Advances on Advances in Geographic Information Systems (GIS'96)*, Rockville, Maryland, USA, November 15 - 16 1996, pp. 139-146.
- [CF93] E. Clementini, P. Di Felice, "An Object Calculus for Geographic Databases", *Proc. of the 1993 ACM/SIGAPP Symposium on Applied Computing: States of the Art and Practice (SAC'93)*, Indianapolis, USA, February 14-16 1993, pp. 302-308.
- [CF94] E. Clementini, P. Di Felice, "A Comparison of Methods for Representing Topological Relationships", *Information Sciences* 80, pp. 1-34, 1994.
- [CF96] E. Clementini, P. Di Felice, "A Model for Representing Topological Relationships Between Complex Geometric Features in Spatial Databases", *Information Sciences* 90 (1-4), pp. 121-136, 1996.
- [CFG01] J.A. Coteló Lema, L. Forlizzi, R.H. Güting, E. Nardelli, M. Schneider, "Algorithms for Moving Objects Databases", Informatik-Report 289, FernUniversität Hagen, 2001. To appear in *The Computer Journal*.
- [CFO93] E. Clementini, P. Di Felice, P. van Oosterom, "A Small Set of Formal Topological Relationships Suitable for End-User Interaction", *Proc. 3th International Symposium on Large Spatial Databases (SSD'93)*, Singapore, June 23-25 1993, D. Abel, B. C.

- Ooi (eds.), *Advances in Spatial Databases*, Lecture Notes in Computer Science 692, Springer-Verlag, pp. 277-295, 1993.
- [CG94] T.S. Cheng, S.K. Gadia, "A Pattern Matching Language for Spatio-Temporal Databases", *Proc. 3th International Conference on Information and Knowledge Management (CIKM'94)*, Gaithersburg, Maryland, November 29 - December 2 1994, pp. 288-295.
- [CGN93] T.S. Cheng, S.K. Gadia, S.S. Nair, "Object Identity and Dimension Alignment in Parametric Databases", *Proc. 2th International Conference on Information and Knowledge Management (CIKM'93)*, Washington, DC, USA, November 1-5 1993, pp. 615-624.
- [CN97] E.P.F. Chan, J.N.H. Ng, "A General and Efficient Implementation of Geometric Operators and Predicates". *Proc. 5th International Symposium on Large Spatial Databases (SSD'97)*, Berlin, Germany, July 15-18 1997, M. Scholl, A. Voisard (eds.), *Advances in Spatial Databases*, Lecture Notes in Computer Science 1262, Springer-Verlag, pp. 69-93, 1997.
- [Co70] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM* 13(6), 1970.
- [Co72] E.F. Codd, "Relational completeness of data base sublanguages", R. Rustin (ed.), *Data Base Systems 6, Courant Computer Symposia Series*, Prentice-Hall, Englewood Cliffs, pp. 65-98, 1972.
- [Co79] E. F. Codd, "Extending the Database Relational Model to Capture More Meaning", *ACM Transactions on Database Systems* 4(4), pp. 397-434, 1979.
- [CSFG96] G. Camara, R.C.M. Souza, U.M. Freitas, J. Garrido, "Spring: integrating remote sensing and GIS by object-oriented data modelling", *Computers and Graphics* 20(3), pp. 395-403, 1996.
- [CT85] J. Clifford, A.U. Tansel, "On an algebra for historical relational databases: Two views", *Proc. of the 1985 ACM SIGMOD International Conference on Management of Data*, Austin, Texas, USA, May 28-31 1985, S. B. Navathe (ed.), *SIGMOD Record* 14(4), pp. 247-265, 1985.
- [CVL+94] D. Chrétien, Y. Viéumont, T. Larue, R. Legoff, D. Pastre, "The GéoSabrina design: the way to build a GIS above a spatial data server", *Proc. of the 1994 ACM Symposium on Applied*

- Computing* (SAC'94), Phoenix, AZ, USA, March 6-8 1994, pp. 328-332.
- [CW96] E.P.F. Chan, J.M.T. Wong, "Querying and Visualization of Geometric Data", *Proc. 4th ACM workshop on Advances on Advances in Geographic Information Systems (GIS'96)*, Rockville, Maryland, USA, November 15-16 1996, pp. 129-138.
- [CZ96] E.P.F. Chan, R. Zhu, "QL/G – A Query Language for Geometric Data Bases", *Proc. 1st International Conference on GIS, Urban Regional and Environmental Planning*, Samos, Greece, April 1996, pp. 271-286.
- [CZ99] C.X. Chen, C. Zaniolo, "Universal Temporal Extensions for Database Languages", *Proc. 15th International Conference on Data Engineering (ICDE'99)*, Sydney, Australia, March 23-26 1999, IEEE Computer Society Press, pp. 428-437.
- [CZ00] C.X. Chen, C. Zaniolo, "SQLST: A Spatio-Temporal Data Model and Query Language", *Proc. 19th International Conference on Conceptual Modeling (ER-2000)*, Salt Lake City, Utah, USA, October 9-12 2000, A. H. F. Laender, S. W. Liddle, V. C. Storey (eds.), *Conceptual Modeling - ER 2000*, Lecture Notes in Computer Science 1920, Springer-Verlag, pp. 96-111, 2000.
- [Da98] J.R. Davis, "IBM's DB2 Spatial Extender: Managing Geo-Spatial Information within the DBMS", Technical Report, IBM Corporation, May 1998.
- [DBVH97] H. Ditt, L. Becker, A. Voigtmann, K. Hinrichs, "Constraints and Triggers in an Object-Oriented Geo Database Kernel" *Proc. 8th International Workshop on Database and Expert Systems Applications (DEXA'97)*, Toulouse, France, September 1-2, 1997, IEEE Computer Society Press, pp. 508-513.
- [DDL03] C. J. Date, H. Darwen, N.A. Lorentzos, *Temporal Data and the Relational Model*, ISBN 1-55860-855-9, Morgan Kaufmann Publishers, San Fransisco, California, 2003.
- [DHT94] V. Delis, T. Hadzilacos, N. Tryfona, "An Introduction to Layer Algebra", *Proc. of the 6th International Symposium on Spatial Data Handling (SDH'94)*, Edinburgh, UK, September 1994, pp. 1020-1041.
- [EB95] M.J. Egenhofer, T. Bruns, "Visual Map Algebra: A Direct-Manipulation User Interface for GIS", *Proc. 3th IFIP 2.6 working conference on visual database systems (IFIP'95)*, Lausanne,

- Switzerland, March 27-29 1995, S. Spaccapietra, R. Jain (eds.), *Visual Database Systems 3, Visual Information Management*, IFIP Conference Proceedings 34 Chapman & Hall, pp. 235-253, 1995.
- [EF91] M. Egenhofer, R. Franzosa, "Point-set topological spatial relations", *International Journal of Geographic Information Systems* 2, pp. 161-174, 1991.
- [Eg89] M.J. Egenhofer, "A formal definition of binary topological relationships", *Proc. 3rd International Conference Foundations of Data Organization and Algorithms (FODO'89)*, Paris, France, June 21-23 1989, W. Litwin, H.-J. Schek (eds.), *Lecture Notes in Computer Science* 367, Springer-Verlag, pp. 457-472, 1989.
- [Eg94] M.J. Egenhofer, "Spatial SQL: A Query and Presentation Language", *IEEE Transactions on Knowledge and Data Engineering* 6(1), pp. 86-95, 1994.
- [EH92] M. J. Egenhofer, J. R. Herring, "Categorizing Binary Topological Relations Between Regions, Lines, and Points in Geographic Databases", Technical Report, Department of Surveying Engineering, University of Maine, 1992.
- [EJS98] O. Etzion, S. Jajodia, S.M. Sripada (eds.), *Temporal Databases: Research and Practice*, *Lecture Notes in Computer Science* 1399, Springer, 1998.
- [ELNR88] H.-D. Ehrich, F. Lohmann, K. Neumann, I. Ramm, "A Database Language for Scientific Map Data", R. Vinken (ed.), *Construction and Display of Geoscientific Maps Derived from Databases*, (Proc. Int. Coll. Dinkelsbühl 1986), pp. 139-152, 1988.
- [ES97] M. Erwig, M. Schneider, "Partition and Conquer", *Proc. International Conference On Spatial Information Theory (COSIT'97)*, Laurel Highlands, Pennsylvania, USA, October 15-18 1997, S. C. Hirtle, A. U. Frank (eds.), *Spatial Information Theory: A Theoretical Basis for GIS*, *Lecture Notes in Computer Science* 1329, pp. 389-407, 1997.
- [ES99] M. Erwig, M. Schneider, "The Honeycomb Model of Spatio-Temporal Partitions", *Proc. International Workshop Spatio-Temporal Database Management (STDBM'99)*, Edinburgh, Scotland, September 10-11 1999, M. H. Böhlen, C. S. Jensen, M. Scholl (eds.), *Lecture Notes in Computer Science* 1678, Springer-Verlag, pp. 39-59, 1999.

- [ES00] M. Erwig, M. Schneider, “Formalization of Advanced Map Operations”, *Proc. 9th International Symposium on Spatial Data Handling (SDH’2000)*, Beijing, China, August 10-12 2000, pp. 8a.3-17.
- [Esri99] “Features and Functions of ArcInfo 8”, White paper, Environmental Systems Research Institute, Inc., 1999.
- [Esri00] “ArcInfo 8: A New GIS for the New Millennium”, White paper, Environmental Systems Research Institute, Inc., 2000.
- [Esri01] “Geospatial Analysis with ArcGIS Desktop Extensions”, White paper, Environmental Systems Research Institute, Inc., 2001.
- [Esri02a] ArcSDE OnLine, URL: <http://arconline.esri.com/arcsdeonline>, Environmental Systems Research Institute, Inc. 2002.
- [Esri02b] ArcObject OnLine, URL: <http://arcobjectsonline.esri.com/ArcObjectsOnline>, Environmental Systems Research Institute, Inc. 2002.
- [FGG+99] A.U. Frank, S. Grumbach, R.H. Güting, C.S. Jensen, M. Koubarakis, N.A. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, T.K. Sellis, B. Theodoulidis, P. Widmayer, “Chorochronos: A Research Network for Spatiotemporal Database Systems”, *SIGMOD Record* 28(3), pp. 12-21, 1999.
- [FGNS00] L. Forlizzi, R.H. Güting, E. Nardelli, M. Schneider, “A Data Model and Data Structures for Moving Objects Databases”, *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, USA, May 16-18 2000, W. Chen, J. F. Naughton, P. A. Bernstein (eds.), *SIGMOD Record* 29(2), ACM, pp. 319-330, 2000.
- [FVM97] A.U. Frank, G.S. Volta, M. McGranaghan, “Formalization of Families of Categorical Coverages”, *International Journal of Geographical Information Science* 11(3), pp. 215-231, 1997.
- [Ga88] S.K. Gadia, “A homogeneous relational model and query languages for temporal databases”, *ACM Transactions on Database Systems* 13(4), pp. 418-448, 1988.
- [GBE+00] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, M. Vazirgiannis, “A Foundation for Representing and Querying Moving Objects”, *ACM Transactions on Database Systems* 25(1), pp. 1-42, 2000.

- [GFDP01] T. Griffiths, A.A.A. Fernandes, N. Djafri, N.W. Paton, "A Query Calculus for Spatio-Temporal Object Databases", *Proc. 8th International Symposium on Temporal Representation and Reasoning (TIME-01)*, Cividale del Friuli, Italy, June 14-16 2001, IEEE Computer Society, pp. 101-110..
- [GFP+01a] T. Griffiths, A.A.A. Fernandes, N.W. Paton, K.T. Mason, B. Huang, M.F. Worboys, "Tripod: A Comprehensive Model for Spatial and Aspatial Historical Objects", *Proc. 20th International Conference on Conceptual Modeling (ER 2001)*, Yokohama, Japan, November 27-30 2001, H. S. Kunii, S. Jajodia, A. Sølvberg (eds.), *Conceptual Modeling - ER 2001*, Lecture Notes in Computer Science 2224, Springer-Verlag, pp. 84-102, 2001.
- [GFP+01b] T. Griffiths, A.A.A. Fernandes, N.W. Paton, K.T. Mason, B. Huang, M.F. Worboys, C. Johnson, J.G. Stell, "Tripod: A Comprehensive System for the Management of Spatial and Aspatial Historical Objects", *Proc. 9th ACM International Symposium on Advances in Geographic Information Systems (GIS 2001)*, Atlanta, GA, USA, November 9-10 2001, pp. 118-123.
- [GJ00] G. Garani, R. Johnson "Joining Nested Relations and Subrelations", *Information Systems* 25(4), pp. 287-307, 2000.
- [GNT91a] M. Gargano, E. Nardelli, M. Talamo, "Abstract data types for the logical modeling of complex data", *Information Systems* 16(6), pp. 565-583, 1991.
- [GNT91b] M. Gargano, E. Nardelli, M. Talamo, "A Model for Complex Data: Completeness and Soundness Properties", *Proc. International Workshop on Geographic Database Management Systems*, Capri, Italia, May 1991, pp. 56-78.
- [Go92] M.F. Goodchild, "Geographical Data Modeling", *Computers and Geosciences* 18(4), pp. 401-408, 1992.
- [GR93] O. Günther, W. Riekert, "The Design of GODOT: An Object-Oriented Geographic Information System", *Data Engineering Bulletin* 16(3), pp. 4-9, 1993.
- [Gras02] *GRASS Homepage*, URL: <http://www.geog.uni-hannover.de/grass/index2.html>, 2002.
- [GRSS97] S. Grumbach, P. Rigaux, M. Scholl, L. Segoufin, "DEDALE, A Spatial Constraint Database". *Proc. 6th International Workshop Database Programming Languages (DBLP-6, 1997)*, Estes Park, Colorado, USA, August 18-20 1997, S. Cluet, R. Hull (eds.),

Lecture Notes in Computer Science 1369 Springer-Verlag, pp. 38-59, 1998.

- [GRS98a] S. Grumbach, P. Rigaux, L. Segoufin, "The DEDALE System for Complex Spatial Queries", *Proc. ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, USA, June 2-4 1998, ACM Press, pp. 213-224.
- [GRS98b] S. Grumbach, P. Rigaux, L. Segoufin, "Spatio-Temporal Data Handling with Constraints", *Proc. 6th international symposium on Advances in Geographic Information Systems (GIS'98)*, Washington, DC, USA, November 6-7 1998, ACM, pp. 106-111.
- [GRS00] S. Grumbach, P. Rigaux, L. Segoufin, "Manipulating Interpolated Data is Easier than You Thought", *Proc. 26th International Conference on Very Large Data Bases (VLDB 2000)*, Cairo, Egypt, September 10-14 2000, Morgan Kaufmann, pp. 156-165, 2000.
- [GS93] R.H. Güting, M. Schneider, "Realms: A Foundation for Spatial Data Types in Database Systems". *Proc. 3th International Symposium on Large Spatial Databases (SSD'93)*, Singapore, June 23-25 1993, D. J. Abel, B. C. Ooi (eds.), *Advances in Spatial Databases*, Lecture Notes in Computer Science 692, Springer-Verlag, pp. 14-35, 1993.
- [GS95] R.H. Güting, M. Schneider, "Realm-Based Spatial Data Types: The ROSE Algebra", *VLDB Journal* 4, pp. 100-143, 1995.
- [Gu88] R.H. Güting, "Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems", *Proc. International Conference on Extending Database Technology (EDBT'88)*, Venice, Italy, March 14-18 1988, J. W. Schmidt, S. Ceri, M. Missikoff, (eds.), *Advances in Database Technology – EDBT'88*, Lecture Notes in Computer Science 303, Springer-Verlag, pp. 506-527, 1988.
- [Gu89] R.H. Güting, "Gral: An Extensible Relational Database System for Geometric Applications", *Proc. 15th International Conference on Very Large Data Bases (VLDB'89)*, Amsterdam, The Netherlands, August 22-25 1989, Morgan Kaufmann, pp. 33-44, 1989.
- [Gu94] R.H. Güting, "An Introduction to Spatial Database Systems", *VLDB Journal* 3(4), pp. 357-399, 1994.

- [HS93] Z. Huang, P. Svensson, "Neighborhood Query and Analysis with GeoSAL, a Spatial Database Language" *Proc. 3th International Symposium on Large Spatial Databases (SSD'93)*, Singapore, June 23-25 1993, D. J. Abel, B. C. Ooi (eds.), *Advances in Spatial Databases*, Lecture Notes in Computer Science 692, Springer-Verlag, pp. 413-436, 1993.
- [HSH92] Z. Huang, P. Svensson, H. Hauska, "Solving Spatial Analysis Problems with GeoSal, A Spatial Query Language", *Proc. 6th International Working Conference on Scientific and Statistical Database Management (SSDBM'92)*, Monte Verita, Switzerland, 1992, pp. 1-17.
- [HT92] T. Hadzilacos, N. Tryfona, "A Model for Expressing Topological Integrity Constraints In Geographic Databases", *Proc. International Conference GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning*, Pisa, Italy, September 21-23 1992, A. U. Frank, I. Campari, U. Formentini (eds.), *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, Lecture Notes in Computer Science 639, Springer-Verlag, pp. 252-268, 1992.
- [HT96] T. Hadzilacos, N. Tryfona, "Logical Data Modeling for Geographical Applications", *International Journal in Geographical Information Systems* 10(2), pp. 179-203, 1996.
- [Ibm01] *IBM DB2 Spatial Extender User's Guide and Reference*, Version 7, International Business Machines Corporation, 2001.
- [Inf00] *Informix Geodetic DataBlade Module User's Guide*, Version 3, Informix Corporation, 2001.
- [Inf01] *IBM Informix Spatial DataBlade Module User's Guide*, Version 8.11, International Business Machines Corporation 2001.
- [Int95] *MGE-PC Getting Started*, Intergraph Corporation, 1995.
- [Int02] *Working with Geomedia Professional*, Intergraph Corporation, 2002.
- [Iso92] ISO/IEC 9075:1992, *Information Technology-Database Languages-SQL*, The International Organization for Standardization, 1992.
- [Iso96a] ISO/IEC JTC 1/SC 21/WG 3: MCI-009, *SQL3 Part 7: Temporal*, J. Melton, ISO Working Draft, 1996.

- [Iso96b] ISO/IEC JTC 1/SC 21/WG 3: MCI-044, *More Elements of Type PERIOD*, J. M. Sykes, Expert's Contribution, 1996.
- [Iso96c] ISO/IEC JTC 1/SC 21/WG 3: MAD-151, *Periods of Integers*, J. M. Sykes, Expert's Contribution, 1996.
- [Iso99] ISO/IEC 9075:1999, *Information Technology–Database Languages–SQL*, The International Organization for Standardization, 1999.
- [Iso02] ISO/IEC JTC 1/SC 32/WG 4: VIE-008, *SQL Multimedia and Application Packages (SQL/MM) Part 3: Spatial*, M. Ashworth (Ed.), ISO/IEC Committee Draft, 2002.
- [JJ92] Y.P. Jang, R.G. Johnson, "Nested relation based temporal data representation", *Proceedings 4th International Hong Kong Computer Society Database Workshop*, pp. 94-111, 1992.
- [JM80] S. Jones, P.S. Mason, "Handling the time dimension in a data base", S.M. Deen, P. Hammersley, (eds.), *International Conference on Data Bases*, June 1980, British Computer Society, pp. 65-83.
- [JS82] G. Jaeschke, H.-J. Schek, "Remarks on the Algebra of Non First Normal Form Relations", *Proc. of the ACM Symposium on Principles of Database Systems (PODS 1982)*, Los Angeles, California, March 29-31 1982, pp. 124-138.
- [Keig02] MFWorks, URL: <http://www.keigansystems.com/tech.html>, Keigan Systems, 2002.
- [KK94] Z. Kemp, A. Kowalczyk, "Incorporating the Temporal Dimension in a GIS", M. Worboys (ed.), *Innovations in GIS 1*, Taylor & Francis, pp. 89-102, 1994.
- [KKR95] P. Kanellakis, G. Kuper, P. Revesz, "Constraint Query Languages". *Journal of Computer and System Sciences* 51(1), pp. 26-52, 1995.
- [K193] N. Kline, "An Update of the Temporal Database Bibliography." *SIGMOD Record* 22(4), pp. 66-80, 1993.
- [KP90] K.C. Kirby, M. Pazner, "Graphic Map Algebra", *Proc. 4th International Symposium on Spatial Data Handling (SDH'90)* Zurich, Switzerland, 1990, Vol. 1, pp. 413-422.
- [KRSS98] G.M. Kuper, S. Ramaswamy, K. Shim, J. Su, "A Constraint-Based Spatial Extension to SQL", *Proc. 6th international symposium on*

- Advances in Geographic Information Systems (GIS'98)*, Washington, DC, USA, November 6-7 1998, pp. 112-117.
- [La92] G. Langran, *Time in Geographic Information Systems*, Taylor and Francis, 1992.
- [LD96] N.A. Lorentzos, H. Darwen, "Extension to SQL2 Binary Operations for Temporal Data". Invited paper, *Proc. 3rd HERMIS Conference*, Athens, September 26-28 1996, pp. 462-469.
- [LD98] N.A. Lorentzos, K.A. Dondis, "Query by Example for Nested Tables", *Proc. 9th International Conference Database and Expert Systems Applications (DEXA'98)*, Vienna, Austria, August 24-28 1998, G. Quirchmayr, E. Schweighofer, T.J.M. Bench-Capon (eds.), *Database and Expert Systems Applications*, Lecture Notes in Computer Science 1460 Springer-Verlag, pp. 716-725, 1998.
- [LGMR01] P.A. Longley, M.F. Goodchild, D.J. Maguire, D.W. Rhind, *Geographic Information Systems and Science*, John Wiley and Sons, Ltd, 2001.
- [LH98] W. F. Limp, D. Harmon, *Inside Geomedia*, OnWord Press, 1998.
- [LJ88a] N.A. Lorentzos, R.G. Johnson, "Extending Relational Algebra to Manipulate Temporal Data", *Information Systems* 13(3), pp. 289-296, 1988.
- [LJ88b] N.A. Lorentzos, R.G. Johnson, "TRA: A model for a temporal relational algebra", *Proc. of the IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems*, Sophia-Antipolis, France, May 13-15, Rolland, F. Bodart, M. Leonard, (eds.), *Temporal Aspects in Information Systems*, North-Holland/Elsevier, 203-215, 1988.
- [LM94] N. A. Lorentzos, Y. Manolopoulos, "Efficient Management of 2-d Interval Relations", *Proc. 5th International Conference Database and Expert Systems Applications (DEXA'94)*, Athens, Greece, September 7-9 1994, D. Karagiannis (ed.), *Database and Expert Systems Applications*, Lecture Notes in Computer Science 856, Springer-Verlag, pp. 72-82, 1994.
- [LM95] N. A. Lorentzos, Y. Manolopoulos, "Functional Requirements for Historical and Interval Extensions to the Relational Model", *Data and Knowledge Engineering* 17, pp. 59-86, 1995.
- [LM97] N.A. Lorentzos, Y.G. Mitsopoulos, "SQL Extension for Interval Data", *IEEE Transactions on Knowledge and Data Engineering* 9(3), pp. 480-499, 1997.

- [LM03] N.A. Lorentzos, Y.G. Mitsopoulos, "Modelling Transaction Time in the Relational Model", Internal Report, Informatics Laboratory, Agricultural University of Athens, May 2003.
- [Lo88] N.A. Lorentzos, *A Formal Extension of the Relational Model for the Representation and Manipulation of Generic Intervals*, Ph.D Dissertation, Birkbeck College, University of London, 1988.
- [Lo93] N.A. Lorentzos, "The Interval Extended Relational Model and Its Application to Valid Time Databases", A. Tansel, J. Clifford, S. Gadia, A. Segev, R. Snodgrass (eds.) *Temporal Databases: Theory, Design and Implementation*, Benjamin / Cummings, pp. 67-91, 1993.
- [Lo00] E. J. Lorup, IDRISI Tutorial on WWW, URL: <http://www.sbg.ac.at/geo/idrisi/wwwtutor/tuthome.htm>, 2000.
- [LPS94] N.A. Lorentzos, A. Poulouvasilis, C. Small, "Implementation of Update Operations for Interval Relations", *Computer Journal* 37(3), pp. 163-176, 1994.
- [LPS95] N.A. Lorentzos, A. Poulouvasilis, C. Small, "Manipulation Operations for an Interval-Extended Relational Model", *Data and Knowledge Engineering* 17, pp. 1-29, 1995.
- [LPV93] T. Larue, D. Pastre, Y. Viémont, "Strong Integration of Spatial Domains and Operators in a relational Database System", *Proc. 3rd International Symposium on Large Spatial Databases (SSD'93)*, Singapore, June 23-25, D. J. Abel, B. C. Ooi (eds.), *Advances in Spatial Databases*, Lecture Notes in Computer Science 692, pp. 53-72, 1993.
- [LR94] H.-C. Liu, K. Ramamohanarao, "Algebraic Equivalences among Nested Relational Expressions", *Proc. of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, Gaithersburg, Maryland, November 29 - December 2, 1994, ACM, pp. 234-243.
- [LT92] R. Laurini, D. Thompson, *Fundamentals of Spatial Information Systems*, The A.P.I.C. series Number 37, Academic Press, 1992.
- [LTV99] N. A. Lorentzos, N. Tryfona, J. R. R. Viqueira, "Relational Algebra for Spatial Data Management", *Proc. International Workshop Integrated Spatial Databases, Digital Images and GIS*, Portland, Maine, USA, June 14-16, 1999, P. Agouris, A. Stefanidis, (eds.), *Lecture Notes in Computer Science* 1737, pp. 192-208, 1999.

- [LVT99] N. A. Lorentzos, J. R. R. Viqueira, N. Tryfona, “Quantum-Based Spatial Extension to the Relational Model”, *Proc. 7th Panhellenic Conference on Informatics*, Ioannina (Greece), August 26-29, 1999, III 34-44.
- [LVT00] N. A. Lorentzos, J. R. R. Viqueira, N. Tryfona, “Quantum-Based Extension to the Relational Model”. D. I. Fotiadis, S. D. Nikolopoulos, (eds.), *Advances in Informatics*, World Scientific, pp. 188-199, 2000.
- [LVT03] N. A. Lorentzos, J. R. R. Viqueira, N. Tryfona, “On a Spatio-temporal Relational Model Based on Quanta”, T. Sellis, M. Koubarakis, A. Frank, S. Grumbach, R. H. Güting, C. Jensen, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Schek, M. Scholl, B. Theodoulidis, N. Tryfona (eds.), *Spatio-temporal Databases: The Chorochronos Approach*, Lecture Notes in Computer Science, Vol. 2520, Springer-Verlag, pp. 139-141 (forthcoming, summer 2003).
- [Ma99] A. MacDonald, *Building a Geodatabase*, Environmental Systems Research Institute, 1999.
- [Mapi01] “Enterprise Mapping Deployments, Managing Spatial Data in a Relational Database Management System”, White Paper, MapInfo Corporation, 2001.
- [Mapi02] *MapInfo Professional User’s Guide 7.0*, MapInfo Corporation, 2002.
- [ME01] J. Melton, A. Eisenberg, “SQL Multimedia and Application Packages (SQL/MM)”, *SIGMOD Record* 30(4), pp. 97-102, 2001.
- [MJ94] C.B. Medeiros, G. Jomier, “Using Versions in GIS”, *Proc. 5th International Conference Database and Expert Systems Applications (DEXA’94)*, Athens, Greece, September 7-9 1994, D. Karagiannis (ed.), *Database and Expert Systems Applications*, Lecture Notes in Computer Science 856, Springer-Verlag, pp. 465-474, 1994.
- [MJ01] J. McCoy, K. Johnston, *Using ArcGis Spatial Analyst*, Environmental Systems Research Institute, 2001.
- [MO86] F. Manola, J.A. Orenstein, “Toward a General Spatial Data Model for an Object-Oriented DBMS”, *Proc. 12th International Conference on Very Large Data Bases (VLDB’86)*, Kyoto, Japan, August 25-28 1986, Morgan Kaufmann, pp. 328-335.

- [MRA00] J. Moreira, C. Ribeiro, T. Abdessalem, "Query operations for moving objects database systems", *Proc. of the 8th ACM Symposium on Advances in Geographic Information Systems (GIS 2000)*, Washington D.C., USA, November 10-11 2000, ACM, pp. 108-114.
- [MS91] E. McKenzie, R. Snodgrass, "Evaluation of relational algebras incorporating the time dimension in databases", *ACM Computing Surveys* 23(4), pp. 501-543, 1991.
- [MS02] J. Melton, A. R. Simon, *SQL:1999 - Understanding Relational Language Components*, Morgan Kaufmann Publishers, Academic Press, 2002.
- [MSR99] J. Moreira, J.-M. Saglio, C. Ribeiro, "Representation and Manipulation of Moving Points: An Extended Data Model for Location Estimation", *Journal of Cartography and Geographic Information Systems*, ACSM, 26(2), 1999.
- [MSW99] M. Minami, M. Sakala, J. Wrightsell, *Using ArcMap*, Environmental Systems Research Institute, 1999.
- [NA93] S.B. Navathe, R. Ahmed, "Temporal extensions to the relational Model and SQL", A. Tansel, J. Clifford, S. Gadia, A. Segev, R. Snodgrass, (eds.), *Temporal Databases: Theory, Design and Implementation*, Benjamin / Cummings, pp. 92-109, 1993.
- [NTE92] R.G. Newell, D. Theriault, M. Easterfield, "Temporal GIS – Modeling the Evolution of Spatial Data in Time", *Computers and Geosciences* 18(4), pp. 427-433, 1992.
- [Ogis99] *The OpenGIS Implementation Specification: Simple Features Specification for SQL*, Revision 1.1, OpenGIS Project Document 99-049, Open GIS Consortium Inc., 1999.
- [Ogis00] *The OpenGIS Abstract Specification Topic 6: The Coverage Type and its Subtypes*, version 6, OpenGIS Project Document 00-106, Open GIS Consortium Inc., 2000.
- [Ogis01a] *The OpenGIS Abstract Specification Topic 1: Feature Geometry (ISO 19107 Spatial Schema)*, Version 5, OpenGIS Project Document 01-101, Open GIS Consortium Inc., 2001.
- [Ogis01b] *The OpenGIS Implementation Specification: Grid Coverage*, Revision 1.00, OpenGIS Project Document 01-004, Open GIS Consortium Inc., 2001.
- [OP01] A. d'Onofrio, E. Pourabbas, "Formalization of Temporal Thematic Map Contents", *Proc. of the 9th ACM International Symposium on*

- Advances in Geographic Information Systems* (GIS 2001), Atlanta, GA, USA, November 9-10 2001, ACM, pp. 15-20.
- [Ora00] *Oracle Spatial: Users Guide and Reference*. Release 8.1.7. Oracle corporation, 2000.
- [Pa82] T. Pavlidis, *Algorithms for graphics and image processing*, Computer Science Press, 1982.
- [PA86] P. Pistor, F. Andersen, "Designing A Generalized NF2 Model with an SQL-Type Language Interface", *Proc. 12th International Conference on Very Large Data Bases (VLDB'86)*, Kyoto, Japan, August 25-28 1986, Morgan Kaufmann, pp. 278-285.
- [Pe90] D.J. Peuquet, "A conceptual framework and comparison of spatial data models", D.J. Peuquet, D.F. Marble (eds.), *Introductory readings in Geographic information systems*, Taylor and Francis, pp. 250-285, 1990.
- [Pf00] D. Pfoser, *Issues in the Management of Moving Point Objects*, PhD Dissertation, Department of Computer Science, Aalborg University, 2000.
- [PLL+98] K. Park, J. Lee, K. Lee, K. Ahn, J. Lee, J. Kim, "The Development of GEUS: A Spatial DBMS Tightly Integrated with an Object-Relational Database Engine", *Proc. Annual Conference Urban & Regional Information Systems Association (URISA'98)*, Charlotte, North Carolina, July 1998, pp. 256-267.
- [Post01] *PostgreSQL 7.2 User's Guide*, PostgreSQL Global Development Group, 2001.
- [PT98] D. Pfoser, N. Tryfona, "Requirements, Definitions, and Notations for Spatiotemporal Application Environments", *Proc. 6th international symposium on Advances in Geographic Information Systems (GIS'98)*, Washington, DC, USA, November 6-7 1998, ACM, pp. 124-130.
- [PTS94] D. Papadias, Y. Theodoridis, T. Sellis, "The Retrieval of Direction Relations Using R-trees", *Proc. 5th International Conference Database and Expert Systems Applications (DEXA'94)*, Athens, Greece, September 7-9 1994, D. Karagiannis (ed.), *Database and Expert Systems Applications*, Lecture Notes in Computer Science 856, Springer-Verlag, pp. 173-182, 1994.
- [PZ87] D. Peuquet, C.X. Zhan, "An Algorithm to Determine the Directional Relationship Between Arbitrarily-Shaped Polygons in the Plane", *Pattern Recognition* 20(1), pp. 65-74, 1987.

- [RFS88] N. Roussopoulos, C. Faloutsos, T. K. Sellis, "An Efficient Pictorial Database System for PSQL". *IEEE Transactions on Software Engineering* 14(5), pp. 639-650, 1988.
- [RHS01] *MapCalc User's Guide*. Red Hen Systems Inc., 2001.
- [RK98] A. Raza, W. Kainz, "Design and Implementation of Temporal GIS for Monitoring the Urban Land Use Change", Proc. Spatial Information Technology Towards 2000 and Beyond, Beijing China, June 17-19 1998, pp. 417-427.
- [RKS88] M.A. Roth, H.F. Korth, A. Silberschatz, "Extended Algebra and Calculus for Nested Relational Databases", *ACM Transactions on Database Systems* 13(4), pp. 389-417, 1988.
- [Ro93] J.W. van Roessel, "Conceptual Folding and Unfolding of Spatial Data for Spatial Queries", *Towards SQL Database Extensions for Geographic Information Systems*, V.B. Robinson, H. Tom (eds.), National Institute of Standards and Technology Report NISTIR 5258, pp. 133-148, 1993.
- [Ro94] J.W. van Roessel, "An Integrated Point-Attribute Model for Four Types of Areal GIS Features", *Proc. 6th International Symposium on Spatial Data Handling (SDH'94)*, Edinburgh, Scotland, UK, 1994, vol. 1, pp. 127-144.
- [RS87] L. Rowe, M. Stonebraker, "The POSTGRES Data Model", *Proc. 13th International Conference on Very Large Data Bases (VLDB'87)*, Brighton, England, September 1-4, 1987, Morgan Kaufmann, pp. 83-96.
- [RSSG02] P. Rigaux, M. Scholl, L. Segoufin, S. Grumbach, "Building a Constraint-Based Spatial Database System Model, Languages, and Implementation", To appear in *Information System Journal*, 2002.
- [RSV02] P. Rigaux, M. Scholl, A. Voisard, *Spatial Databases: with application to GIS*, Morgan Kaufmann Publishers, Academic press, 2002.
- [Sa90a] N.L. Sarda, "Algebra and query language for a historical data model", *Computer Journal* 33(1), pp. 11-18, 1990.
- [Sa90b] N. L. Sarda, "Extensions to SQL for historical databases", *IEEE Transactions on Knowledge and Data Engineering* 2(2), pp. 220-230, 1990.
- [Sa90c] H. Samet, *Applications of Spatial Data Structures, Computer Graphics, Image Processing and GIS*, Addison Wesley, 1990.

- [SA95] H. Samet, W.G. Aref, "Spatial Data Models and Query Processing", W. Kim (ed.), *Modern Database Systems: The Object Model, Interoperability, and Beyond*, ACM Press and Addison-Wesley, pp. 338-360, 1995.
- [SCR99] S. Shekhar, S. Chawla, S. Ravada, "Spatial Databases: Accomplishments and Research Needs", *IEEE Transactions on Knowledge and Data Engineering* 11(1), pp. 45-55, 1999.
- [SDS00] Z. Stojanovic, S. Djordjevic-Kajan, D. Stojanovic, "Visual Query and Analysis Tool of the Object-Relational GIS Framework", *Proc. of the 2000 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2000)*, McLean, VA, USA, November 6-11 2000, pp. 328-335.
- [SH91] P. Svensson, Z. Huang "Geo-SAL - a Query Language for Spatial Data Analysis", *Proc. 2nd International Symposium on Large Spatial Databases (SSD'91)*, Zürich, Switzerland, August 28-30 1991, O. Gunther, H.J. Schek (eds.), *Advances in Spatial Databases*, Lecture Notes in Computer Science 525. Springer-Verlag, pp. 119-140, 1991.
- [Sn87] S. Snodgrass, "The temporal query language TQUEL", *ACM Transactions on Database Systems* 12(2), pp. 247-298, 1987.
- [Sn95] R.T. Snodgrass, *The TSQL2 Temporal Query Language*, Kluwer, 1995.
- [SR86] M. Stonebraker, L. Rowe, "The Design of POSTGRES", *Proc. of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 28-30, 1986*, pp. 340-355.
- [SRH90] M. Stonebraker, L. Rowe, M. Hirohama, "The implementation of Postgres". *IEEE Transactions on Knowledge and Data Engineering* 2(1), pp. 125-142, 1990.
- [SS86] H.-J. Schek, M.H. Scholl, "The Relational Model with Relation-Valued Attributes", *Information Systems* 11(2), pp. 137-147, 1986.
- [SS88] R. Stam, R. Snodgrass, "A Bibliography on Temporal Databases", *IEEE/Data Engineering* 11(4), pp. 53-61, 1988.
- [SS96] E. Stefanakis, T. Sellis, "A DBMS repository for the application domain of Geographic Information Systems", *Proc. 7th International Symposium on Spatial Data Handling (SDH'96)*, Delft, The Netherlands, August 12-16 1996, Taylor-Francis, pp. 119-130.

- [SV89] M. Scholl, A. Voisard, "Thematic Map Modeling", *Proc. 1st International Symposium on Large Spatial Databases (SSD'89)*, Santa Barbara, California, July 17-18 1989, A. Buchmann et al. (eds.), *Design and Implementation of Large Spatial Databases*, Lecture Notes in Computer Science 409, Springer-Verlag, pp. 167-190, 1990.
- [SV92] M. Scholl, A. Voisard, "Object-Oriented Database Systems for Geographic Applications: An Experiment with O2", F. Bancilhon, C. Delobel, P. C. Kanellakis (eds.), *The O2 Book*, Morgan Kaufman, 1992.
- [SW86] H.-J. Schek, W. Waterfeld, "A Database Kernel System For Geoscientific Applications", *Proc. 2nd International Symposium on Spatial Data Handling (SDH'86)*, Seattle, USA, June 1986.
- [SWCD97] P. Sistla, O. Wolfson, S. Chamberlain, S. Dao, "Modeling and Querying Moving Objects", *Proc. of the 13th International Conference on Data Engineering (ICDE'97)*, Birmingham, U.K., April 7-11 1997, IEEE Computer Society, pp. 422-432.
- [Ta86] A.U. Tansel, "Adding time dimension to relational model and extending relational algebra", *Information Systems 11(4)*, pp. 343-355, 1986.
- [TC90] A. Tuzhilin, J. Clifford, "A Temporal Relational Algebra as a Basis for Temporal Relational Completeness", *Proc. 16th International Conference on Very Large Data Bases (VLDB'90)*, Brisbane, Queensland, Australia, August 13-16 1990, Morgan Kaufmann, pp. 13-23.
- [TCG+93] A.U. Tansel, J. Clifford, S. Gadia, A. Segev, R. Snodgrass, *Temporal Databases: Theory, Design and Implementation*, Benjamin / Cummings, 1993.
- [TG89] A.U. Tansel, L. Garnett, "Nested historical relations", *Proc. of the 1989 ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, May 31 - June 2 1989, *SIGMOD Record* 18(2), pp. 284-293, 1989.
- [TH98] N. Tryfona, T. Hadzilacos "Logical Data Modelling of Spatio Temporal Applications: Definitions and a Model", *Proc. of the 1998 International Database Engineering and Applications Symposium (IDEAS 1998)*, Cardiff, Wales, U.K., July 8-10 1998, IEEE Computer Society, pp. 14-23.

- [TL82] D.C. Tschritzis, F.H. Lochovsky, *Data Models*, Prentice Hall, 1982.
- [To90] C.D. Tomlin, *Geographic Information Systems and Cartographic Modeling*, Prentice Hall, 1990.
- [To91] C.D. Tomlin, "Cartographic Modeling", D. Maguire, M. Goodchild, D. Rhind, eds., *Geographical Information Systems: Principles and Applications*, Longman Group Ltd., Harlow, Essex, U.K., pp. 361-374, 1991.
- [To94] C.D. Tomlin, "Map Algebra: one perspective", *Landscape and Urban Planning*, an International Journal of Landscape Ecology, *Landscape Planning and Landscape Design* 30, pp. 3-12, 1994.
- [To97] C.D. Tomlin, "Cartographic modeling techniques for highway corridor planning", *2nd Annual International Seminar on GIS Applications in the Public Sector*, Korea Research Institute for Human Settlements, 1997.
- [TPS96] Y. Theodoridis, D. Papadias, E. Stefanakis, "Supporting Direction Relations in Spatial Database Systems", *Proc. 7th International Symposium on Spatial Data Handling (SDH'96)*, Delft, The Netherlands, August 12-16 1996, Taylor-Francis, II:12A. 1-15.
- [TT96] V.J. Tsotras, A. Kumar, "Temporal Database Bibliography Update." *SIGMOD Record* 25(1), pp. 41-51, 1996.
- [Tu00] C. Tucker, *Using ArcToolbox*, Environmental Systems Research Institute, 2000.
- [VBH97] A. Voigtmann, L. Becker, K. Hinrichs, "Physical Design Aspects of an Object-Oriented Geo-Database Kernel", *Proc. 8th International Workshop on Database and Expert Systems Applications (DEXA'97)*, Toulouse, France, September 1-2, 1997, IEEE Computer Society Press, pp. 529-534.
- [VE93] G. Volta, M. Egenhofer, "Interaction with GIS Attribute Data Based on Categorical Coverages", *Proc. International Conference On Spatial Information Theory (COSIT'93)*, Marciana Marina, Elba Island, Italy, September 19-22 1993, A. Frank, I. Campari (eds.), *Spatial Information Theory: A Theoretical Basis for GIS*, Lecture Notes in Computer Science 716, Springer-Verlag, pp. 215-233, 1993.
- [Vi00] J. R. R. Viqueira, "Relational Algebra for Spatio-temporal Data Management", *Proc. of the EDBT 2000 PhD Workshop*, March 31

- April 1, 2000. <http://www.edbt2000.uni-konstanz.de/phd-workshop/>
- [VL01] J. R. R. Viqueira, N. A. Lorentzos, "Spatio-temporal SQL Extension", *Proc. 8th Panhellenic Conference on Informatics*, Cyprus, November 8-10, 2001, Vol. 1, pp. 264-273.
- [VL03a] J. R. R. Viqueira, N. A. Lorentzos, "Spatio-temporal SQL", Y. Manolopoulos, S. Evripidou, A. Kakas (eds.), *Advances in Informatics - Post-proceedings 8th Panhellenic Conference in Informatics*, Lecture Notes in Computer Science 2563 Springer-Verlag, pp. 50-63, 2003.
- [VL03b] J.R.R. Viqueira, N.A. Lorentzos, "Management of Continuous Spatial Changes", Submitted to the *9th Panhellenic Conference on Informatics*, Salonica, Greece, November 21-23, 2003.
- [VLG98] C. Vassilakis, N.A. Lorentzos, P. Georgiadis, "Implementation of Transaction and Concurrency Control Support in a Temporal DBMS", *Information Systems* 23(5), pp. 335-350, 1998.
- [VLGM94] C. Vassilakis, N.A. Lorentzos, P. Georgiadis, Y.G. Mitsopoulos, "ORES: Design and Implementation of a Temporal DBMS", Technical Report, Department of Informatics, University of Athens, 1994.
- [VLT01] J.R.R. Viqueira, N.A. Lorentzos, N. Tryfona, "Formalism for Spatio-Temporal Data Management", *Proc. HERCMA Conference*, Athens, September 20-22, 2001.
- [VO92] T. Vrijlbrief, P. van Oosterom, "The GEO++ system: An Extensible GIS", *Proc. 5th International Symposium on Spatial Data Handling (SDH'92)*, Charleston, South Carolina, August 1992. pp. 40-50.
- [Vo97] A. Voigtmann, *An Object-Oriented Database Kernel for Spatio-Temporal Geo-Applications*. PhD Dissertation, Westf. Wilhelms-Universität Münster, Germany, 1997.
- [Wi98] S. Winter, "Bridging Vector and Raster Representation in GIS" *Proc. 6th international symposium on Advances in Geographic Information Systems (GIS'98)*, Washington, DC, USA, November 6-7 1998, ACM, pp. 57-62.
- [WF99] S. Winter, A.U. Frank, "Functional Extensions of a Raster Representation for Topological Relations", *Proc. 2nd International Conference Interoperating Geographic Information Systems (INTEROP'99)*, Zurich, Switzerland, March 10-12 1999,

- A. Vckovski, K. Brassel, H.-J. Schek, (eds.), *Interoperating Geographic Information Systems*, Lecture Notes of Computer Science 1580, Springer-Verlag, pp. 293-304, 1999.
- [WF00] S. Winter, A.U. Frank, "Topology in Raster and Vector Representation", *GeoInformatica* 4(1), pp. 35-65, 2000.
- [WH94] M. Wachowicz, R.G. Healey, "Towards Temporality in GIS", M. Worboys (ed.), *Innovations in GIS I*, Taylor & Francis, pp. 105-115, 1994.
- [Wo94] M.F. Worboys, "A Unified Model for Spatial and Temporal Information", *The Computer Journal* 37(1), pp. 36-34, 1994.
- [Wo95] M.F. Worboys, *GIS: A Computing Perspective*, Taylor and Francis, 1995.
- [WS92] W. Waterfeld, H.-J. Schek, "The DASDBS Geokernel – An Extensible Database System for GIS", A.K. Turner (ed.), *Three-Dimensional Modeling with Geoscientific Information Systems*, Kluwer Academic Publishers, pp. 69-84, 1992.
- [WXCJ98] O. Wolfson, B. Xu, S. Chamberlain, L. Jiang, "Moving Objects Databases: Issues and Solutions", *Proc. 10th International Conference on Scientific and Statistical Database Management (SSDBM'98)*, Capri, Italy, July 1-3 1998, IEEE Computer Society, pp. 111-122.
- [YC94a] T.-S. Yeh, B. de Cambray, "Managing Multidimensional (2D, 2.5D, and 3D) Data in a Geographical Database". *Proc. 6th International Conference on Management of Data (COMAD'94)*, Windsor Manor Sheraton & Towers, Bangalore, India, December 19-21 1994, pp. 19-21.
- [YC94b] T.-S. Yeh, B. de Cambray, "How to Model Highly Variable Data in a Complex Object Data Model", *Proc. 6th International Conference on Management of Data (COMAD'94)*, Windsor Manor Sheraton & Towers, Bangalore, India, December 19-21 1994, pp. 169-186.
- [YC95] T.-S. Yeh, B. de Cambray, "Modeling Highly Variable Spatio-Temporal Data", *Proc. 6th Australasian Database Conference (ADC'95)*, Glenelg/Adelaide, South Australia 1995, Australian Computer Science Communications, Vol 17, No. 2, pp. 221-230, 1995.
- [Ze99] M. Zeiler, *Modeling Our World: The ESRI Guide to Geodatabase Design*, Environmental Systems Research Institute, Inc. 1999.

APPENDIX A

IMPORTANT TAUTOLOGIES

- $Unfold[\mathbf{A}](Unfold[\mathbf{A}](R)) = Unfold[\mathbf{A}](R)$
- $Unfold[A_1, A_2](R) = Unfold[A_2, A_1](R)$
- $Unfold[\mathbf{A}](Fold[\mathbf{A}](R)) = Unfold[\mathbf{A}](R)$
- If only quanta are contained in attributes \mathbf{A} then $Unfold[\mathbf{A}](R) = R$

- $Fold[\mathbf{A}](Fold[\mathbf{A}](R)) = Fold[\mathbf{A}](R)$
- $Fold[A_1, A_2](R) \neq Fold[A_2, A_1](R)$
- $Fold[\mathbf{A}](Unfold[\mathbf{A}](R)) = Fold[\mathbf{A}](R)$

- $Normalise[\mathbf{A}](Normalise[\mathbf{A}](R)) = Normalise[\mathbf{A}](R)$
- $Normalise[A_1, A_2](R) \neq Normalise[A_2, A_1](R)$
- $Normalise[A_1, A_2](R) = Fold[A_1, A_2](Unfold[A_2](R))$

- $R_1 \ QUnion[\mathbf{A}] \ R_2 = Normalise[\mathbf{A}](R_1 \ Union \ R_2)$

APPENDIX B

BNF FOR SQL EXTENSION

B.1 GENERAL REMARKS

- Only the extensions to SQL:1999 are given. Hence, it is assumed that the SQL:1999 documentation is available for reference. The extension is given in bold.

B.2 QUERY EXPRESSIONS

```
<query expression> ::=
    <IXSQL non-join query expression>
    | <IXSQL joined table>
    | <IXSQL unary query expression>

<IXSQL non-join query expression> ::=
    (<non-join query expression>) [<reformat clause>] [<normalise clause>]
    | <non-join query expression>

<non-join query expression> ::=
    <non-join query term>
    | <query expression> UNION [ALL]
        [EXPANDING (<reformat column list>)]
        [<corresponding spec>]
        <IXSQL query term>
    | <query expression> EXCEPT [ALL]
        [EXPANDING (<reformat column list>)]
        [<corresponding spec>]
        <IXSQL query term>
    | <query expression> WUNION [ALL]
        OF (<reformat column list>)
        [EXPANDING (<reformat column list>)]
        <IXSQL query term>
```

```

| <query expression> WEXCEPT [ALL]
      OF (<reformat column list>)
      [EXPANDING (<reformat column list>)]
      <IXSQL query term>

<IXSQL query term> ::=
  <IXSQL non-join query term>
  | <IXSQL joined table>
  | <IXSQL unary query expression>

<IXSQL non-join query term> ::=
  <non-join query term>
  | (<non-join query term>) [<reformat clause>] [<normalise clause>]

<non-join query term> ::=
  <non-join query primary>
  | <IXSQL query term> INTERSECT [ALL]
      [EXPANDING(<reformat column list>)]
      [<corresponding spec>]
      <IXSQL query primary>
  | <IXSQL query term> WINTERSECT [ALL]
      OF (<reformat column list>)
      [EXPANDING(<reformat column list>)]
      <IXSQL query primary>

<IXSQL query primary> ::=
  <non-join query primary>
  | <IXSQL joined table>
  | <IXSQL unary query exp>

<non-join query primary> ::=
  <simple table>
  | (<IXSQL non-join query expression>)

<simple table> ::=
  <IXSQL query specification>
  | <IXSQL table value constructor>
  | <IXSQL explicit table>

<IXSQL query specification> ::=
  <query specification>[<reformat clause>] [<normalise clause>]

<query specification> ::=
  SELECT <set quantifier> <select list> <table expression>

<table expression> ::=
  <from clause> [<where clause>] [<group by clause>] [<having clause>]

```

<from clause> ::=
 FROM <table reference> [{, <table reference>}...]

<table reference> ::=
 <table name> [[AS] <correlation name> [(derived column list)]]
 | <table subquery> [[AS] <correlation name> [(derived column list)]]
 | **<IXSQL joined table>**
 | **<IXSQL unary query expression>**

<IXSQL row value constructor> ::=
 (<row value constructor>)[<reformat clause>][<normalise clause>]

<IXSQL table value constructor> ::=
 (<table value constructor>) [<reformat clause>] [<normalise clause>]

<IXSQL explicit table> ::=
 TABLE <table name> [<reformat clause>] [<normalise clause>]

<IXSQL joined table> ::=
 (<joined table>) [<reformat clause>] [<normalise clause>]
 | <joined table>

<joined table> ::=
 <cross join>
 | <qualified join>
 | **<IXSQL overlay>**
 | (<joined table>)

<cross join> ::=
 <table reference> CROSS JOIN
 [EXPANDING (<reformat column list >)]
 <table reference>

<qualified join> ::=
 <table reference> [NATURAL] [<join type>] JOIN
 [EXPANDING (<reformat column list >)]
 <table reference>
 [<join specification>]

<IXSQL overlay> ::=
 <table reference> [NATURAL][<overlay type>] OVERLAY [ALL]
 [OF (<reformat column list>)]
 [EXPANDING (<reformat column list >)]
 <table reference>

<overlay type> ::=
 INNER
 | {LEFT | RIGHT | FULL} [OUTER]

<IXSQL unary query expression> ::=
 <unary query expression>
 | (<unary query expression>) [<reformat clause>] [<normalise clause>]

<unary query expression> ::=
 (<unary query expression>)
 | <table reference> { **COMPLEMENTATION**
 | **BOUNDARY**
 | **ENVELOPE [ALL]**
 | **OF (<reformat column list>)**
 | **[EXPANDING (<reformat column list >)]**
 | <table reference> **BUFFER [ALL]**
 | **OF (<reformat column list>)**
 | **WITHIN DISTANCE (<reformat column list>)**
 | **[EXPANDING (<reformat column list >)]**

<reformat clause> ::=
REFORMAT AS <reformat item>

<reformat item> ::=
FOLD [ALL] <reformat column list> [<reformat item>]
| UNFOLD [ALL] <reformat column list> [<reformat item>]

<normalise clause> ::=
NORMALISE ON [ALL] <reformat column list>

<reformat column list> ::=
 <reformat column> [{, <reformat column>}...]

<reformat column> ::=
 <column reference> | <unsigned integer>

Notes

- The keyword **EXPANDING** has been borrowed from [LD96].

B.3 LITERALS

<IXSQL literal> ::=
 <literal>
 | <IXSQL period literal>

<IXSQL period literal> ::=
 [<literal>, <literal>]


```

<literal> ::=
    <signed numeric literal>
    | <general literal>
<general literal> ::=
    <character string literal>
    | <national character string literal>
    | <bit string literal>
    | <hex string literal>
    | <datetime literal>
    | <interval literal>
    | <IXSQL spatial literal>
<IXSQL spatial literal> ::=
    <spatial object type> <spatial quanta string>
<spatial object type> ::=
    POINT
    | PLINE
    | LINE
    | PSURFACE
    | SURFACE
<spatial quanta string> ::=
    ‘<spatial quantum>[{'<spatial quantum>}...]’
<spatial quantum> ::=
    <spatial quantum type> <unsigned integer>
<spatial quantum type> ::=
    P
    | H
    | V
    | S

```

B.4 SEARCH CONDITIONS

```

<IXSQL predicate> ::=
    <predicate>
    | <IXSQL period predicate>
    | <IXSQL spatial predicate>
<IXSQL period predicate> ::=
    <IXSQL period binary predicate>

```

<IXSQL period binary predicate> ::=
 <IXSQL period value expression>
 <period binary predicate name>
 <IXSQL period value expression>

<period binary predicate name> ::=

before
 | meets
 | overlaps
 | lcovers
 | covers
 | rcovers
 | rcovered
 | covered
 | lcovered
 | roverlaps
 | met
 | after
 | psubinterv
 | subinterv
 | psupinterv
 | supinterv
 | overlaps
 | cp
 | merges
 | precedes
 | prequals
 | follows
 | folequals
 | adjacent
 | disjoint
 | surrounds

<IXSQL spatial predicate> ::=
 <IXSQL spatial binary predicate>
 | **<IXSQL spatial n-ary predicate>**

<IXSQL spatial binary predicate> ::=
 <IXSQL spatial value expression>
 <spatial binary predicate name>
 <IXSQL spatial value expression>

<spatial binary predicate name> ::=
 cp
 | **disjoint**
 | **surrounds**

<IXSQL spatial n-ary predicate> ::=
 is_point (<s IXSQL spatial value expression>)
 | **is_pure_line (<IXSQL spatial value expression>)**
 | **is_line (<IXSQL spatial value expression>)**
 | **is_pure_surface (<IXSQL spatial value expression>)**
 | **is_surface (<IXSQL spatial value expression>)**
 | **is_hybrid_surface (<IXSQL spatial value expression>)**
 | **is_simple (<IXSQL spatial value expression>)**
 | **is_circular (<IXSQL spatial value expression>)**
 | **conductive (<IXSQL spatial value expression>,
 < IXSQL spatial value expression>,
 < IXSQL spatial value expression>,
 < IXSQL spatial value expression>)**
 | **has_holes (<IXSQL spatial value expression>)**

Notes

- Both <comparison predicate>s (=, <, >, etc.) and <quantified comparison predicate>s (= SOME, > ANY, etc.) of SQL:1999 can be applied to pairs of <row value constructor>s containing some <value expression> of either a period or a spatial type.
- The majority of the <IXSQL period predicate>s have already been defined in [LM97].

B.5 VALUE EXPRESSIONS

<value expression> ::=
 <atomic value expression>
 | **<IXSQL period value expression>**

<atomic value expression> ::=
 <numeric value expression>
 | <string value expression>
 | <datetime value expression>
 | <interval value expression>
 | **<IXSQL spatial value expression>**

```

<IXSQL common primary> ::=
    <common primary>
    | <IXSQL period common primary>

<IXSQL period common primary > ::=
    start(<IXSQL period value expression>)
    | end(<IXSQL period value expression>)
    | middle(<IXSQL period value expression>)
    | topoint(<IXSQL period value expression>)
    | succ(<atomic value expression>, <numeric value expression>)

<IXSQL numeric primary> ::=
    <numeric primary>
    | <IXSQL period numeric primary>
    | <IXSQL spatial numeric primary>

<IXSQL period numeric primary> ::=
    ord(<atomic value expression>)
    | duration (<IXSQL period value expression>)
    | span (<atomic value expression>, <atomic value expression>)
    | distance (<atomic value expression>, <atomic value expression>)
    | windowno (<atomic value expression>,
                <numeric value expression>,
                <atomic value expression>)

<IXSQL spatial numeric primary> ::=
    ord(<IXSQL spatial value expression>)
    | h_coord(<IXSQL spatial value expression>)
    | v_coord(<IXSQL spatial value expression>)
    | distance(<IXSQL spatial value expression>,
               <IXSQL spatial value expression>)
    | greatest_distance(<IXSQL spatial value expression>,
                       <IXSQL spatial value expression>)
    | length(<IXSQL spatial value expression>)
    | area(<IXSQL spatial value expression>)

<IXSQL datetime primary> ::=
    <datetime primary>
    | <IXSQL period datetime primary>

<IXSQL period datetime primary> ::=
    min_date()
    | max_date()
    | form_instant(<numeric value expression>)

```

<IXSQL period value expression> ::=
to_interv(<atomic value expression>
| interv(<atomic value expression>, <atomic value expression>
| intervsect(<IXSQL period value expression>
<IXSQL period value expression>
| merge(<IXSQL period value expression>
<IXSQL period value expression>
| window(<atomic value expression>, <numeric value expression>
<numeric value expression>

<IXSQL spatial value expression> ::=
form_point (<numeric value expression>, <numeric value expression>
| to_point (<IXSQL spatial value expression>
| to_pure_line (<IXSQL spatial value expression>
| to_line (<IXSQL spatial value expression>
| to_pure_surface (<IXSQL spatial value expression>
| to_surface(<IXSQL spatial value expression>

Notes

- The majority of the functions for the manipulation of periods have been defined in [LM97].

APPENDIX C

IMPLEMENTATION

C.1 Introduction

In this appendix pseudocode is provided that implements predicate *conductive* and the relational algebra operations *Unfold* and *Fold*, when applied to spatial data. The following assumptions are made:

Internally, a spatial object is represented as a list in canonical form (see Definition 3.13).

It is recalled that $I_n = \{i \mid i \in I, 0 \leq n-1\}$. Use of this n value is made in the next section.

Some preliminary functions, used in Section C.2, are the following:

- Function *ord*(q) returns the ordinal number of a quantum q (Section 3.5).
- For a given integer k , the functions $P(k)$, $H(k)$, $V(k)$, $S(k)$ return, respectively, the quantum point, pure horizontal quantum line, pure vertical quantum line and pure quantum surface, whose ordinal number equals k (see relevant discussion in Section 3.5).
- For a quantum q , *type*(q) returns
 - ‘P’ if q is a point,
 - ‘H’ if q is a horizontal quantum line,
 - ‘V’ if q is a vertical quantum line,
 - ‘S’ if q is a quantum surface.

Finally, the following definition is given.

Definition C.1: Let s be a list of quanta and let $q_A, q_B \in s$. It is then defined that

$$q_A \text{ } q\textit{connected} \text{ } q_B \Leftrightarrow (\exists q_1, q_2, \dots, q_n \text{ in } s) \\ (q_A \cap q_1 \neq \emptyset \wedge \\ q_B \cap q_n \neq \emptyset \wedge \\ q_i \cap q_{i+1} \neq \emptyset, i = 1, 2, \dots, n-1).$$

It is said that q_A is *quantum connected* to q_B (recall relevant definition is Section 3.3).

It is easy to show that *qconnected* is an equivalence relation. The algorithm that determines the elements of each of the equivalence classes of such a relation is also well known. Use of it is made in algorithm `connected_equivalence_classes`, given in the next section.

C.2 Algorithms

The algorithms that implement predicate *conductive* and operations *Unfold* and *Fold* are given in a bottom-up fashion.

Function Name: `delete_element_from_list`

Input: An element e of any data type and an list s . The data type of the elements in s is the same as the data type of e .

Output: If e is not in s then the result is s , otherwise it is the list obtained by the elimination of element e from s .

Code: The code for this function is well known, therefore only its signature is given below.

```
Function delete_element_from_list (e:any,
                                  s:list(any)): list(any)
```

Function Name: `add_element_to_ordered_list`

Input: An element e of any data type and an ordered list s . The data type of the elements in s is the same as the data type of e .

Output: A new ordered list resulting from the addition of element e into s , at the proper place.

Code: The code for this function is well known, therefore only its signature is given.

```
Function add_element_to_ordered_list(e:any,
                                    s:list(any)): list(any)
```

Function Name: `exists_element_in_ordered_list`

Input: An element e of any data type and an ordered list s . The data type of the elements in s is the same as the data type of e .

Output: If the element e is in list s then the result is true, otherwise it is false.

Code: The code for this function is well known, therefore only its signature is given below.


```
Function exists_element_in_ordered_list (e:any,
                                         s:list(any)): boolean
```

Function Name: add_distinct_element_to_ordered_list

Input: An element e of any data type and an ordered list s . The data type of the elements in s is the same as the data type of e .

Output: If e is already in s the result is s , otherwise it is a new ordered list, resulting from the addition of e into s , at the proper place.

Code: The code for this function is well known, therefore only its signature is given below.

```
Function add_distinct_element_to_ordered_list (e:any,
                                             s:list(any)): list(any)
```

Function Name: delete_element_from_ordered_list

Input: An element e of any data type and an ordered list s . The data type of the elements in s is the same as the data type of e .

Output: If e is not in s then the result is s , otherwise it is the list obtained by the elimination of e from s .

Code: The code for this function is well known, therefore only its signature is given below.

```
Function delete_element_from_ordered_list (e:any,
                                          s:list(any)): list(any)
```

Function Name: add_non_contained_quantum_to_list

Input: A quantum q and an ordered list of quanta s in minimal canonical form.

Output: If there is a quantum q_i in s such that $q \subseteq q_i$ then the result is s , otherwise it is a new ordered list of quanta resulting from the addition of q into s , at the proper place. Besides, all the quanta q_j in s such that $q_j \subset q$, are eliminated from s .

Code:

```
Function add_non_contained_quantum_to_list (q:quantum,
                                           s:list(quantum)): list(quantum)
```

Var

```
result: list(quantum)
k: integer
```

Begin

```
result := s
k := ord(q)
Case type(q)
'P':
  If not(exists_element_in_ordered_list(H(k), result) or
        exists_element_in_ordered_list(H(k-1), result) or
        exists_element_in_ordered_list(V(k), result) or
        exists_element_in_ordered_list(V(k-n), result) or
        exists_element_in_ordered_list(S(k), result) or
```

```

        exists_element_in_ordered_list (S(k-1), result) or
        exists_element_in_ordered_list (S(k-n), result) or
        exists_element_in_ordered_list (S(k-n-1), result))
    Then
        result := add_distinct_element_to_ordered_list (q,
                                                       result)
    EndIf
'H':
    If not(exists_element_in_ordered_list(S(k), result) or
           exists_element_in_ordered_list(S(k-n), result))
    Then
        result := add_distinct_element_to_ordered_list (q,
                                                       result)
        result := delete_element_from_ordered_list(P(k),
                                                  result)
        result := delete_element_from_ordered_list(P(k+1),
                                                  result)
    EndIf
'V':
    If not(exists_element_in_ordered_list(S(k), result) or
           exists_element_in_ordered_list(S(k-1), result))
    Then
        result := add_distinct_element_to_ordered_list (q,
                                                       result)
        result := delete_element_from_ordered_list(P(k),
                                                  result)
        result := delete_element_from_ordered_list(P(k+n),
                                                  result)
    EndIf
'S':
    result := add_distinct_element_to_ordered_list (q,
                                                  result)
    result := delete_element_from_ordered_list(P(k),
                                                  result)
    result := delete_element_from_ordered_list(P(k+1),
                                                  result)
    result := delete_element_from_ordered_list(P(k+n),
                                                  result)
    result := delete_element_from_ordered_list(P(k+n+1),
                                                  result)
    result := delete_element_from_ordered_list(H(k),
                                                  result)
    result := delete_element_from_ordered_list(H(k+n),
                                                  result)
    result := delete_element_from_ordered_list(V(k),
                                                  result)
    result := delete_element_from_ordered_list(V(k+1),
                                                  result)
EndCase
add_non_contained_quantum_to_list:= result
End

```

Function Name: maxcform

Input: A spatial object g .

Output: The maximal canonical form of g , $\text{maxcform}(g)$, i.e. the ordered list of all the quanta contained in g .

Code:

```

Function maxcform(g:surface): list(quantum)

Var
result: list(quantum)
k: integer

Begin
  result := empty list
  For each q in g
    k := ord(q)
    result := add_distinct_element_to_ordered_list(q, result)
    Case type(q)
      'H':
        result := add_distinct_element_to_ordered_list(P(k), result)
        result := add_distinct_element_to_ordered_list(P(k+1), result)
      'V':
        result := add_distinct_element_to_ordered_list(P(k), result)
        result := add_distinct_element_to_ordered_list(P(k+n), result)
      'S':
        result := add_distinct_element_to_ordered_list(P(k), result)
        result := add_distinct_element_to_ordered_list(P(k+1), result)
        result := add_distinct_element_to_ordered_list(P(k+n), result)
        result := add_distinct_element_to_ordered_list(P(k+n+1), result)
        result := add_distinct_element_to_ordered_list(H(k), result)
        result := add_distinct_element_to_ordered_list(H(k+n), result)
        result := add_distinct_element_to_ordered_list(V(k), result)
        result := add_distinct_element_to_ordered_list(V(k+1), result)
    EndCase
  EndFor
  maxcform := result
End

```

Function Name: adjacent

Input: Two spatial objects g_1 and g_2 such that $g_1 \subset g_2$.

Output: An ordered list of quanta $\langle q_1, q_2, \dots, q_n \rangle$ such that for each $i = 1, 2, \dots, n$, $q_i \subseteq g_2$ and $q_i \not\subseteq g_1$ and $q_i \cap g_1 \neq \emptyset$.

Code:

```

Function adjacent (g1: surface, g2: surface): list(quantum)

Var
s: list(quantum)
q: quantum
result: list(quantum)

```

```

Begin
  result := empty list
  s := maxcform(g2)
  For each q in s
    If (q ⊄ g1) and (q ∩ g1 ≠ ∅) Then
      result := add_element_to_ordered_list(q, result)
    EndIf
  EndFor
  adjacent := result
End

```

Function Name: conductive

Input: Four spatial objects of data type surface, $g_1, g_2, g_{\text{CONDUCTOR}}, g_{\text{INSULATOR}}$

Output: It implements predicate *conductive*. Therefore, it evaluates to true iff there is a *path* contained in $g_{\text{CONDUCTOR}}$, from g_1 to g_2 , which does not cross $g_{\text{INSULATOR}}$ (see Section 3.4).

Code:

```

Function conductive (g1: surface, g2: surface,
                    gCONDUCTOR: surface,
                    gINSULATOR: surface ): boolean

Var
s: list(quantum)
q: quantum
g: surface

Begin
  If (g1 ⊆ gINSULATOR) or
    (g2 ⊆ gINSULATOR) or
    (g1 ∩ gCONDUCTOR = ∅) or
    (g2 ∩ gCONDUCTOR = ∅) Then conductive := false
  Else
    If (g1 ∩ g2 ∩ gCONDUCTOR ≠ ∅) and
      (g1 ∩ g2 ∩ gCONDUCTOR ⊄ gINSULATOR)
    Then conductive := true
    Else
      s := adjacent(g1, gCONDUCTOR)
      g := g1
      For each q in s
        g := add_non_contained_quantum_to_list(q, g)
      EndFor
      conductive := conductive (g, g2, gCONDUCTOR, gINSULATOR)
    EndIf
  EndIf
End

```


Function Name: select**Input:** A relation R and a well formed formula s.**Output:** A new relation resulting from the application of relational operation *Select*[s](R).**Code:** The code for this function is well known, therefore only its signature is given below.

```
Function select(s: string, R: relation): relation
```

Function Name: except**Input:** Two relations R and S**Output:** A new relation resulting from the application of relational operation *R Except S*.**Code:** The code for this function is well known, therefore only its signature is given below.

```
Function except(R: relation, S: relation): relation
```

Function Name: fold**Input:** A relation R, the list X of attribute names in the scheme of R and the name of an attribute G in the scheme of R.**Output:** A new relation resulting from the application of relational operation *Fold* to R on attribute G.**Code:**

```
Function fold(R: relation, X: list(attribute_name)
             G: attribute_name): relation
```

Var

```
F: relation
S: relation
R': relation
A: list(attribute_name)
s1: list(quantum)
s2: list(list(quantum))
t: tuple
t1:tuple
t2:tuple
s3:list(quantum)
```

Begin

```
R' := R
F := empty relation
A := delete_element_from_list(G, X)
while R' not empty
  t1 := first tuple in R'
  S := select("A = t1.A", R')
  R' := except(R', S)
  s1 := empty list
```

```
For each t in S
  For each q in t.G
    s1 := add_non_contained_quantu_to_list(q, s1)
  EndFor
EndFor
s2 := connected_equivalence_classes(s1)
For each s3 in s2
  t2 := (t1.A, s3)
  F := add_tuple_to_relation(t2, F)
EndFor
EndWhile
fold := F
End
```