LIBRARY
Michigan State
University

PLACE IN RETURN BOX
to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE

1/98 c:/CIRC/DateDue.p65-p.14

A DATA MODEL AND DESIGN PRINCIPLES FOR TEMPORAL DATABASES WITH HOMOGENEOUS AND NON-HOMOGENEOUS DATA

By

Jaruloj Eamsiri Chongstitvatana

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1998

ABSTRACT

A DATA MODEL AND DESIGN PRINCIPLES FOR TEMPORAL DATABASES WITH HOMOGENEOUS AND NON-HOMOGENEOUS DATA

By

Jaruloj Eamsiri Chongstitvatana

Existing data models and design principles for temporal databases are based on the assumption that a temporal data is associated with an interval as well as its sub-intervals. This type of temporal data is called *homogeneous data*. A temporal data which is associated with an interval, but not its sub-intervals, is called a *non-homogeneous data*. Existing data models cannot capture non-homogeneous data accurately, and the existing design principles are not applicable in temporal databases that contain non-homogeneous data. In this dissertation, the relational data model is extended to support both homogeneous and non-homogeneous data. A design principle which avoids inconsistency in temporal databases, that contain homogeneous data as well as non-homogeneous data, is studied.

In the proposed extension of the relational data model, temporal relations are classified into two types; *property relations* and *representative relations*. A tuple in a property relation is associated with the valid time and its sub-intervals, while a tuple in a representative relation is associated with only the whole interval of its valid time. Thus, the valid time in a property relation is decomposable, but the valid time in a representative relation is not. Based on this characteristics, the valid time in a property relation and that in a representative relation cannot be used in the same manner. In the extension of relational algebra for temporal relations, the calculation of the valid time in a relation, created by relational operators, is determined by the types of the temporal relations. Thus, it guarantees proper use of the valid time.

A type of inconsistency, called *P-inconsistency*, can occur in temporal databases with homogeneous and non-homogeneous data. A normal form, called P-consistency Normal Form (*PCNF*), which avoids P-inconsistency, is proposed in this dissertation. PCNF is based on types of attributes, functional dependencies, and property dependencies (*P-dependencies*), in temporal relations. A normalization algorithm for PCNF is presented. However, it does not always give the minimum number of normalized relations. We prove that finding a decomposition that gives the minimum number of normalized relations is an NP-complete problem. We have also proven that the problem of finding equivalent sets of attributes under a set of dependencies is an NP-complete problem. This problem is a more general problem, and is the basis of the proof that finding a decomposition that gives the minimum number of normalized relations is an NP-complete problem. Finally, a heuristic that gives a minimum number of normalized relations is provided.

Copyright by
JARULOJ EAMSIRI CHONGSTITVATANA
1998

To my parents.

ACKNOWLEDGMENTS

To my advisor, Dr. Sakti Pramanik, I wish to express my sincere gratitude for his valuable guidance, patience, and support. I value the many hours of discussion with him throughout the past years of my graduate study. I would like to thank Dr. Jon Sticklen for introducing me to the area of temporal reasoning. I appreciate the help from members of my guidance committee, Dr. Walter Esselman and Dr. John Forsyth. I would also like to thank Dr. Eric Torng for his time and suggestions.

I also wish to acknowledge my family: to my parents, Chaab and Vannee Eamsiri, and my sisters and brother, who always wholeheartedly gave me encouragement and support. No words can fully express how grateful I feel towards each of them. To my husband, Dr. Prabhas Chongstitvatana, and my daughter, Vitaranee Chongstitvatana, I cherish our family life, together and apart. Without you all, I could never have become what I am now.

Table of Content

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Temporal Databases

The importance of temporal aspects of information in databases has been recognized in many research [11, 13, 26, 28]. A database that supports a temporal aspect of information is called a *temporal database*. Temporal aspects of information have been studied in many areas, such as linguistics [14, 45, 57, 61, 87], logic [66, 67], and artificial intelligence [3, 16, 58, 73]. Most of the work on temporal databases is based on the relational model [27, 28]. Extensions of deductive databases to support temporal data have also been studied [1, 7, 19, 20, 21, 22, 63]. Temporal databases in object-oriented data model have recently been addressed [12, 18, 30, 34, 37, 65, 69, 84, 86].

Temporal aspect of information is classified into *valid time, transaction time*, and *user-defined time* [76]. The valid time indicates when a data is true in the real world. For example, the valid time of the temporal data, "John is a manager", is the interval beginning when John takes the position as a manager and ending when John leaves the position. The transaction time of a temporal data is the time between the insertion and the deletion of the data. The transaction time indicates when a data is considered to be true in the database, which is not necessarily the same as the time when it is actually true in the real world. For example, the transaction time of the fact "John is a manager" is the interval beginning when the fact is inserted in the database, and ending when the data it is removed from the database. The data is not necessarily inserted exactly when John takes the position, nor removed exactly when John leaves the position. The user-defined time is any arbitrary time whose semantics is defined by users. For example, an employee's birthday is a user-defined time. A *valid-time database* supports valid time, a *transaction-time database* supports transaction time, and a *bi-temporal database* supports both valid time and transaction time [46]. Most of the research on temporal databases focus on valid time [5, 18, 19, 21, 23, 24, 26, 30, 39, 40, 41, 52, 53, 54, 62, 70, 71, 72, 73, 79, 83, 84, 86]. A few works address both valid time and transaction time [11, 40, 59, 74, 75].

Research issues in temporal databases include:

1. The extension of a data model to support time [5, 6, 12, 18, 19, 23, 30, 37, 39, 41, 40, 50, 53, 65, 69, 79, 86],
2. Query languages for temporal databases [5, 7, 34, 39, 40, 55, 62, 70, 74],
3. Design principles for temporal databases [48, 52, 62, 71, 82, 83, 84],
4. Imprecise information in temporal databases [15, 32, 33].

Some of these issues are discussed briefly in this section. They are also examined further in Chapter 2.

1.1.1 Data Models for Temporal Data

The extensions of various data models to support time have been studied. For the extensions of the relational data model, there are two approaches to incorporate time. One of these approaches is to attach time to each tuple, which is called *tuple-timestamping*. This approach has been used in many data models [11, 47, 49, 53, 59, 62, 70, 72, 74, 75]. The other approach is to attach time to each attribute, which is called *attribute-timestamping*. This approach is adopted in some data models [23, 24, 39, 41, 79]. In the attribute-timestamping models, values of attributes associated with the same key are grouped in the same tuple. Therefore, a data model with attribute-timestamping is sometimes called a *temporally-grouped* data model, while a data model with tuple-timestamping is sometimes called a *temporally-ungrouped* data model. Examples of temporal relations in a temporally-grouped data model and a temporally-ungrouped data model are shown in Figure 1.1 (a) and (b), respectively. The difference between these two approaches is addressed [25]. An advantage of a temporally-ungrouped model is simplicity. A temporal relation in a temporally-ungrouped model can be represented in a 1NF relation. On the other hand, a temporal relation in a temporally-grouped data model cannot be represented in a first normal form (1NF) relation. However, some queries are meaningful in a temporally-grouped data model, but not in a temporally-ungrouped data model. Consider the query for the history of all employee's address, disregarding the employee's names. This query on the temporally-grouped relation in Figure 1.1 (a) results in two tuples; <[[*1980, 1987*]: *123 Oak St., Lansing, MI*], [[*1988, 1995*]: *45 Shore Rd., Chicago, IL*], [[*1996, 1997*]: *67 5th St., New York, NY*]> and <[[1988, *1990*]: *123*

Oak St., Lansing, MI], [[*1991,1997*]: *90 Palm Rd., Miami, FL*]>. Each of these tuples is the history of each employee's address. However, the same query on the temporally-ungrouped relation *Address* in Figure 1.1 (b) is not meaningful because the relationship among addresses of one employee is lost. Furthermore, John's address during 1980-1987 and Mary's address during 1988-1990, which are both "*123 Oak St. Lansing MI*", are combined as <*123 Oak St. Lansing MI, 1980-1990*>.

Name	*Address*
[[*1980, 1997*]: *John*]	[[*1980, 1987*]: *123 Oak St., Lansing, MI*], [[*1988, 1995*]: *45 Shore Rd., Chicago, IL*], [[*1996, 1997*]: *67 5th St., New York, NY*]
[[*1980, 1997*]: *Mary*]	[[*1988, 1990*]: *123 Oak St., Lansing, MI*], [[*1991, 1997*]: *90 Palm Rd,. Miami, FL*]

(a) A relation *Address* in a temporally-grouped data model.

Name	*Address*	*Time*
John	*123 Oak St., Lansing, MI*	[*1980, 1987*]
John	*45 Shore Rd., Chicago, IL*	[*1988, 1995*]
John	*67 5th St., New York, NY*	[*1996, 1997*]
Mary	*123 Oak St., Lansing, MI*	[*1988, 1990*]
Mary	*90 Palm Rd., Miami, FL*	[*1991, 1997*]

(b) A relation *Address* in a temporally-ungrouped data model.

Figure 1.1: A relation *Address* in a temporally-grouped and a temporally-ungrouped data models.

TEMPLOG [1, 7], Datalog$_{1S}$ [19, 20, 21, 22], and Temporal DATALOG [63] are extensions of a deductive database Datalog to support time. In Datalog$_{1S}$, time is added as another explicit argument in each predicate, and the next operator (+1) is provided for time. In TEMPLOG and Temporal DATALOG, time is not an explicit argument, but time is referred to by temporal operators. Extensions of the object-oriented data model are also studied [12, 18, 30, 34, 37, 65, 69, 86].

1.1.2 Query Languages for Temporal Databases

Many query languages proposed for temporal databases are based on SQL [17] and Quel [78]. SQL is extended to support temporally-ungrouped models [62, 70, 75], and a temporally-grouped model [5]. Quel is also extended to support a temporally-ungrouped model [74]. The completeness of query languages are defined by Clifford, Croker and Tuzhilin [25]. The completeness for temporally-grouped and temporally-ungrouped models are different because some queries are meaningful in temporally-grouped models, but not in temporally-ungrouped models.

1.1.3 Design Principles for temporal Databases

Design principles proposed for temporal databases can be classified into two types. One is the extension of traditional design principles, such as the third normal form (3NF), for the temporal relational data model [48]. The other type of design principle focuses on redundancy in temporal data [11, 52, 62, 72, 82, 83]. The elimination of redundancy is based on the assumption of the characteristics of temporal data.

1.2 Types of Temporal Data

The assumption about the characteristic of temporal data adopted in most temporal data models is very simple. A temporal data which is applied to an interval is assumed to be applied to any sub-interval. However, this assumption is not applicable for many temporal data. There are many studies on other characteristics of temporal data in linguistics [14, 45, 57, 87], philosophy [61], and artificial intelligence (AI) [3, 58, 73].

In McDermott's study [58], facts, events, and fluents are defined. A fact or an event is true over an interval, and a fluent's value changes over time. In Allen's temporal logic [3], temporal data are classified into properties, events, and processes. If a property is true for an interval, it is true for any sub-intervals. An event is true for the whole interval in which it occurs but not for its sub-intervals. A process is true for some sub-intervals in which it occurs. These two classifications are used in automatic planning, and, thus, they are also influenced by types of temporal data which are of interest for planning. Shoham proposed a temporal logic system as a basis for general classification of temporal data [73].

These complex characteristics of temporal data addressed in these works are also present in temporal databases. However, these types of temporal data have not been addressed in temporal databases. This problem is addressed in the next section.

1.3 Problem Description

1.3.1 Motivation

In existing temporal data models, it is assumed that a temporal data is associated with every point in an interval. For example, in Clifford and Warren's historical database

(HDB) [26], a temporal data is associated with a *state*, which is a point of reference in time. A data is associated with a set of states if and only if it is associated with each state in the set. In Gadia's data model [39], time is represented by a finite union of intervals. Thus, an interval of time is equivalent to a union of its sub-intervals. The implication of this representation is that a temporal data associated with an interval is associated with its sub-intervals, as well. Similar assumptions are made in many temporal data models [5], [23], [41], [54]. This type of temporal data is called *homogeneous data* [68]. For example, "*John is married*", is a homogeneous data. Data in transaction-time databases are always homogeneous. If a data is in a transaction-time database during an interval, it is in the database during any sub-intervals. However, the assumption is not always true for valid-time intervals. As mentioned earlier, in many studies of temporal data, there are temporal data which do not comply with this assumption. For example, "*Mary walks from home to school*", is an event which is true for an interval $[t_1, t_2]$, where t_1 is the time Mary leaves home and t_2 is the time she arrives at school. However, it is not true for an interval $[t_1, t_3]$, where $t_1<t_3<t_2$, because Mary does not yet arrive at school at t_3. Data which are created by aggregation over time do not satisfy the assumption. For example, the total precipitation and the highest temperature over an interval are created by the aggregation sum and maximum, respectively. Both the total precipitation and the highest temperature are applied for the whole interval, but not its sub-intervals. We call this type of temporal data *non-homogeneous data*.

Non-homogeneous data cannot be represented accurately in existing temporal data models. Consider the representation of the highest temperature in the data model in

HDB. Suppose the highest temperature in Washington D.C. is 98°F for 1997, and is 50°F for January 1997. Let a state represent a month. The year 1997 is represented by a set *S* of states, which represent the months in 1997. Since the highest temperature, 98°F, is associated with *S* in this data model, it is associated with each state in *S*. That is, it is also associated with the month of January 1997. However, the highest temperature in Washington D.C. during January 1997 is 50°F, not 98°F. Therefore, the highest temperature is not represented correctly in this data model.

1.3.2 STATEMENT OF THE PROBLEM

In existing temporal data models, temporal data are assumed to be homogeneous. However, it is essential to capture non-homogeneous data in temporal databases. In this dissertation, we address the following problems:

1. Extend the relational data model to support both homogeneous and non-homogeneous data.

2. Develop a design principle that avoids inconsistency in temporal databases containing homogeneous and non-homogeneous data.

In this work, we focus only on valid-time databases. Tuple-timestamping is adopted in the extension of the relational data model. Relational operators are extended for temporal relations. These operators on temporal relations are defined, based on characteristics of the temporal relations.

Furthermore, inconsistency in temporal relations is identified. A normal form that avoids the inconsistency is defined. The normal form is based on characteristics of temporal data and two types of data dependencies in temporal relations.

1.4 Organization of Dissertation

This dissertation is composed of 8 chapters. Previous works on temporal data models, query languages, and design principles for temporal databases are presented in Chapter 2. In Chapter 3, characteristics of temporal data are studied. Attributes and relations in temporal databases are classified according to their characteristics. Based on types of temporal relations defined in Chapter 3, an extension of the relational data model is presented in Chapter 4. An extension of Datalog, a deductive database, is presented in Chapter 5. In Chapter 6, inconsistency in a temporal database with homogeneous and non-homogeneous data is identified. Two types of data dependencies on temporal data are discussed. Based on types of temporal data and types of data dependencies, a design principle which avoids the anomaly is proposed. In Chapter 7, the algebra presented in Chapter 4 is extended to handle imprecise valid time. Finally, concluding remarks are presented in Chapter 8.

Chapter 2

BACKGROUND

In this chapter, related works which are the background for this dissertation, are examined. In Section 2.1, existing temporal databases are discussed, and we examine why these temporal databases cannot capture non-homogeneous data. In Section 2.2, design principles for temporal databases are discussed. We also show why these design principles are not applicable for a temporal database which contains non-homogeneous data. Finally, in Section 2.3, we show that non-homogeneous data are an essential part of temporal data, and are studied in other areas, such as AI.

2.1 Data Models and Query Languages for Homogeneous Data

The majority of the works on temporal databases are based on the relational data model. In this section, temporal relational data models and query languages, and temporal deductive databases are examined.

2.1.1 TEMPORAL RELATIONAL DATABASES

There are two approaches for extending the relational data model to support temporal data. Time can be associated with each attribute, or each tuple. This is a major difference among these data models. As a result of different timestamping, operations on temporal relations are defined differently in these data models. A similarity among many temporal data models is the assumption that temporal data is homogeneous. These data models cannot capture non-homogeneous data accurately. A few temporal data models are discussed here. A more comprehensive survey can be found in Jensen, Snodgrass, and Soo's work [47].

Clifford and Warren's Historical Database

Clifford and Warren aim to define the semantics of time in temporal databases [26]. A data model for temporal databases, called historical database (*HDB*), is proposed. The importance of the incorporation of temporal semantics in a temporal data model is recognized. A temporal data is assumed to be true at any point in its valid time. *STATE* is an attribute which represents a point of time. *STATE* is associated with each tuple, which represents a temporal data. This representation of temporal data cannot capture non-homogeneous data. If a non-homogeneous data is true for an interval, which contains more than one *STATE*, it is not necessarily true for each *STATE*. However, in HDB, a temporal data is associated with a *STATE*. As a result, temporal data cannot be associated with an interval, and not with each *STATE* in the interval.

Ariav's Temporally Oriented Data Model

Ariav proposes a temporally oriented data model (TODM) [5], which addresses the transaction time (called a recording time in [5]). TOSQL, which is an extension of SQL, is defined for TODM. Clauses, such as *WHILE*, *AT*, etc., are included in TOSQL, and they specify the time of relations which can be involved in a *SELECT* statement. Since the temporal aspect of data in this work is the transaction time, non-homogeneous data have not been addressed in this work.

Gadia's Homogeneous Relational Model

Homogeneous relational model (HRM) [39] is an attribute-timestamping model. The value of an attribute is a function from its *temporal domain* to values. The temporal domain of an attribute in a tuple is the time during which the value of the attribute is defined in the tuple. The temporal domains of all attributes in a tuple must be the same (hence, homogeneous). This restriction aims to avoid null value for an attribute at a time. Later, this restriction is relaxed [41]. Relational operators $\cup$, $\cap$, $-$, Π, σ, and $\bowtie$ on temporal relations are defined similar to traditional relational operators.

In HRM, a temporal data associated with an interval is equivalent to the temporal data associated with the union of its sub-intervals. This implies that a temporal data is associated with an interval and its sub-intervals. Thus, non-homogeneous data cannot be captured in HRM.

Clifford and Croker's Historical Relational Data Model

Similar to Gadia's HRM, historical relational data model (HRDM) [23, 24] is an attribute-timestamping model. The difference between these two models is the concept of *mergable tuples*. Two tuples with the same scheme are mergable if their valid times overlap and the values of all attributes in the overlapping lifespans are equal. Based on mergable tuples, operators $\cup_o$, $\cap_o$, and $-_o$, which are the extension of $\cup$, $\cap$, and $-$, are defined. The concept of mergable tuples is based on the assumption that a temporal data is true for all sub-intervals of its valid time, i.e. homogeneous. Therefore, the operators $\cup_o$, $\cap_o$ and $-_o$ are not applicable for non-homogeneous data.

Tansel's Model

Tansel presents an attribute-timestamping model, which allows set values for attributes [24, 79]. This data model allows nested relations. Thus, relations in this data model are clearly not in 1NF. Relational algebra for temporal relations are defined. Operators *Pack*, *Unpack*, *Decomposition*, and *Formation* are used to transform these non-1NF relations into 1NF relations. An operator *Slice* restricts the valid time of one attribute in each tuple by the valid time of another attribute. This operator is based on the assumption that the value of an attribute is associated with an interval as well as its sub-intervals. Therefore, the algebra is not applied to non-homogeneous data.

Snodgrass's TQuel

TQuel [74] is an extension of Quel, which addresses both valid time and transaction time. Valid time and transaction time are associated with each tuple. However, the difference between homogeneous and non-homogeneous data is not addressed in TQuel.

Navathe and Ahmed's TSQL

TOSQL [62] is an extension of SQL, for a tuple-timestamping temporal data model. The valid time is represented by two end points, T_S and T_E. Additional features in the *SELECT* statement include the *WHEN* clause, which specifies the valid time of relations in the statement, and the *TIME SLICE* clause, which restricts temporal data involved in the query by the valid time. Similar to the operator *Slice* [24], the *TIME SLICE* clause is not applicable to non-homogeneous data. Thus, TSQL is not applicable for temporal databases that contain non-homogeneous data.

Lorentzos's Model

Lorentzos presents a tuple-timestamping data model for temporal databases [52, 53]. The *Fold* operator and the *Unfold* operator are defined for temporal relations. The *Fold* operator transforms a temporal relation in which the valid time is a point into a temporal relation in which the valid time is an interval. The *Unfold* operator transforms a temporal relation in which the valid time is an interval into a temporal relation in which the valid time is a point. These operators are based on the assumption that a temporal data is associated with each point in its valid time. Thus, they are not applicable for non-homogeneous data.

The data model for temporal data is extended to support interval data, and a query language, IXSQL, is defined for interval data [54, 55]. Time is considered a one-dimensional interval. Spatial objects, such as area, are considered two-dimensional intervals.

Jensen, Soo and Snodgrass's Bi-temporal Conceptual Data Model

The bi-temporal conceptual data model (BCDM) is proposed as a conceptual model which unifies different temporal data models [49, 50]. A tuple is associated with a set of *bi-temporal chronons*. The domain of bi-temporal chronons is the Cartesian product of the domains of valid time and transaction time. The domains of valid time and transaction time are sets of time points. From the data model, it is implied that a temporal data is associated with each point in its valid time. Thus, non-homogeneous data cannot be captured in BCDM.

TSQL2

TSQL2 [75] is an extension of SQL-92 [29, 60] to support both valid time and transaction time. It is a proposed standard query language for temporal databases. TSQL2 supports multiple granularities of time, multiple calendars, temporal aggregates and imprecise valid time. However, the difference between homogeneous and non-homogeneous data is not addressed here.

2.1.2 TEMPORAL DEDUCTIVE DATABASES

Similar to a deductive database, a temporal deductive database is composed of an extensional database (EDB) and an intensional database (IDB). Temporal operators O (next) for TEMPLOG, *+1* for Datalog$_{1S}$, and `next` for Temporal DATALOG are allowed in IDB rules. These operators in IDB rules can recursively define infinite relations. Therefore, infinite relations can be captured in these temporal deductive databases. On the other hand, relations in a temporal relational data model are always finite. In this respect, these temporal deductive databases can capture relations which cannot be captured in temporal relational databases [7, 8].

In TEMPLOG, Datalog$_{1s}$, and Temporal DATALOG, a point of time is associated with a temporal data. A temporal data is, therefore, associated with each point in an interval. As a result, non-homogeneous data cannot be captured in these temporal databases.

Following, various design principles for temporal relational databases are examined.

2.2 Design Principles for Temporal Databases with Homogeneous Data

Design principles for temporal relational databases have been investigated in many studies. Some of these design principles aim to avoid redundancies caused by the nature of temporal data. Examples of these design principles are first temporal normal form [72], time normal form [62], P and Q normal forms [52], and two different T3NF's [82, 83]. Others aim to avoid the same type of anomaly addressed in traditional relational databases. An example of this type of design principle is the temporal third normal form (T3NF) [48]. These normal forms are based on the characteristic of homogeneous data, and they are not applicable for temporal databases that contain both homogeneous and non-homogeneous data. Here, normal forms for temporal relational databases are examined.

2.2.1 First Temporal Normal Form

The first temporal normal form (1TNF) aims to avoid redundancy in temporal relations. 1TNF does not allow two tuples with the same key and overlapping valid time. It is assumed that such tuples can be combined into one tuple. For example, consider a relation *EmpDept(Name, Dept, T)*, which contains the tuples <*Mary*, *Advertising*, [*1990*, *1994*]> and <*Mary*, *Advertising*, [*1993*, *1996*]>. These two tuples are redundant because they can be combined as <*Mary*, *Advertising*, [*1990*, *1996*]>. Redundancy addressed in 1TNF is based on the characteristic of homogeneous data. Thus, it is not applicable when a database contains non-homogeneous data.

2.2.2 Time Normal Form

The time normal form (TNF) avoids redundancy caused by two attributes in a relation, which do not change values at the same time. For example, in the relation *EmpRecord(Name, Addr, Dept, T)* shown in Figure 2.1 (a), the values of *Addr* and *Dept* are not necessarily changed at the same time. When the value of *Dept* for an employee is changed and the value of *Addr* remains the same, the value of *Dept* is duplicated in another tuple. Thus, it is redundant. In Figure 2.1 (a), John's address, "*123 Oak St., Lansing, MI*", is duplicated. To avoid the redundancy, the relation can be decomposed into TNF relations, *EmpAddr(Name, Addr, T)* and *EmpDept(Name, Dept, T)*, as shown in Figure 2.1(b) and (c).

TNF does not always avoid redundancy when a relation contains non-homogeneous data.

If the same value of a non-homogeneous data is associated with adjacent intervals, it does not imply that the same value is associated with the union of the intervals.

2.2.3 P and Q Normal Forms

The P normal form (PNF) avoids redundancy by preventing tuples with overlapping or adjacent valid time. A relation is in Q normal form (QNF) if it is in PNF and there is only one non-prime attribute in the relation. This also avoids duplicate values when non-prime attributes do not change values at the same time. However, a relation can be decomposed unnecessarily.

Name	*Addr*	*Dept*	*T*
John	*123 Oak St., Lansing, MI*	*Marketing*	*[1993, 1995]*
John	*123 Oak St., Lansing, MI*	*Purchasing*	*[1996, 1997]*

(a) The relation *EmpRecord.*

Name	*Addr*	*T*
John	*123 Oak St., Lansing, MI*	*[1993, 1997]*

(b) The relation *EmpAddr.*

Name	*Dept*	*T*
John	*Marketing*	*[1993, 1995]*
John	*Purchasing*	*[1996, 1997]*

(c) The relation *EmpDept.*

Figure 2.1: The decomposition of a relation *EmpRecord* into relations in TNF.

2.2.4 Temporal 3NF

The temporal 3NF (T3NF) [48] is an extension of 3NF, which is based on temporal functional dependencies. The temporal functional dependency is an extension of the traditional functional dependency. Similar to 3NF, T3NF avoids the anomaly caused by transitive dependencies. In this work, temporal data is assumed to be associated with every point in its valid time. Thus, temporal functional dependency and T3NF are not applicable when non-homogeneous data are involved.

2.2.5 WIJSEN'S T3NF

Dynamic and temporal functional dependencies (DFD and TFD) are defined by Wijsen [83]. A TFD X G Y means, at any time, the value of X determines the value of Y. A DFD X N Y means the value change of X implies the value change of Y. T3NF is based on DFD, TFD, and traditional functional dependencies. Similar to 3NF, T3NF avoids transitive dependencies.

TFD and DFD are defined on data which are true at each point of time. For non-homogeneous data, it is not possible to derive data which are true at each point of time from data which are true for an interval. Thus, TFD and DFD cannot capture dependencies involving non-homogeneous data.

2.2.6 WANG ET AL'S T3NF

A temporal functional dependency is defined based on the granularity of time in the dependency [82]. The values of attributes in a dependency do not change within a unit of time, which is defined by the granularity. T3NF is an extension of 3NF and thus avoids transitive dependency. Furthermore, it avoids redundancy by preventing two temporal functional dependencies such that the time unit in one dependency contains more than one time unit in another dependency. For example, consider a relation *Payroll(Employee, Position, Dept, T)* shown in Figure 2.2 (a). *Employee* determines *Position* and the value of *Position* does not change with in a month. *Employee* determines *Dept* and the value of *Department* does not change within a year. That is, the granularity of the dependency *Employee* $\rightarrow$ *Position* is a month, and the granularity of the dependency *Employee* $\rightarrow$ *Dept* is a year. *Dept* in the second tuple is redundant because, for each employee, *Dept* remains the same for every month in a year. The relation can be decomposed into relations in T3NF, as shown in Figure 2.2 (b) and (c).

This temporal functional dependency cannot capture functional dependencies involving non-homogeneous data. Thus, T3NF is not applicable in temporal databases that contain non-homogeneous data.

Employee	***Position***	***Dept***	***T***
Smith	sales	toys	1/96-6/96
Smith	supervisor	toys	7/96-12/96

(a) a relation *Payroll.*

Employee	***Position***	***T***
Smith	sales	1/96-6/96
Smith	supervisor	7/96-12/96

(b) A relation *EPosition.*

Employee	***Dept***	***T***
Smith	toys	1/96-12/96

(c) A relation *EDept.*

Figure 2.2: The decomposition of a relation *Payroll* into T3NF.

2.3 Homogeneous and Non-homogeneous Data in Artificial Intelligence

In this section, two frameworks for the classification of temporal data in AI are discussed. The first one is Allen's temporal logic [3], which is intended for the reasoning about actions. The second one is Shoham's interval logic [73], which is a general framework for classifications of temporal data. We will show that both homogeneous and non-homogeneous data are an important foundation in reasoning about temporal data. Therefore, non-homogeneous data also needs to be addressed in temporal databases.

2.3.1 Allen's Temporal Logic

Allen proposes a temporal logic for reasoning about action and time [3]. This work addresses non-activity actions (e.g. standing, sleeping), actions that are not easily decomposed (e.g. hiding), and simultaneous actions. These types of actions cannot be captured in prior work, such as case grammar [38] and situation calculus [56]. In Allen's temporal logic, static and dynamic aspects of the world are described in temporally-qualified assertions or *temporal facts.* Temporal facts are classified into properties and occurrences, which are further classified into events and processes. A *property* (e.g., John is a manager) is true for an interval and its sub-intervals. An *event* (e.g., John ran 10 miles) is true for the whole interval, but not during any sub-interval. If a *process* (e.g., John was building a boat) is true for an interval, it is true for some sub-interval. Non-activity actions are captured by events which do not cause any change. Actions that are not easily decomposed and simultaneous actions are captured through the relationships between temporal facts, such as, causality, belief, and intention, which are pre-defined in this logic.

In this classification, properties are homogeneous data while events and processes are non-homogeneous data. All three types of temporal data are essential in the reasoning process.

2.3.2 SHOHAM'S INTERVAL LOGIC

It is argued [73] that a fixed classification of temporal facts [3, 58] is not appropriate for all applications. *Interval logic* is proposed as a framework for classifications of temporal facts. Different types of temporal facts can be defined through interval logic. A proposition is *downward-hereditary* if the proposition which is true for an interval is true for its sub-intervals. For example, the proposition, "John is married", is downward-hereditary. If John is married for an interval, he is married for any of its sub-intervals. A proposition is *upward-hereditary* if the proposition which is true for all sub-intervals is also true for the interval. For example, the proposition, "the average speed of a plane is 3,000 miles/hour", is upward-hereditary. If the average speed of a plane is 3,000 miles/hour for two intervals, the average speed for the union of the intervals is also 3,000 miles/hour. Different classifications can also be created based on the interval logic. Properties, events, and processes can also be captured in interval logic.

In this framework, a temporal data can be neither upward-hereditary nor downward-hereditary. This type of temporal data is non-homogeneous, and needs to be addressed in the classification.

2.4 Conclusions

In this chapter, three areas of related work are discussed. First, extensions of the relational databases and deductive databases to support temporal data are discussed in Section 2.1. In these temporal databases, temporal data are assumed to be homogeneous data, and non-homogeneous data cannot be captured accurately. In Section 2.2, normal forms defined for temporal relations are examined. Most of these normal forms aim to avoid redundancy caused by the characteristic of homogeneous data. In these works, a temporal data is assumed to be associated with every sub-interval of its valid time. These normal forms are not applicable when non-homogeneous data are involved. Finally, in Section 2.3, characteristics of temporal data which are addressed in artificial intelligence are examined. Some of these temporal data are non-homogeneous data, and they also need to be considered in temporal databases.

Chapter 3

CHARACTERISTICS OF TEMPORAL DATA IN DATABASES

3.1 Introduction

Characteristics of temporal data is an important basis for temporal data models, queries, and design principles. Many research on temporal databases [5, 22, 23, 24, 26, 39, 41, 46, 53, 62, 74] are based on the assumption that, if a temporal data is true for an interval, it is true at any point within the interval. For example, if John is single during 1970-1997, he is single on January 1, 1996. However, this assumption is not applicable for some temporal data. For example, if the total precipitation at the White House in January 1997 is 4 inches, the total precipitation at the White House on January 1, 1997 is not 4 inches. Existing temporal data models cannot capture this type of temporal data accurately.

When a temporal database contains this type of temporal data, an additional complication is present. Similar queries on different types of temporal data cannot be expressed by the same expression. For example, the queries for employee's marital status and total precipitation in 1997 are different. If the marital status is valid for an interval t which intersects the year 1997, it is also valid for a sub-interval of 1997. Thus, the marital status whose valid time intersects with the interval 1997 are selected. On the other hand, if the total precipitation is valid for an interval *t* which intersects the year 1997, it is not valid for a sub-interval of 1997. Thus, only tuples in which valid time is contained in or is equal to the interval 1997 are selected. The difference between these two queries is the result of the difference between the characteristics of the marital status and the total precipitation.

Furthermore, existing normal forms for temporal relations [11, 48, 52, 62, 72, 83] are also based on the assumption that a temporal data is associated with every point in an interval. They are not applicable when a temporal database contains temporal data that violate this assumption.

Characteristics of temporal data are studied in many areas, such as philosophy, linguistics, and AI. Some are examined in Section 2.3. Some of these characteristics are useful in database applications. In this chapter, characteristics of temporal data are studied. First, at the attribute level, the characteristics of different types of time-varying attributes, i.e. homogeneous and non-homogeneous attributes, are studied in Section 3.2. Then, the concept of *hereditary* [73] is extended in Section 3.3. Based on hereditary, two types of temporal relations, i.e. property and representative relations, are defined in Section 3.4. The difference between these two types of temporal relations is an important basis of this work. Moreover, these characteristics are also present in the spatial domain, and are discussed in Section 3.5. Section 3.6 provides the conclusion.

3.2 Homogeneous and Non-homogeneous Attributes

In temporal relations, the relationship between time-varying attributes and the valid time can be different. For example, consider the attributes *InterestRate* and *MinimumBalance*, which represent the interest rate and the lowest balance for an account during an interval, respectively. If the interest rate during the whole year of 1996 is 4%, the interest rate for any sub-interval of 1996 is also 4%. That is, the value of the attribute *InterestRate* is associated with an interval and its sub-intervals. On the other hand, if the lowest balance for 1996 is $500, the lowest balance for a sub-interval of 1996 is not necessarily $500. That is, the value of the attribute *MinimumBalance* is associated with the whole interval but not its sub-intervals. Based on this difference, time-varying attributes can be classified into two categories: homogeneous and non-homogeneous. A time-varying attribute is homogeneous if the value of the attribute is associated with an interval as well as its sub-intervals. A time-varying attribute is non-homogeneous if the value of the attribute is associated with the whole interval but not its sub-intervals. *InterestRate* is a homogeneous attribute, and *MinimumBalance* is a non-homogeneous attribute. The same concept also applies to sets of attributes. Homogeneous and non-homogeneous sets of attributes are defined in Definition 3.1 and Definition 3.2, respectively.

Definition 3.1: Homogeneous sets of attributes

A set of time-varying attributes *X* is ***homogeneous*** if the value of *X* is associated with an interval as well as its sub-intervals.

Definition 3.2: Non-homogeneous sets of attributes

A set of time-varying attributes X is *non-homogeneous* if the value of X is associated with an interval, but not with its sub-intervals.

The set of attributes {*Acc#*, *InterestRate*} is homogeneous because the same values of both *Acc#* and *InterestRate* are associated with an interval as well as its sub-intervals. For the set {*Acc#*, *MinimumBalance*}, the value of *MinimumBalance* is associated with an interval, but not necessarily with its sub-intervals. Thus, the value of {*Acc#*, *MinimumBalance*} associated with each sub-interval is not necessarily equal to the value associated with the interval. Therefore, {*Acc#*, *MinimumBalance*} is non-homogeneous. Obviously, a set of attributes X is homogeneous if all the attributes in X are homogeneous, and X is non-homogeneous if at least one of the attributes in X is non-homogeneous.

Similar to the relationship between an attribute and the valid time, the relationship between the set of time-varying attributes and the valid time can be different in different temporal relations. This relationship depends on the meaning of temporal data. Next, characteristics of temporal relations are examined.

3.3 Characteristics of Temporal Relations

One of the interesting features of temporal data is the relationship between data associated with an interval and its sub-intervals. Homogeneous and non-homogeneous attributes are defined on this type of relationship. Properties, processes, and events [3] and upward- and downward-hereditary [73] are also based on this type of relationship.

For example, a temporal relation *Status(Name MaritalStatus T)* is downward-hereditary because the value of (*Name MaritalStatus*) for an interval is also applied to its sub-intervals. It is also upward-hereditary. If the value of (*Name MaritalStatus*) for all sub-intervals of an interval is the same, the value of (*Name MaritalStatus*) for an interval is the same as the value for its sub-intervals. Here, a classification of temporal relations, based on this relationship, is proposed. These characteristics, temporal aggregation and temporal decomposition, are the generalization of upward- and downward-hereditary.

In a temporal aggregation, a temporal data associated with a interval is derived from temporal data associated with its sub-intervals. A temporal-aggregatable relation is defined as follows:

Definition 3.3: Temporal-aggregatable relations

Let R be a set of time-varying attributes, T be the valid time, $r(R\ T)$ be a temporal relation, and f_a be the aggregate function. $r(R\ T)$ is *temporal-aggregatable* if any tuples $\tau_1, \tau_2, \ldots, \tau_n$ in r implies $\tau = f_a(\tau_1, \tau_2, \ldots, \tau_n)$, where $\tau(T) = \tau_1(T) \cup \tau_2(T) \cup \ldots \cup \tau_n(T)$.

Temporal aggregation is the generalization of upward-hereditary. In other words, an upward-hereditary relation is a temporal-aggregatable relation with an identity aggregate function. The following example shows temporal aggregatable relations.

Example 3.1: Consider the relation *Balance(Acc#, MinimumBalance, T)* representing the lowest balance in bank accounts over an interval. The lowest balance over an interval is the minimum of the lowest balance over its sub-intervals. Thus, the relation *Balance* is temporal-aggregatable with the following aggregate function f_a:

$f_a(\tau_1, \tau_2, \ldots, \tau_n) = <\tau_1(Acc\#), min(\tau_1(MinimumBalance), \tau_2(MinimumBalance), \ldots, \tau_n(MinimumBalance)), \tau_1(T) \cup \tau_2(T) \cup \ldots \cup \tau_n(T)>$, where $\tau_1(Acc\#) = \tau_2(Acc\#) = \ldots = \tau_n(Acc\#)$.

The relation *Apartment(Location Size Rent T)* represents the apartment rent. The rent for an apartment is determined by the length of the period of occupancy. Thus, the relation *Apartment* is temporal-aggregatable with the following aggregate function g_a:

$g_a(\tau_1, \tau_2, \ldots, \tau_n) = <\tau_1(Location), \tau_1(Size), \tau_1(Rent)+\tau_2(Rent)+\ldots+\tau_n(Rent), \tau_1(T)\cup\tau_2(T)\cup \ldots\cup\tau_n(T)>$, where $\tau_1(T)$, $\tau_2(T)$, ... and $\tau_n(T)$ are disjoint, $\tau_1(Location) = \tau_2(Location) = \ldots = \tau_n(Location)$ and $\tau_1(Size) = \tau_2(Size) = \ldots = \tau_n(Size)$.

Consider a relation *TempRecord(State HighestTemperature T)* which contains the highest temperature for each state over an interval. The highest temperature for a state over an interval is the maximum of the highest temperature for the state over its sub-intervals. Thus, the relation *TempRecord(State HighestTemperature T)* is temporal-aggregatable with the following aggregate function h_a:

$h_a(\tau_1, \tau_2, \ldots, \tau_n) = <\tau_1(State), \tau_1(HighestTemperature) + \tau_2(HighestTemperature) + \ldots + \tau_n(HighestTemperature), \tau_1(T) \cup \tau_2(T) \cup \ldots \cup \tau_n(T)>$, where $\tau_1(State) = \tau_2(State) = \ldots = \tau_n(State)$.

In a temporal decomposition, a temporal data associated with a sub-interval is derived from a temporal data associated with an interval. A temporal-decomposable relation can be defined as follows:

Definition 3.4: Temporal-decomposable relations

Let R be a set of time-varying attributes, T be the valid time, $r(R\ T)$ be a temporal relation, and f_d be the decomposition function. $r(R\ T)$ is *temporal-decomposable* if a tuple τ in r implies $\tau = f_d(\tau, t)$, where t is in $\tau(T)$.

Similar to temporal aggregation, temporal decomposition is the generalization of downward-hereditary. A downward-hereditary relation is a temporal-decomposable relation with an identity decomposition function. The following example shows temporal-decomposable relations.

Example 3.2: Consider the relation *Apartment(Location Size Rent T)* in Example 3.1. The rent for an apartment is determined by the length of the period of occupancy. Thus, the relation *Apartment* is temporal-decomposable with the following decomposition function f_d:

$f_d(\tau, t) = <\tau(Location), \tau(Size), \tau(Rent)\times\frac{|t|}{|\tau(T)|}, t>$, where $|i|$ is the length of the interval I and t is a sub-interval of $\tau(T)$.

Another example of temporal decomposable relations is the relation *Status (Name MaritalStatus T)* representing employee's marital status. An employee's marital status during a sub-interval is the same as the marital status during the interval. Thus, the relation *Status* is temporal-decomposable with the identity decomposition function g_d:

$g_d(\tau, t) = <\tau(Name), \tau(MaritalStatus), t>$, where t is a sub-interval of $\tau(T)$.

Temporal aggregation and temporal decomposition are useful in temporal databases because, given the aggregate function and the decomposition function, more redundancy can be avoided. Temporal data associated with an interval can be derived from temporal data associated with its sub-intervals, or vice versa. Aggregate and decomposition functions can be used to infer temporal data. However, aggregation and decomposition require computation. Thus, both the storage cost and the computation cost must be considered. However, this is not the principle issue here and will not be further examined.

Based on the concept of temporal-decomposition, two types of temporal relations are defined in the next section. These two types of relations are an important basis for temporal database design.

3.4 Property and Representative Relations

The focus in this section is temporal-decomposable relations whose decomposition functions are identity functions. If a temporal relation has an identity decomposition function, the values of time-varying attributes are associated with all sub-intervals of its valid time. In other words, the valid time can be decomposed and the same values of time-varying attributes are applied to the sub-intervals. As mentioned earlier, it is assumed in many temporal databases that all temporal data have this characteristic. It is called a property relation.

Definition 3.5: Property relation

Let R be a set of time-varying attributes and T be the valid time. A temporal relation $r(R\ T)$ is a *property relation* if, for any tuple τ in r, $\tau(R)$ is associated with any sub-interval of $\tau(T)$.

The following example shows a property relation.

Example 3.3: Consider a relation *Status*(*Name MaritalStatus T*). A tuple <*John*, *married*, [*1994, 1997*]> in the relation *Status* implies <*John*, *married*, [*1994, 1995*]> and <*John*, *married*, [*1996, 1997*]>. Thus, the relation *Status* is a property relation.

On the other hand, for some temporal relation, there is no decomposition function or its decomposition function is not an identity function. In this type of temporal relation, the values of time-varying attributes are associated with the whole interval, but not its sub-intervals. In other words, the valid time in this type of relation cannot be decomposed. This type of relation is called a representative relation.

Definition 3.6: Representative relation

Let R be a set of time-varying attributes and T be the valid time. A temporal relation $r(R\ T)$ is a *representative relation* if there is a tuple τ in r such that $\tau(R)$ is not associated with some sub-interval of $\tau(T)$.

The following example shows a representative relation.

Example 3.4: Consider the relation *TempRecord(State HighestTemperature T)* in Example 3.2. In the relation *TempRecord*, a tuple *<MI, 99°F, [1/1/96, 12/31/96]>* does not imply *<MI, 99°F, [1/1/96, 9/30/96]>* or *<MI, 99°F, [10/1/96, 12/31/96]>*. Thus, the relation *TempRecord* is a representative relation.

The difference between a property relation and a representative relation is the decomposability of the valid time. In a property relation, the valid time can be decomposed, and the same value of the time-varying attributes is associated with the sub-intervals. On the other hand, in a representative relation, the valid time cannot be decomposed because the same value of the time-varying attributes is not associated with the sub-intervals.

The decomposability of the valid time in a temporal relation is an important basis for the interpretation of the semantics of queries, which will be discussed in Chapter 4. This characteristic is also present in spatial domain, and will be discussed next.

3.5 Property and Representative Relations in Spatial Databases

The similarity between temporal and spatial data has been indicated in many works [54, 55, 64]. There are many research on the extension of the relational data model for spatial databases [36, 44]. Characteristics of temporal data discussed earlier are also present in spatial data. The following example shows a property relation and a representative relation in spatial databases.

Example 3.5: The spatial relation *Soil*(*Type Area*) that contains the area and the soil type in the area is a property relation because the type of soil in any sub-area is the same as the type of soil in the area. On the other hand, the spatial relation *Population*(*Animal Number Area*) that contains the area, the type of animal, and its population in the area is a representative relation because the population for a sub-area is not the same as the population for the whole area.

In spatio-temporal databases, the characteristics of a spatio-temporal relation with respect to the spatial domain and the temporal domain are independent. A spatio-temporal relation can be a property relation with respect to one domain and a representative relation with respect to another. For example, consider a spatio-temporal relation *Vegetation*(*Plant Area T*) which contains types of the majority of the vegetation in an area during a time interval. The relation *Vegetation* is a representative relation with respect to the spatial domain because, for any interval, the majority of the vegetation in a sub-area is not necessarily the same as that in the whole area. On the other hand, the relation *Vegetation* is a property relation with respect to the temporal domain because, for a specific area, the majority of the vegetation during an interval is also the majority of the vegetation during its sub-interval.

3.6 Conclusions

In this chapter, characteristics of temporal data are studied. Time-varying attributes are classified into homogeneous and non-homogeneous attributes. Temporal relations are characterized by their decomposability and aggregatability, which are the generalization of downward-hereditary and upward-hereditary. A temporal data associated with a sub-interval can be derived by a decomposition function, and a temporal data associated with the union of intervals can be derived by a aggregation function. This characterization of temporal data can be used for inferring data in temporal databases.

Based on the decomposability of the valid time, temporal relations are classified into property and representative relations. Research on temporal databases assume temporal data is associated with every point in an interval. As a result, representative relations cannot be represented correctly in these temporal databases. The difference between a property relation and a representative relation is an important basis of this work. Finally, the characteristics of temporal data are also present in spatial data. Thus, the characterization can also be applied to spatial data.

Chapter 4

EXTENSION OF RELATIONAL ALGEBRA FOR PROPERTY AND REPRESENTATIVE RELATIONS

4.1 Introduction

Capturing relationships between data is a major task in database design. Representing relationships between data through data is fundamental to database development. However, certain types of relationships are not normally stored in the databases because they are not clearly identified in databases. Often these relationships are encoded into the query processing programs, thus resulting in undue complexity of the application development. On the other hand, capturing these relationships provides an opportunity for explicit representation and development of abstract operations based on these relationships.

The focus of this section is to capture two types of relationships between time-varying data and the valid time in temporal relations, i.e. property and representative relations, in relational algebra. As mentioned in Section 3.3, values of time-varying attributes in a property relation are associated with every point in the valid time (i.e., the valid time in a property relation is decomposable) while values of time-varying attributes in a representative relation are associated with the whole interval (i.e., the valid time in a representative relation is not decomposable.) These characteristics can be captured in temporal databases and incorporated in queries. These characteristics can be utilized in two aspects of queries; temporal predicates, such as *before* and *after*, and relational operators. Interpretations of temporal predicates, based on characteristics of property and representative relations, are examined in Section 4.2. In Section 4.3, the extension of relational operators based on these characteristics is studied. Since property and representative relations are also present in spatial databases, relational algebra can be extended, in a similar manner, to incorporate the characteristics of the relations in spatial databases. In Section 4.4, the extension of relational algebra for spatial databases is discussed.

4.2 Interpretations of Temporal Predicates for Valid Time in Property and Representative Relations

Allen [2] presents thirteen temporal predicates that represent the relationships between two intervals, e.g. *BEFORE, DURING, OVERLAP*, etc. In his work, characteristics of temporal data associated with the intervals are considered. The meaning of each temporal predicate is independent of the characteristics of the temporal data associated with the intervals. However, in temporal databases, these temporal predicates are applied to the valid time in temporal relations. Therefore, the characteristics of the temporal relations need to be considered. For example, consider the interpretations of *before* in the following two queries:

Q_1: Find all employees who paid more than \$1000 income tax before working in project A.

Q_2: Find all employees who worked in project B before working in project A.

The interpretation applied to *before* in Q_1 applies to the whole interval when an employee paid more than \$1,000 income tax before the interval when the same employee worked in project A. However, this interpretation is not applicable to *before* in Q_2. If an employee works in project B during t_3, he/she works in project B during sub-intervals of t_3. Therefore, *before* in Q_2 means "there is a sub-interval of t_3 which is before the whole interval that the employee works in project A". The first interpretation is named *before*$_\forall$, and the second is named *before*$_\exists$. These two interpretations can be defined in terms of Allen's temporal predicates, which is denoted by italic capital letters, as follows:

Let A and B be sets of time-varying attributes, T_A and T_B be the valid time, $R_A(A\ T_A)$ and $R_B(B\ T_B)$ be two temporal relations, and $t_1 \subseteq_i t_2$ denote t_1 is a sub-interval of t_2.

$$\begin{aligned} \mathit{before}_\forall(T_A, T_B) &= \forall\, t_A\ t_A \subseteq_i T_A\ \mathit{BEFORE}(t_A, T_B) \\ &= \mathit{BEFORE}(T_A, T_B). \\ \mathit{before}_\exists(T_A, T_B) &= \exists\, t_A\ t_A \subseteq_i T_A\ \mathit{BEFORE}(t_A, T_B) \\ &= \mathit{BEFORE}(T_A,T_B) \vee \mathit{MEET}(T_A,T_B) \vee \mathit{OVERLAP}(T_A,T_B) \vee \\ &\quad \mathit{FINISHED\_BY}(T_A,T_B) \vee \mathit{CONTAIN}(T_A,T_B). \end{aligned}$$

Types of temporal relations in queries determine which interpretation of a temporal relationship is applicable. The interpretation *before*$_\exists$ is not applicable in Q_1 because the income tax relation is a representative relation and its valid time is not decomposable. Thus, *before*$_\forall$ is used in Q_1. On the other hand, in Q_2, the relation which contains the project which each employee is in is a property relation and the valid time in this relation is decomposable. Therefore, *before*$_\exists$ is applicable in Q_2. If *before*$_\forall$ is used in Q_2, not all employees who worked in project B before working in project A are selected. For example, given Mary worked in project A and B during 1993-1997 and 1990-1995 respectively, Mary is not selected in Q_2 if *before*$_\forall$ is used. However, "*Mary worked in project B during 1990-1995*" implies that she worked in project B during 1990-1992, which is before 1993-1997, and should be selected in Q_2. Therefore, *before*$_\forall$ is not applicable in Q_2.

The interpretations of predicates *after*, *during*, *contain*, and *equal* can be defined similarly, as shown in Table 4.1. The predicates *meet, start, finish, overlap,* and their inverses compare the beginning and/or the ending points of the intervals, not the intervals themselves. Thus, they do not depend on types of temporal relations, and there is only one interpretation for each of them, which is the same as the relationship between intervals defined by Allen [2].

An argument in a temporal predicate can also be a time-interval constant. A time-interval constant behaves like the valid time of a representative relation because it refers to the whole interval, not its sub-intervals. Thus, the interpretations of temporal predicates with time-interval constants are the same as the interpretations of the temporal predicates with valid time of representative relations.

Table 4.1: Temporal predicates for valid time in temporal relations.

Let A and B be sets of time-varying attributes, T_A and T_B be valid time and $R_A(A\ T_A)$ and $R_B(B\ T_B)$ be temporal relations. The column ***Predicate*** contains names of temporal relationship between T_A and T_B. The columns $\boldsymbol{R_A}$ and $\boldsymbol{R_B}$ are types of R_A and R_B, where P and R denote property and representative relations respectively. The column ***Interpretation*** contains the interpretation of the corresponding predicate and the column ***Definition*** contains the definitions of interpretation of the predicates.

Predicate	$\boldsymbol{R_A}$	$\boldsymbol{R_B}$	***Interpretation***	***Definition***
before(T_A,T_B)	R	P/R	*before*$_\forall(T_A,T_B)$	*BEFORE*(T_A, T_B)
before(T_A,T_B)	P	P/R	*before*$_\exists(T_A,T_B)$	*BEFORE*(T_A, T_B) ∨ *MEET*(T_A, T_B) ∨ *OVERLAP*(T_A, T_B) ∨ *CONTAIN*(T_A, T_B)
after(T_A,T_B)	R	P/R	*after*$_\forall(T_A,T_B)$	*AFTER*(T_A, T_B)
after(T_A,T_B)	P	P/R	*after*$_\exists(T_A,T_B)$	*AFTER*(T_A, T_B) ∨ *MET_BY*(T_A, T_B) ∨ *OVERLAPPED_BY*(T_A, T_B) ∨ *DURING*(T_A, T_B)
during(T_A,T_B)	R	P/R	*during*$_\forall(T_A,T_B)$	*DURING*(T_A, T_B)
during(T_A,T_B)	P	P/R	*during*$_\exists(T_A,T_B)$	*OVERLAP*(T_A, T_B) ∨ *OVERLAPPED_BY*(T_A, T_B) ∨ *CONTAIN*(T_A, T_B) ∨ *DURING*(T_A, T_B) ∨ *START*(T_A, T_B) ∨ *STARTED_BY*(T_A, T_B) ∨ *FINISH*(T_A, T_B) ∨ *FINISHED_BY*(T_A, T_B) ∨ *EQUAL*(T_A, T_B)
contain(T_A,T_B)	P/R	R	*contain*$_\forall(T_A,T_B)$	*CONTAIN*(T_A, T_B)
contain(T_A,T_B)	P/R	P	*contain*$_\exists(T_A,T_B)$	*OVERLAP*(T_A, T_B) ∨ *OVERLAPPED_BY*(T_A, T_B) ∨ *CONTAIN*(T_A, T_B) ∨ *DURING*(T_A, T_B) ∨ *START*(T_A, T_B) ∨ *STARTED_BY*(T_A, T_B) ∨ *FINISH*(T_A, T_B) ∨ *FINISHED_BY*(T_A, T_B) ∨ *EQUAL*(T_A, T_B)
equal(T_A, T_B)	R	R	*equal*$_{\forall\forall}(T_A, T_B)$	*EQUAL*(T_A, T_B)
equal(T_A, T_B)	R	P	*equal*$_{\forall\exists}(T_A, T_B)$	*DURING*(T_A, T_B) ∨ *START*(T_A, T_B) ∨ *FINISH*(T_A, T_B) ∨ *EQUAL*(T_A, T_B)
equal(T_A, T_B)	P	R	*equal*$_{\exists\forall}(T_A, T_B)$	*CONTAIN*(T_A, T_B) ∨ *STARTED_BY*(T_A, T_B) ∨ *FINISHED_BY*(T_A, T_B) ∨ *EQUAL*(T_A, T_B)
equal(T_A, T_B)	P	P	*equal*$_{\exists\exists}(T_A, T_B)$	*OVERLAP*(T_A, T_B) ∨ *OVERLAPPED_BY*(T_A, T_B) ∨ *CONTAIN*(T_A, T_B) ∨ *DURING*(T_A, T_B) ∨ *START*(T_A, T_B) ∨ *STARTED_BY*(T_A, T_B) ∨ *FINISH*(T_A, T_B) ∨ *FINISHED_BY*(T_A, T_B) ∨ *EQUAL*(T_A, T_B)

4.3 Relational Operators Based on Types of Temporal Relations

In this section, the extension of relational operators, based on types of temporal relations, is discussed. The operators *select*, *join*, *union*, and *difference* are extended so that the manipulation of the valid time in the temporal relations involved in the operators is implied from types of the relations. On the other hand, the *project* operator does not depend on types of temporal relations because it does not involve the valid time. Next, the extension of the relational operators is presented.

4.3.1 SELECT

The *select* operator selects tuples that satisfy a predicate on the values of time-varying attributes or the valid time. The *select* operator on time-varying attributes does not depend on the type of the temporal relation. On the contrary, the *select* operator on the valid time depends on the type of the temporal relation. In the *select* operator on the valid time, temporal predicates on the valid time are used to select tuples from a relation. The interpretations of the temporal predicates are determined by types of temporal relations, as shown in Table 4.1. Furthermore, the valid time in the selected tuple is not necessarily the same as that in the original tuple. If the valid time in a relation is decomposable and a sub-interval of the valid time in a tuple satisfies the temporal predicate, the tuple can be selected. However, the valid time in the tuple needs to be modified so that the valid time of the selected tuple is the sub-interval that satisfies the temporal predicate. The valid time in a relation created by the *select* operator on different temporal predicates is shown in Table 4.2. This calculation of the valid time is also applied for the *join* operator because join is based on *select*.

Table 4.2: The valid time in relations created by the operators *select* or *join*.

Let A and B be sets of time-varying attributes, T_A and T_B be valid time and $R_A(A\ T_A)$ and $R_B(B\ T_B)$ be temporal relations. The column ***Predicate*** contains names of temporal predicates between T_A and T_B used in the operators *select* or *join*. The columns $\boldsymbol{R_A}$ and $\boldsymbol{R_B}$ are types of R_A and R_B, where P and R denote property and representative relations respectively. The column ***Time*** contains the valid time in the relation created by the operator.

Predicate	***A***	***B***	***Time***
$before(T_A, T_B)$	R	P/R	T_A
$before(T_A, T_B)$	P	P/R	$[begin(T_A), min(end(T_A), begin(T_B))]$
$after(T_A, T_B)$	R	P/R	T_A
$after(T_A, T_B)$	P	P/R	$[max(end(T_B), begin(T_A)), end(T_A)]$
$during(T_A, T_B)$	R	P/R	T_A
$during(T_A, T_B)$	P	P/R	$T_A \cap T_B$
$contain(T_A, T_B)$	P/R	R	T_A
$contain(T_A, T_B)$	P/R	P	$T_A \cap T_B$
$equal(T_A, T_B)$	R	R	T_A
$equal(T_A, T_B)$	R	P	T_A
$equal(T_A, T_B)$	P	R	T_B
$equal(T_A, T_B)$	P	P	$T_A \cap T_B$

The following example shows the *select* operator on the valid time.

Example 4.1: Consider the following queries:

> ***Q_3: Find the positions of all employees before 1990.***
>
> ***Q_4: Find the commission for employees before 1990.***

Let *Employee*(*Name Position T*) be a property relation, *Commission*(*Name Amount T*) be a representative relation, and T be the valid time. Q_3 and Q_4 can be expressed as shown in Figure 4.1 (c) and (d). According to Table 4.1, the interpretation $before_{\exists}$ is applied in Q_3 and $before_{\forall}$ is applied in Q_4. According to Table 4.2, the valid time of the relation created by Q_3 is [*begin*(*Employee.T*), *min*(*end*(*Employee.T*), 1990)]. The tuple <*Susan*, *manager*, [*1985*, *1989*]> is the result of Q_3 is created from the tuple <*Susan*, *manager*, [*1985*, *1993*]> in the relation *Position* because the sub-interval [*1985*, *1989*] of [*1985*, *1993*] satisfies the predicate *before*(*Employee.T*, *1990*). The valid time of the relation created by Q_4 is *Commission.T* because *Commission.C* is not decomposable. Figure 4.1 shows the relations *Employee* and *Commission* and the results of the queries Q_3 and Q_4.

Name	*Position*	*T*
John	*president*	*[1980,1988]*
Susan	*manager*	*[1985,1993]*
Mary	*sales*	*[1992,1996]*

(a) The relation *Employee*

Name	*Amount*	*T*
John	*$40,000*	*[1980,1988]*
Susan	*$30,000*	*[1985,1993]*
Mary	*$10,000*	*[1992,1996]*

(b) The relation *Commission*

Name	*Position*	*T*
John	*president*	*[1980,1988]*
Susan	*manager*	*[1985,1989]*

$\sigma_{before\ (TE,\ 1990)}Employee$

(c) The result of the query Q_3.

Name	*Amount*	*T*
John	*$40,000*	*[1980,1988]*

$\sigma_{before\ (TS,\ 1990)}Commission$

(d) The result of the query Q_4.

Figure 4.1: The results of the queries Q_3 and Q_4.

4.3.2 JOIN

The *join* operator is the combination of the *select* operator and the cross product. Therefore, the extension of *join* is based on the extension of *select*. Similar to the *select* operator, the *join* operator on time-varying attributes does not depend on types of temporal relations. The *join* operator on the valid time, however, depends on types of temporal relations. The interpretation of the temporal predicate in *join* is determined by the types of relations, as shown in Table 4.1. The valid time in the relation created by *join* is calculated according to Table 4.2. The following example shows the *join* operator on different types of relations.

Example 4.2: Consider the following queries:

> *Q_5: Find position of all employees after John was the president.*
>
> *Q_6: Find commission of all employees after John was the president.*

The relational expressions of Q_5 and Q_6 are shown in Figure 4.2 (a) and (b). The interpretation $after_\exists$ is applied in Q_5 and the interpretation $after_\forall$ in Q_6. The valid time in the relation created from Q_5 is [*max*(*end*(*Employee2.T*), *begin*(*Employee1.T*)), *end*(*Employee1.T*)], and the valid time in the relation created from Q_6 is *Comission.T*. Figure 4.2 shows the results of the queries Q_5 and Q_6.

Name	*Position*	*T*
Susan	*manager*	*[1989, 1993]*
Mary	*sales*	*[1992, 1996]*

$\pi_{(Employee1.Name\ Employee1.Position)}\ Employee \bowtie_{after(Employee1.T,\ Employee2.T)}$
$\sigma_{(Employee2.Position='President' \wedge Employee2.Name='John')}\ Employee.$

(a) The result of the query Q_5.

Name	*Amount*	*T*
Mary	*$10,000*	*[1992, 1996]*

$\Pi_{(Comission.Name\ Amount)}\ (Commission \bowtie_{after(Commission.T,\ Employee.T)}$
$\sigma_{(Position='President' \wedge Employee.Name='John')}\ Employee).$

(b) The result of the query Q_6.

Figure 4.2: The results of the queries Q_5 and Q_6.

4.3.3 Union

The *union* operator for temporal relations is similar to the union operator for non-temporal relations. The difference between the union operator on non-temporal relations and that on temporal relations is in the definition of "*union-compatible*". Non-temporal relations are union-compatible if their schemes are the same. For non-temporal relations, types of temporal relations must also be considered. Temporal relations are union-compatible when their schemes are the same and they are of the same type. When relations are of different types, the temporal data in the relations have different semantics and it does not make sense to combine them. Thus, the *union* operator cannot be performed on different types of relations.

4.3.4 Difference

Similar to the *union* operator, the *difference* operator cannot be performed on different types of relations. If both relations are representative relations, the set difference can be performed directly on two relations. However, if both relationships are property relations, the difference between the valid time must be calculated for tuples with the same values of time-varying attributes and overlapping valid time. The following example shows the difference of two property relations.

Example 4.3: Consider the following query:

> *Q_7: Find employees who are not Michigan residents.*

The relation *Resident* is shown in Figure 4.3 (a). The tuple <*Susan*, [*1985, 1993*]>

projected from the relation *Employee* is not equal to <*Susan*, [*1980, 1986*]> projected from the relation *Resident*. However, in these two tuples, the values of time-varying attribute are equal and the values of the valid time overlap. During the difference of the two intervals, i.e. [*1987, 1990*], Susan is an employee who is not a Michigan resident. Thus, <*Susan*, [*1987, 1990*]> is in the result of Q_7, as shown in Figure 4.3 (b).

Name	***Residence***	***T***
John	*Michigan*	[*1970, 1990*]
Susan	*Michigan*	[*1980, 1986*]
Mary	*Michigan*	[*1981, 1989*]

(a) The relation *Resident*

Name	***T***
Susan	[*1987, 1990*]
Mary	[*1992, 1996*]

$\Pi_{(Name\ T)}\, Employee - \Pi_{(Name\ T)}\, \sigma_{(Residence=\text{"}MI\text{"})}\, Resident$

(b) The result of the query Q_7.

Figure 4.3: The difference between *Employee* and *Resident*.

Types of temporal relations in relational operators determine the manipulation of valid time. Furthermore, they are used to determine the interpretations of temporal predicates, as shown in Section 4.2. Therefore, types of temporal relations must also be stored in the database.

The characteristics of property and representative relations are also present in spatial databases. Characteristics of spatial relations can also be incorporated in the relational operators for spatial databases. Similar extension of predicates and relational operators can be applied in the spatial domain. In the next section, the extension of the relational algebra for spatial relational databases are discussed.

4.4 Extension of Relational Algebra for Spatial Databases

As shown earlier in Section 3.5, the characteristics of property and representative relations are also present in the spatial domain. In this section, the extensions of relational algebra for spatial databases are examined. First, the extension of spatial predicates are studied. The relational operators for spatial databases are similar to those defined in Section 4.3.

There are many works on the spatial predicates [35, 64]. In this section, the predicates *disjoint*, *meet*, *covers* and, others [64] are adopted. Similar to the temporal predicates, the interpretations of these spatial predicates depend on types of spatial relations. For example, the predicate *covered_by* can be interpreted as $covered\_by_{\exists}$ (AP,AQ), meaning some part of the area *AP* is covered by *AQ*, or $covered\_by_{\forall}(AP,AQ)$, meaning the whole area *AP* is covered by *AQ*. The interpretation $covered\_by_{\exists}$ (AP,AQ) is applicable if *AP* is the spatial attribute in a property relation. The interpretation $covered\_by_{\forall}(AP,AQ)$ is applicable if *AP* is the spatial attribute in a representative relation. The following example shows the queries that use the predicate *covered_by* on a property relation and a representative relation.

Example 4.4: Consider the predicate *covered_by* (i.e. inside) in the following queries:

Q_8: *Find area inside a national park where soil type is sandy.*

Q_9: Find area inside a national park where deer population <100 per sq. mile.

As shown in Example 3.5, the relation *Soil* is a property relation and the relation *Population* is a representative relation. Therefore, the interpretation *covered_by*$_\forall$ is applicable for the deer population, and the interpretation *covered_by*$_\exists$ is applicable for the soil data.

The extension of relational operators for temporal databases is mostly applicable for spatial databases. However, the calculation of the spatial attribute in the operators *select* and *join*, which depends on spatial predicates, must be defined. For example, the value of the spatial attribute in the relation created by using the predicate *covered_by*$_\forall$(*AP*, *AQ*) is *AP*, and that using the predicate *covered_by*$_\exists$(*AP*, *AQ*) is $AP \cap AQ$, where *AP* and *AQ* are spatial attributes. The calculation of the spatial attribute can be derived by using the same concept as the calculation of the valid time, and will not be presented here.

4.5 Conclusions

In this chapter, the relational algebra is extended to incorporate characteristics of property and representative relations. The valid time of a relation created from a relational operator is handled according to the types of the input relations. Furthermore, the interpretations of temporal predicates depend on the types of the temporal relations whose valid times are in the temporal predicates. Since the characteristics of property and representative relations are also present in spatial databases, the interpretations of spatial predicates also depend on types of spatial relations. Furthermore, the extension of the relational operators for temporal databases is also applicable to spatial databases.

In conclusion, it is shown that characteristics of property and representative relations can be incorporated in relational algebra. The extension of relational algebra discussed in this chapter can be used as a basis for other temporal databases, such as a temporal deductive database presented in Chapter 6.

When the characteristics of property and representative relations are considered, inconsistency in temporal databases can be identified. It is crucial to avoid inconsistency in temporal databases. In the next chapter, a design principle for temporal relational databases, which avoids the inconsistency, is studied.

Chapter 5

DESIGN PRINCIPLES FOR TEMPORAL DATABASES WITH HOMOGENEOUS AND NON-HOMOGENEOUS DATA

5.1 Introduction

Normal forms discussed in Section 2.2 are not applicable when non-homogeneous data are present in temporal databases. In temporal databases with homogeneous and non-homogeneous data, inconsistency can occur. Consider the relation *TaxRecord*, in Figure 5.1, which contains homogeneous attributes, *Name* and *MaritalStatus*, and a non-homogeneous attribute, *IncomeTax*. Inconsistency occurs if *John's income tax for* [*1/1/96, 12/31/96*] is inserted into the relation *TaxRecord*. This is because the valid time, [*1/1/96, 12/31/96*], which is associated with a non-homogeneous attribute *IncomeTax*, are not decomposable. We cannot assign either "*single*" or "*married*" as the value of *MaritalStatus* during [1/1/96, 12/31/96]. If we assign "*single*" as the value of *MaritalStatus* during [1/1/96, 12/31/96], it is contradicted to the second tuple. If we assign "*married*" as the value of *MaritalStatus* during [1/1/96, 12/31/96], it is contradicted to the first tuple. If the value of *MaritalStatus* is left as null, it also causes inconsistency. This null value may mean that John's marital status during 1996 is unknown, but this contradicts to the first and the second tuples. We call this type of inconsistency *P-inconsistency.*

Name	*MaritalStatus*	*IncomeTax*	*T*
John	*single*	$\perp$	[*1/1/96, 8/9/96*]
John	*married*	$\perp$	[*8/9/96, 12/31/96*]

Figure 5.1: P-inconsistency in the relation *TaxRecord.*

The goal in this section is to provide a design principle that avoids P-inconsistency. This principle is based on both the types of attributes and data dependencies in temporal databases. In Section 5.2, we define two types of data dependencies. Inference rules for the dependencies are presented in Section 5.3. We prove that the inference rules are both sound and complete. In Section 5.4, a normal form that avoids P-inconsistency is presented. In Section 5.5, equivalent sets of attributes and minimal covers, which are the basis for the normalization, are defined. For two given sets of attributes, determining if they are equivalent can be done in polynomial-time [9]. For our problem, however, we need to find equivalent sets of attributes for a given set of attributes. This problem has not been addressed previously in other works. We prove that this problem is NP-complete. In Section 5.6, we show that finding an optimal decomposition is NP-complete, and present a polynomial-time heuristic for the decomposition.

5.2 Functional Dependencies and P-Dependencies in Temporal Relations

In this section, we define a type of data dependency in temporal relations that is a basis of the design principle proposed here. First, we define basic temporal normal form (BTNF). Similar to 1NF for non-temporal relations and 1TNF for temporal relations, BTNF is the basic requirement for temporal relations. However, 1TNF is not applicable when non-homogeneous attributes are present. BTNF is defined as follows:

Definition 5.1: Basic Temporal Normal Form

A temporal relation scheme *R* is in *basic temporal normal form* (*BTNF*) if the domain of each time-varying attribute in *R* is a set of atomic values and the domain of the valid time is a set of time intervals.

The relation *TaxRecord* in Figure 5.1 is in BTNF because the values of the attributes *Name*, *MaritalStatus*, and *IncomeTax*, are atomic and the values of the valid time are intervals. Throughout this chapter, we assume that all temporal relations are in BTNF. Note also that when we use the terms "relation" and "attribute", they mean "temporal relation" and "time-varying attribute" unless stated otherwise.

Functional dependencies can be used to describe data dependencies in temporal relations when the valid time is considered atomic in the dependencies. For example, the valid time *T* is considered atomic in the functional dependency *State T* $\rightarrow$ *HighestTemperature*. If the valid time of two tuples are exactly the same, there is only one value of the highest temperature for each state and each time interval. Nothing can be said about the highest temperature for sub-intervals of the valid time. However, functional dependencies cannot capture some dependencies in temporal data. Consider the dependency of the attribute *MaritalStatus* on the attribute *Name*, where *MaritalStatus* and *Name* represent the marital status and a person, respectively. A person has only one marital status at any time. Therefore, if the values of *Name* in two tuples are the same and the values of the valid time overlap, the values of *MaritalStatus* during the overlapping time must be the same. Thus, the values of *MaritalStatus* in two tuples are also the same. This dependency cannot be captured by a functional dependency.

A temporal functional dependency [48] captures this type of data dependency in temporal data. A set of attributes *Y* is temporally functional dependent on *X* if, at any time, there is one value of *Y* for each value of *X*. The dependency of *MaritalStatus* on *Name* can be captured by the temporal functional dependency. However, when non-homogeneous attributes are involved, temporal functional dependencies are not applicable because the values of non-homogeneous attributes for sub-intervals are not necessarily the same as the values for the whole interval. Consider a non-homogeneous attribute *MinimumBalance* and a homogeneous attribute *InterestRate* in the relation *Rate* in Figure 5.2. *MinimumBalance* represents the level of the lowest balance in a bank account during an interval, and *InterestRate* represents the interest rate for an interval. We assume if the balance in an account is kept within a certain level for a month interval, an interest rate is applied to the account for the whole month. That is, the level of the lowest balance over a month interval determines the interest rate for that month. In two tuples, if the values of *MinimumBalance* are the same and the values of the valid time overlap, the values of *InterestRate* are equal, as shown in the relation *Rate*. The dependency of *InterestRate* on *MinimumBalance* is similar to, but not exactly the same as, temporal functional dependency. This dependency cannot be captured by temporal functional dependency because the values of *MinimumBalance* for sub-intervals are not necessarily equal to the value for the whole interval. However, the value of *InterestRate* for any sub-interval is determined by the value of *MinimumBalance* for the whole interval. For example, from the first tuple in relation *Rate*, the interest rate for an account with $500 lowest balance during [1/15/96, 1/31/96] is 4%. Even though the lowest balance for a sub-interval, say [1/16/96, 1/31/96], is $1,000, the interest rate for the sub-interval is still 4%, not 5%.

MinimumBalance	*InterestRate*	*T*
$500	*4%*	*[1/1/96, 1/31/96]*
$500	*4%*	*[1/16/96, 2/15/96]*
$1,000	*5%*	*[1/1/96, 1/31/96]*
$500	*5%*	*[2/16/96, 3/15/96]*

Figure 5.2: The relation *Rate*.

We modify the definition of temporal functional dependencies to capture this type of dependency, and call it *property dependency*. The semantics of property dependency is the same as a temporal functional dependency when all time-varying attributes in the dependency are homogeneous. However, they are different when non-homogeneous attributes are present. In Definition 5.2, the property dependency of Y on X and T holds if any two tuples with the same value of X and overlapping valid time T has the same value of Y.

Definition 5.2: Property dependency or P-dependency

Let $r(R)$ be a temporal relation, and X and Y be sets of attributes in R. Let τ_i, for $i = 1$ or 2, be a tuple in r, $\tau_i(X)$ denote the value of X in τ_i, and $\tau_i(T)$ denote the value of the valid time T in τ_i. $r(R)$ satisfies a *property dependency* or *P-dependency* $X\ T \Rightarrow Y$ or X and T *P-*

determines *Y* in *r*(*R*) if, for any two tuples τ_1 and τ_2 in *r*, $\tau_1(Y)=\tau_2(Y)$ if $\tau_1(X)=\tau_2(X)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$.

In temporal database design, it is necessary to consider functional dependencies as well as P-dependencies. In the next section, inference rules involving both functional dependencies and P-dependencies are presented.

5.3 Inference Rules for Functional Dependencies and P-dependencies

In this section, we first examine the relationship between functional dependencies and P-dependencies. Then, inference rules which involve only P-dependencies are presented. These inference rules directly correspond to the inference rules for functional dependencies. We show that inference rules presented here are sound, and they together with inference rules for functional dependencies are complete.

5.3.1 The Relationship between Functional Dependencies and P-dependencies

First, we examine inference rules that give the relationships between functional dependencies and P-dependencies. Let *r*(*R*) be a relation, *X* and *Y* be sets of attributes in *R*, and $X\ T \rightarrow Y$ denote the functional dependency of *Y* on *X* and *T*. The following inference rules hold for any relation.

I1: $X\,T \Rightarrow Y$ implies $X\,T \rightarrow Y$.

I2: $X\,T \rightarrow Y$ implies $X\,T \Rightarrow Y$ if *X* and *Y* are homogeneous.

From I1, we see that a P-dependency implies a functional dependency. From I2, a functional dependency implies a P-dependency only when all time-varying attributes in the dependency are homogeneous. Next, we prove I1 and I2 are sound.

Proposition 5.1: I1 and I2 are sound.

Proof:

I1 follows from the definitions of functional dependency and P-dependency.

For I2: Let X and Y be sets of homogeneous attributes, $r(R)$ be a temporal relation that satisfies $X\ T \rightarrow Y$, and τ_1 and τ_2 be any two tuples in r such that $\tau_1(X)=\tau_2(X)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$. Since X and Y are homogeneous, the values of X and Y for the sub-interval $\tau_1(T)\cap\tau_2(T)$ are the same as the values of X and Y for the whole intervals $\tau_1(T)$ and $\tau_2(T)$. Thus, $\tau_1(X)=\tau_2(X)$, $\tau_1(Y)=\tau_2(Y)$, and $X\ T \Rightarrow Y$. □

From I1 and I2, we can conclude that a functional dependency is equivalent to a P-dependency if all time-varying attributes in the dependency are homogeneous. For example, *Name* $T \rightarrow$ *MaritalStatus* and *Name* $T \Rightarrow$ *MaritalStatus* are equivalent because *Name* and *MaritalStatus* are homogeneous.

Lemma 5.1: If X and Y are sets of homogeneous attributes, $X\ T \rightarrow Y$ is equivalent to $X\ T \Rightarrow Y$.

Proof:

Let X and Y be sets of homogeneous attributes. From I1, $X\ T \Rightarrow Y$ implies $X\ T \rightarrow Y$. From

I2, $X\,T \rightarrow Y$ implies $X\,T \Rightarrow Y$. Thus, $X\,T \rightarrow Y$ is equivalent to $X\,T \Rightarrow Y$. □

5.3.2 Inference Rules for P-dependencies

Inference rules for P-dependencies are similar to those for functional dependencies. In Proposition 5.2, we show that reflexivity, augmentation, and transitivity hold for P-dependencies.

Proposition 5.2: Let $r(R)$ be a relation, and X, Y, and Z be sets of attributes in R. For any $r(R)$, the following inference rules hold.

I3 (Reflexivity): $X\,Y\,T \Rightarrow X$.

I4 (Augmentation): $X\,T \Rightarrow Y$ implies $X\,Z\,T \Rightarrow Y\,Z$.

I5 (Transitivity): $X\,T \Rightarrow Y$ and $Y\,T \Rightarrow Z$ imply $X\,T \Rightarrow Z$.

Proof:

For I3: It is obvious that, for any tuples τ_1 and τ_2 in r, if $\tau_1(X\,Y)=\tau_2(X\,Y)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$, then $\tau_1(X)=\tau_2(X)$. Thus, $X\,Y\,T \Rightarrow X$.

For I4: Since $X\,T \Rightarrow Y$, for any tuples τ_1 and τ_2 in r, $\tau_1(Y)=\tau_2(Y)$ if $\tau_1(X)=\tau_2(X)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$. Then, for any tuples τ_1 and τ_2 in r, $\tau_1(Y\,Z)=\tau_2(Y\,Z)$ if $\tau_1(X\,Z)=\tau_2(X\,Z)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$. Thus, $X\,Z\,T \Rightarrow Y\,Z$.

For I5: Since $X\,T \Rightarrow Y$, for any tuples τ_1 and τ_2 in r, $\tau_1(Y)=\tau_2(Y)$ if $\tau_1(X)=\tau_2(X)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$. Since $Y\,T \Rightarrow Z$, $\tau_1(Z)=\tau_2(Z)$ if $\tau_1(Y)=\tau_2(Y)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$. Then, for any tuples τ_1 and τ_2 in r, $\tau_1(Z)=\tau_2(Z)$ if $\tau_1(X)=\tau_2(X)$ and $\tau_1(T)\cap\tau_2(T)\neq\emptyset$. Thus, $X\,T \Rightarrow Z$.□

The following example shows the derivation of P-dependencies which uses reflexivity, augmentation, and transitivity of P-dependencies.

Example 5.1: Given the following P-dependencies:

P1: *Name T* $\Rightarrow$ *Dept Rank*, and

P2: *Rank CompanyProfit T* $\Rightarrow$ *Bonus*.

The P-dependency *Name CompanyProfit T* $\Rightarrow$ *Bonus* can be derived from P1 and P2 by using I3-I5 as shown below.

P3:	*Name CompanyProfit T* $\Rightarrow$ *Dept Rank CompanyProfit*	(From P1 by I4)
P4:	*Dept Rank CompanyProfit T* $\Rightarrow$ *Rank CompanyProfit*	(By I3)
P5:	*Name CompanyProfit T* $\Rightarrow$ *Rank CompanyProfit*	(From P3 and P4 by I5)
P6:	*Name CompanyProfit T* $\Rightarrow$ *Bonus*	(From P2 and P5 by I5)

5.3.3 Complete Set of Inference Rules for Functional Dependencies and P-dependencies

Now, we show that I1-I5, together with inference rules for functional dependencies, are complete for deriving both functional dependencies and P-dependencies in temporal data. We call reflexivity, augmentation, and transitivity for functional dependencies I6, I7 and I8, respectively. To prove the completeness, we need to define the transitive closure of a set of attributes, based on functional dependencies and P-dependencies. Note that when we use the term “dependencies”, we mean functional dependencies and P-dependencies.

Definition 5.3: Closure

Let X and Y be sets of attributes, and G be a set of dependencies. The *closure* of G, denoted by G^+, is the set of dependencies derived from G, using I1-I8. The *functional closure* of X under G, denoted by $X_F{}^+$, = $\{Y \mid X\,T \rightarrow Y$ is in $G^+\}$. The *property closure* of X under G, denoted by $X_P{}^+$, = $\{Y \mid X\,T \Rightarrow Y$ is in $G^+\}$.

Using the definition of closure, equivalent sets of functional dependencies and P-dependencies can be defined similar to equivalent sets of functional dependencies. That is, a set of dependencies G is equivalent to a set of dependencies H if $G^+=H^+$. Also note that, for any set of attributes X, $X^+_P \subseteq X_F{}^+$ because $X\,T \Rightarrow Y$ implies $X\,T \rightarrow Y$. The following example shows functional and property closures of a set of attributes.

Example 5.2: Let *Acc#*, *Name*, *Addr* and *InterestRate* be homogeneous attributes, *MinimumBalance* be a non-homogeneous attribute, and *Bank*(*Acc# Name Addr InterestRate MinimumBalance T*) be a temporal relation. Given the set of dependencies, D1-D4 shown below, on the relation *Bank*, we show the derivation of the property closure and the functional closure of *Acc#*.

D1: *Acc#* $T \Rightarrow$ *Name*

D2: *MinimumBalance* $T \Rightarrow$ *InterestRate*

D3: *Acc#* $T \rightarrow$ *MinimumBalance*

D4: *Name* $T \rightarrow$ *Addr*

The property closure of *Acc#*, $Acc\#_P{}^+$, is {*Acc#, Name, Addr, InterestRate*}, and the derivation is shown below.

D5:	*Acc# T* $\Rightarrow$ *Acc# Name*	(From D1, by I4)
D6:	*Name T* $\Rightarrow$ *Addr*	(From D4, by I2)
D7:	*Acc# Name T* $\Rightarrow$ *Acc# Name Addr*	(From D6, by I4)
D8:	*Acc# T* $\Rightarrow$ *Acc# Name Addr*	(From D5 and D7, by I5)
D9:	*MinimumBalance T* $\rightarrow$ *InterestRate*	(From D2, by I1)
D10:	*Acc# T* $\rightarrow$ *InterestRate*	(From D3 and D9, by I8)
D11:	*Acc# T* $\Rightarrow$ *InterestRate*	(From D10, by I2)
D12:	*Acc# Name Addr T* $\Rightarrow$ *Acc# Name Addr InterestRate*	(From D11, by I4)
D13:	*Acc# T* $\Rightarrow$ *Acc# Name Addr InterestRate*	(From D8 and D12, by I5)

The functional closure of *Acc#*, $Acc\#_F^+$, is {*Acc#, Name, Addr, MinimumBalance, InterestRate*}, and it can be derived similarly.

I1-I8 are complete, i.e., a dependency can be derived from a set of dependencies *G* by using I1-I8 if and only if the dependency is implied by *G*. The proof is similar to the proof of completeness for inference rules for traditional functional dependencies [10]. We show that there exists a temporal relation *r* that satisfies any functional dependency or P-dependency which is in the closure of a set of dependencies and no other functional dependency or P-dependency which is not in the closure.

Theorem 5.1: I1-I8 are complete.

The Proof is shown in Appendix A.

5.4 P-inconsistency and Normal Forms for Temporal Relations

In this section, two normal forms for temporal relations are presented. First, we show how P-inconsistency occurs and present a normal form that avoids P-inconsistency. Second, we extend traditional 3NF to avoid anomalies in temporal relations. Since functional dependencies are present in temporal relations, anomalies addressed in non-temporal normal forms can also occur in temporal relations.

5.4.1 P-INCONSISTENCY AND P-CONSISTENCY NORMAL FORM

In some temporal relations, certain information cannot be inserted. Consider the relation *AccInterest*, in Figure 5.3, under the set of dependencies {*Acc# T* $\rightarrow$ *MinimumBalance*, *MinimumBalance T* $\Rightarrow$ *InterestRate*}. The information, 'the level of the lowest balance for the account number 276282 during [1/16/96, 2/15/96] was \$500', cannot be inserted in the relation *AccInterest* because there are two different interest rates for the \$500 minimum balance during the interval. The interest rate is 4% for [1/16/96, 1/31/96] and 5% for [2/1/96, 2/15/96]. It is not possible to put both values of *InterestRate* in a tuple because it violates BTNF. However, the interval [1/16/96, 2/15/96] cannot be decomposed into intervals [1/16/96, 1/31/96] and [2/1/96, 2/15/96] for each value of *InterestRate* because *MinimumBalance* is a non-homogeneous attribute. We call this problem *P-inconsistency*. In the relation *AccInterest*, P-inconsistency occurs because there are a P-dependency *MinimumBalance T* $\Rightarrow$ *InterestRate* and another dependency *Acc# T* $\rightarrow$ *MinimumBalance*, which contains a non-homogeneous attribute

MinimumBalance.

Acc#	*MinimumBalance*	*InterestRate*	*T*
176282	*$500*	*4%*	*[1/1/96, 1/31/96]*
176282	*$500*	*5%*	*[2/1/96, 2/28/96]*

Figure 5.3: P-inconsistency in the relation *AccInterest.*

P-inconsistency in the relation *AccInterest* can be avoided by decomposing the relation *AccInterest* into the relations *AccBalance* and *BalanceRate*, in Figure 5.4. The decomposition cannot be done based on functional dependencies only. The property dependency and types of attributes are important basis for the decomposition. In these two relations, a P-dependency *MinimumBalance T* $\Rightarrow$ *InterestRate* and another dependency *Acc# T* $\rightarrow$ *MinimumBalance* with a non-homogeneous attribute *MinimumBalance* are not present in the same relation. Thus, P-inconsistency is avoided, and the information 'the level of the lowest balance for the account 276282 during [1/16/96, 2/15/96] was $500' can be inserted, as shown in Figure 5.4 (a), without any problem.

However, if all attributes in the P-dependency are included in another dependency with non-homogeneous attributes, or vice versa, the two dependencies can be allowed in the same relation. For example, suppose {*MinimumBalance T* $\Rightarrow$ *InterestRate*, *InterestRate T* $\rightarrow$ *MinimumBalance*} is the set of dependencies on the relation *BalanceRate*, in Figure 5.4(b). P-inconsistency cannot occur in this relation because any valid data must satisfy both dependency, and thus there is only one value of *InterestRate* for an interval. Therefore, P-inconsistency occurs when a P-dependency and another dependency with non-homogeneous attributes are present in the same relation while one dependency does not contain all attributes in another, or vice versa.

Acc#	*MinimumBalance*	*T*
176282	*$500*	*[1/1/96, 1/31/96]*
176282	*$500*	*[2/1/96, 2/28/96]*
276282	*$500*	*[1/16/96, 2/15/96]*

(a) The relation *AccBalance.*

MinimumBalance	*InterestRate*	*T*
$500	*4%*	*[1/1/96, 1/31/96]*
$500	*5%*	*[2/1/96, 2/28/96]*

(b) The relation *BalanceRate.*

Figure 5.4: The decomposition of the relation *AccInterest* into the relations *AccBalance* and *BalanceRate* to avoid P-inconsistency.

Without losing generality, we assume that each attribute in a relation must be in a dependency. If there are attributes which are not in any dependency, a relation is created for each attribute. Since there is no constraint on the relation, P-inconsistency cannot occur. Now, we only have to deal with attributes in the dependencies. Trivial dependencies, such as *Name T* $\rightarrow$ *Name* or *Name T* $\Rightarrow$ *Name*, can be disregarded because they are always satisfied. Then, a normal form that avoids P-inconsistency can be defined as follows:

Definition 5.4: P-consistency normal form

Let *R* be a relation scheme, *X* and *Y* be sets of attributes in *R*, *A* be a single attribute in *R*, and *G* be a set of dependencies on *R*. *R* is in *P-consistency normal form* (*PCNF*) with respect to G if R is in BTNF, and;

1. There is no non-homogeneous attribute in R, or

2. All dependencies on R are functional dependencies, except P-dependencies $X\ T \Rightarrow Y$ such that X, Y and T are all in every non-trivial dependency on R, or

3. There is a non-trivial P-dependency of the form $X\ T \Rightarrow A$ in G, such that X or A are non-homogeneous, there is no $Y \subset X$ that $Y\ T \Rightarrow A$, and the P-dependency contains all attributes in R.

The following example shows a set of relations in PCNF.

Example 5.3: Consider a set of dependencies on the relation *Bank* in Example 5.2. The relation *Bank* can be decomposed into three PCNF relations; *Customer(Acc# Name Addr T)*, *Balance(Acc# MinimumBalance T)* and *Rate(MinimumBalance InterestRate T)*. According to Item 1 in Definition 5.4, the relation *Customer* under the set of dependencies $\{Acc\#\ T \Rightarrow Name, Name\ T \rightarrow Addr\}$ is in PCNF because there is no non-homogeneous attribute in the relation. According to Item 2, the relation *Balance* under the set of dependencies $\{Acc\#\ T \rightarrow MinimumBalance\}$ is in PCNF because there is no non-trivial P-dependency on the relation. According to Item 3, the relation *Rate* under the set of dependencies $\{MinimumBalance\ T \Rightarrow InterestRate\}$ is in PCNF because *MinimumBalance* is non-homogeneous, and there is a P-dependency *MinimumBalance T* $\Rightarrow$ *InterestRate* which contains all attributes in the relation.

Next, we prove that PCNF avoids P-inconsistency. That is, a non-trivial P-dependency and another non-trivial dependency with non-homogeneous attributes are not present in the same PCNF relation, unless one dependency contains all attributes in another dependency.

Theorem 5.2: PCNF avoids P-inconsistency.

Proof:

We show that PCNF prevents the presence of a P-dependency and another dependency with non-homogeneous attributes in the same relation unless one dependency contains all attributes in another.

First, consider a relation that contains no non-homogeneous attributes, according to Item 1 in Definition 5.4. Then there is no dependency with non-homogeneous attributes on the relation. Thus, P-inconsistency cannot occur.

Second, there are two types of relations according to Item 2 in Definition 5.4. One is a relation on which there is no non-trivial P-dependency. The other is a relation on which there is a non-trivial P-dependency derived from *G*, and all attributes in the P-dependency are contained in every non-trivial dependency on *R*. If there is no non-trivial P-dependency on *R*, it is clear that P-inconsistency cannot occur. If there is a non-trivial P-dependency derived from *G*, and all attributes in the P-dependency are contained in every non-trivial dependency on *R*, then there is no non-trivial P-dependency that does not contain all attributes in each functional dependency on *R*. Thus, P-inconsistency cannot occur.

Finally, consider Item 3 in Definition 5.4. If there is a non-trivial P-dependency of the form $X\ T \Rightarrow A$ in G, such that X or A are non-homogeneous and there is no $Y \subset X$ that $Y\ T \Rightarrow A$, and the P-dependency contains all attributes in R, then there is no non-trivial P-dependency which does not contain all attributes in other dependency on R. Then, P-inconsistency cannot occur. Therefore, PCNF avoids P-inconsistency. □

5.4.2 Third Normal Form for Temporal Relations

Anomalies addressed in non-temporal relations can also occur in temporal relations. For example, consider the relation *AccInterest* in Figure 5.3. The information, 'the interest rate for $500 minimum balance during [2/16/96, 3/15/96] was 5%', cannot be inserted if there is no account with $500 minimum balance during [2/16/96, 3/15/96]. Similar to anomalies in non-temporal relations, this anomaly is caused by the transitive dependency which arises from *Acc# T* $\rightarrow$ *MinimumBalance* and *MinimumBalance T* $\Rightarrow$ *InterestRate*.

The anomalies in temporal relations can be avoided by preventing transitive or partial dependencies caused by both functional dependencies and P-dependencies. For example, the relation *AccInterest* is decomposed into the relations *AccBalance* and *BalanceRate*, as shown in Figure 5.4. The transitive dependency caused by *Acc# T* $\rightarrow$ *MinimumBalance* and *MinimumBalance T* $\Rightarrow$ *InterestRate* is eliminated in the two relations. Then, the information, 'the interest rate for $500 minimum balance during [2/15/96-3/15/96] was 5%', can be inserted in the relation *BalanceRate* without any problem. The traditional

3NF can be modified to apply for temporal relations as follows:

Definition 5.5: 3NF for temporal relations

A relation scheme *R* is in *3NFT* (*3NF for temporal relations*) with respect to a set of dependencies *G* if *R* is in BTNF and there is no non-prime attribute in *R* which is transitively dependent or partially dependent, based on both functional dependencies and P-dependencies, on the key of *R*.

The definitions of key and non-prime attribute in temporal relations are the same as those in non-temporal relations. Next, we give examples of relations in 3NFT.

Example 5.4: The relation *Bank* in Example 5.2 can be decomposed into the relations *ANM*(*Acc# Name MinimumBalance T*), *Address*(*Name Addr T*), and *Rate* (*MinimumBalance InterestRate T*), which are in 3NFT.

5.4.3 Relationships between P-consistency Normal Form and Third Normal Form for Temporal Relations

Notice that a 3NFT relation is not necessarily a PCNF relation, and vice versa. For example, the relation *ANM* (*Acc# Name MinimumBalance T*) in Example 5.4 is in 3NFT, but it is not in PCNF because *Acc# T* $\Rightarrow$ *Name* and *Acc# T* $\rightarrow$ *MinimumBalance* are in the same relation and *MinimumBalance* is non-homogeneous. On the other hand, the relation *Customer*(*Acc# Name Addr T*) under a set of dependencies {*Acc# T* $\Rightarrow$ *Name*, *Name T* $\rightarrow$ *Addr*}, in Example 5.3, is in PCNF. However, it is not in 3NFT because *Addr* is transitively dependent on *Acc#* and *T*.

It is desirable to have a relation in both PCNF and 3NFT. We will discuss a normalization algorithm that gives relations in both PCNF and 3NFT in Section 5.6. The problem of normalization into PCNF is NP-complete if there are equivalent sets of attributes in the relation. We give a heuristic for decomposition into PCNF and 3NFT in polynomial time. Before we can discuss normalization into PCNF and 3NFT, we need to examine equivalent sets of attributes and minimal covers for functional dependencies and P-dependencies, which are necessary for normalization into PCNF.

5.5 Equivalent Sets of Attributes and Minimal Covers of a Set of Functional Dependencies and P-Dependencies

Equivalent sets of attributes under a set of functional dependencies and P-dependencies can be defined in a similar manner to those for a set of functional dependencies. X and Y are functionally equivalent, denoted by $X\ T \leftrightarrow Y\ T$, under a set of functional dependencies if $X\ T \rightarrow Y$ and $Y\ T \rightarrow X$. X and Y are P-equivalent, denoted by $X\ T \Leftrightarrow Y\ T$, under a set of functional dependencies and P-dependencies if $X\ T \Rightarrow Y$ and $Y\ T \Rightarrow X$. Next, we give examples of equivalent sets of attributes.

Example 5.5: Consider a relation *TaxInterest* and a set of dependencies G shown below.

$$G = \{\ \textit{MinimumBalance}\ T \Rightarrow \textit{AccountType PercentTaxWithheld InterestRate},$$
$$\textit{AccountType}\ T \Rightarrow \textit{MinimumBalance PercentTaxWithheld},$$
$$\textit{AveBalance InterestRate}\ T \rightarrow \textit{Interest}\,\}.$$

Here, *MinimumBalance* $T \Rightarrow$ *AccountType* and *AccountType* $T \Rightarrow$ *MinimumBalance* under G. Therefore, *MinimumBalance* and *AccountType* are P-equivalent. Since a P-dependency implies a functional dependency, *MinimumBalance* and *AccountType* are also functionally equivalent.

A minimal cover of a set of functional dependencies and P-dependencies can be defined in a similar way to a minimal cover of a set of functional dependencies. That is, a set of dependencies G is a minimal cover if it is reduced and non-redundant, and the right-hand side of each dependency in G contains only one attribute. When there are equivalent sets of attributes under a set of dependencies, there are more than one minimal cover for the set of dependencies. The next example shows two different minimal covers of G in Example 5.5.

Example 5.6: The following sets of dependencies, G_1 and G_2, are two different minimal covers of G in Example 5.5.

$G_1 = \{$ *MinimumBalance T* $\Rightarrow$ *AccountType*,

MinimumBalance T $\Rightarrow$ *PercentTaxWithheld*,

MinimumBalance T $\Rightarrow$ *InterestRate*,

AccountType T $\Rightarrow$ *MinimumBalance*,

AveBalance InterestRate T $\rightarrow$ *Interest* $\}$.

$G_2 = \{$ *MinimumBalance T* $\Rightarrow$ *AccountType*,

AccountType T $\Rightarrow$ *PercentTaxWithheld*,

AccountType T $\Rightarrow$ *InterestRate*,

AccountType T $\Rightarrow$ *MinimumBalance*, *AveBalance InterestRate T* $\rightarrow$ *Interest*}.

Two different minimal covers of the same set of dependencies give the same decomposition for 3NFT. For example, based on G_1 and G_2, the relation *TaxInterest* can be decomposed into the relations *TaxInterestRate* (*MinimumBalance AccountType PercentTaxWithheld InterestRate T*) and *BalanceInt* (*AveBalance InterestRate Interest T*). However, a different minimal cover can give a different decomposition for PCNF. Given that *MinimumBalance*, *AveBalance* and *Interest* are non-homogeneous and *AccountType*, *InterestRate*, and *PercentTaxWithheld* are homogeneous. Based on G_1, the relation *TaxInterest* can be decomposed into R_1(*MinimumBalance AccountType T*), R_2(*MinimumBalance PercentTaxWithheld T*), R_3(*MinimumBalance InterestRate T*), and R_4(*AveBalance InterestRate Interest T*). However, based on G_2, the relation *TaxInterest* can be decomposed into R_5 (*MinimumBalance AccountType T*), R_6 (*AccountType PercentTaxWithheld InterestRate*), and R_7(*AveBalance InterestRate Interest T*). The latter decomposition is more desirable because it gives a smaller number of relations.

If a non-homogeneous set of attributes is P-equivalent to a homogeneous set of attributes, the non-homogeneous set can be replaced by the homogeneous set. If a P-dependency with non-homogeneous attributes can be substituted by a P-dependency without non-homogeneous attributes, the number of relations in PCNF may be reduced. For example, *MinimumBalance T* $\Rightarrow$ *PercentTaxWithheld* and *MinimumBalance T* $\Rightarrow$ *InterestRate* in G_1 can be replaced by *AccountType T* $\Rightarrow$ *PercentTaxWithheld* and *AccountType T* $\Rightarrow$ *InterestRate* because *MinimumBalance* and *AccountType* are P-equivalent. This new minimal cover, G_2, as shown in Example 5.6, gives a better decomposition of the relation

TaxInterest. In this section, we show that finding equivalent sets of attributes is an NP-complete problem. Furthermore, we show that the problem of finding a dependency to replace another dependency in a minimal cover is also NP-complete.

5.5.1 Finding Equivalent Sets of Attributes

For functional dependencies on non-temporal relations, the problem of determining if two given sets of attributes are equivalent has polynomial-time complexity [9]. The problem of finding equivalent sets of attributes for a given set of attributes has not been previously addressed. Here, we show that finding equivalent sets of attributes is an NP-complete problem. First, we introduce the notion of determinative, which can be used to derive equivalent sets of attributes. The determinative of the attribute A in the set X contains subsets of the closure of X which determine A. The determinative is formally defined as follows:

Definition 5.6: Determinative

Let R be a relation scheme, X be a set of attributes in R, A be an attribute in X, and G be a set of dependencies on R. The *functional determinative* of A in X under G, denoted by DF(A, X, G), is $\{Y \mid Y \subseteq X_F^+, Y\ T \rightarrow A$ under $G\}$. The *property determinative* of A in X under G, denoted by DP(A, X, G), is $\{Y \mid Y \subseteq X_P^+, Y\ T \Rightarrow A$ under $G\}$.

Following is an example of determinatives.

Example 5.7: Consider the relation *TaxInterest*, the set of dependencies G in Example 5.5, and a set of attributes X = {*MinimumBalance*, *Interest*}. (*AccountType*) is in

DP(MinimumBalance, X, G), and (*Interest*) is in *DP(Interest, X, G)*.

A set of attributes which is equivalent to a given set X can be created from the determinative of each attribute in X. Consider the relation *TaxInterest* in Example 5.7. We can find a set of attributes which is P-equivalent to (*MinimumBalance Interest*) by choosing one member from DP(*MinimumBalance, X, G*), e.g. (*AccountType*), and one member from DP(*Interest, X, G*), e.g. (*Interest*). Thus, (*AccountType Interest*) is P-equivalent to (*MinimumBalance Interest*). Now, we prove that an equivalent set of attributes of X can be created from the determinative of each attribute in X.

Lemma 5.2: Let R be a relation scheme, X and Y be sets of attributes in R, and G be a set of dependencies on R. $X\,T \leftrightarrow Y\,T$ (or $X\,T \Leftrightarrow Y\,T$) under G if and only if $Y{=}\cup_{A\in X} Y_A$ where $Y_A \in \mathrm{DF}(A, X, G)$ (or $Y_A \in \mathrm{DP}(A, X, G)$, respectively) for each attribute A in X.

Proof:

First, we prove the "if" part. Let $Y_A \in \mathrm{DF}(A, X, G)$ for each A in X, and $Y{=}\cup_{A\in X} Y_A$. $X\,T \rightarrow Y_A$ and $Y_A\,T \rightarrow A$ because $Y_A \in \mathrm{DF}(A, X, G)$. Since $X\,T \rightarrow Y_A$ for all A in X and $Y{=}\cup_{A\in X} Y_A$, $X\,T \rightarrow Y$. Since $Y_A\,T \rightarrow A$ for all A in X and $Y{=}\cup_{A\in X} Y_A$, $Y\,T \rightarrow X$. Thus, $X\,T \leftrightarrow Y\,T$ under G. Similarly, we can prove the "if" part for P-equivalent sets of attributes.

Second, we prove the "only if" part. Let $X\,T \leftrightarrow Y\,T$ under G. Since $X\,T \rightarrow Y$, $Y \subseteq X_F^{+}$. Since $Y\,T \rightarrow X$, for each attribute A in X, there is $Y_A \subseteq Y$ such that $Y_A\,T \rightarrow A$. That is, there is $Y_A \in \mathrm{DF}(A, X, G)$ for each attribute A in X and $Y{=}\cup_{A\in X} Y_A$. Similarly, we can prove the "only if" part in a P-equivalent set of attributes. $\square$

Next, we show that finding functionally equivalent or P-equivalent sets of attributes is an NP-complete problem. The problem of finding equivalent sets of attributes is formally defined as follows:

Equivalent set of attributes problem (EQ):

Input: *A* relation scheme R, a set of attributes H, a set of dependencies G, and a set of attributes X, such that $X\, T \rightarrow A$ (or $X\, T \Rightarrow A$) is in G.

Question: Find $Y \subseteq H$ such that $X \neq Y$, $X\, T \leftrightarrow Y\, T$ ($X\, T \Leftrightarrow Y\, T$) under G.

We show that EQ is NP-complete by showing a reduction of the hitting set problem [42], which is an NP-complete problem, to EQ. The hitting set problem is defined as follows:

Hitting set problem (HS):

Input: A collection of sets C_1, C_2,..., C_n, where $C_i \subseteq$ a finite set S, for i=1, 2,..., n, and a positive integer $K \leq |S|$.

Question: Find a set $S' \subseteq S$ such that $|S'| \leq K$ and for i=1, 2,..., n, there is $c_i \in C_i$ such that $c_i \in S'$.

We consider a special case of EQ, where every dependency is a P-dependency whose left-hand side contains only one time-varying attribute. In the proof, we assume that each set C_i in HS is the determinative of an attribute in X. A set of dependencies can be created from the determinatives of attributes in X. According to Lemma 5.2, the hitting set of the determinatives of attributes in X is P-equivalent to X. Thus, there is a polynomial-time reduction of HS to the special case of EQ, and EQ is NP-complete.

Lemma 5.3: EQ is NP-complete.

The proof is shown in Appendix B.

From Lemma 5.3, we see that, given a set of functional dependencies on a non-temporal relation, finding equivalent sets of attributes for a given set is also NP-complete. Next, we examine minimal covers for a set of functional dependencies and P-dependencies.

5.5.2 Minimal Covers for a Set of Functional Dependencies and P-dependencies

As shown earlier, a minimal cover of a set of functional dependencies and P-dependencies is similar to that of a set of functional dependencies. It is formally defined as follows:

Definition 5.7: Minimal covers

Let R be a relation scheme, G be a set of dependencies on R, and X and Y be sets of attributes in R. G is a *minimal cover* if G is reduced, non-redundant and, for each $X\ T \rightarrow Y$ (or $X\ T \Rightarrow Y$) in G, Y contains only one attribute.

The algorithm that creates a minimal cover for a set of functional dependencies can be easily modified for it to be applicable to a set of functional dependencies and P-dependencies because the inference rules for functional dependencies and those for P-dependencies are similar. We will not discuss the algorithm here. It is easy to see that a minimal cover of a set of functional dependencies and P-dependencies can be created in polynomial time.

5.5.3 SUBSTITUTING DEPENDENCIES IN A MINIMAL COVER

As shown earlier, one minimal cover can give a better decomposition than another minimal cover. A better decomposition can be obtained if a P-dependency with non-homogeneous attributes in a minimal cover can be replaced by a dependency without non-homogeneous attributes. In this section, we examine when a dependency in a minimal cover can be replaced by other dependency (or dependencies). Then, we show the problem of finding a replacement of a dependency is an NP-complete problem.

As shown in Example 5.6, there are possibly more than one minimal cover for a set of dependencies. A different minimal cover can be obtained by substituting a dependency in a minimal cover by other dependency or dependencies. In Example 5.6, a dependency *MinimumBalance T* $\Rightarrow$ *PercentTaxWithheld* is replaced by a dependency *AccountType T* $\Rightarrow$ *PercentTaxWithheld*. We call a dependency that can be replaced by a single dependency a ***1-1 substitutable dependency***, and the dependency with which it is replaced a ***substitute dependency***.

Some of the dependencies in a minimal cover cannot be replaced by a single dependency, but can be replaced by a set of dependencies. For example, consider a minimal cover *G*={*EmployeeID T* $\Rightarrow$ *LicensePlate*, *LicensePlate T* $\Rightarrow$ *AutoID*, *AutoID T* $\Rightarrow$ *EmployeeID*} on a relation *Auto*(*EmployeeID LicensePlate AutoID T*). The dependency *EmployeeID T* $\Rightarrow$ *LicensePlate* cannot be substituted by one dependency. However, it is possible to substitute the dependency by the dependencies *EmployeeID T* $\Rightarrow$ *AutoID* and *AutoID T* $\Rightarrow$ *LicensePlate*. We call a dependency that can be replaced by a set of dependencies a *1-n substitutable dependency.* Some of the dependencies in a minimal cover have no equivalent sets of attributes, and thus they are in every possible minimal covers of the set of dependencies. We call this type of dependency a *non-substitutable dependency.* The dependencies in Example 5.1 are non-substitutable dependencies. Next, we formally define 1-1substitutable, 1-n substitutable, and non-substitutable dependencies.

Definition 5.8: 1-1 substitutable, 1-n substitutable and non-substitutable dependencies

Let R be a relation scheme, G be a minimal cover on R, X, Y, and Z be sets of attributes in R, and A, B, and C be attributes in R.

For $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) in G, if there are Y and B such that $X{\neq}Y$ or $A{\neq}B$, and G-$\{X\,T \Rightarrow A$ (or $X\,T \rightarrow A$, respectively)$\}$ $\cup$ $\{Y\,T \Rightarrow B$ (or $Y\,T \rightarrow B$, respectively)$\}$ is a minimal cover of G, then $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) is *1-1 substitutable* in G and $Y\,T \Rightarrow B$ (or $Y\,T \rightarrow B$, respectively) is a *substitute dependency* of $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$, respectively).

$X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) in G is *1-n substitutable* if there are no Z and no C such that $X{\neq}Z$ or $A{\neq}C$, and $G{\equiv}G$-$\{X\,T \Rightarrow A$ (or $X\,T \rightarrow A$, respectively)$\}$ $\cup$ $\{Z\,T \Rightarrow C$ (or $Z\,T \rightarrow C$, respectively)$\}$, but there are Y, B, and a set of dependencies $G'{\neq}\varnothing$ such that $X{\neq}Y$ or $A{\neq}B$, and $G{\equiv}G$-$\{X\,T \Rightarrow A$ (or $X\,T \rightarrow A$, respectively)$\}$ $\cup$ $\{Y\,T \Rightarrow B$ (or $Y\,T \rightarrow B$, respectively)$\}$ $\cup\, G'$.

$X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) in a minimal cover G is *non-substitutable* if for any minimal cover $G'{\equiv}G$, $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) is in G'.

Next, we prove that a dependency is a 1-1 substitutable dependency if there is a set of attributes that are equivalent to the set of attributes in the dependency, and the equivalent set of attributes can be derived without using the 1-1 substitutable dependency.

Lemma 5.4: Let R be a relation scheme, G be a minimal cover of a set of dependencies on R, X and Y be sets of attributes in R, A and B be attributes in R, and $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) be in G. $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) is 1-1 substitutable if and only if there are Y and B such that $X\,T \Leftrightarrow Y\,T$ (or $X\,T \leftrightarrow Y\,T$) and $X\,A\,T \Leftrightarrow Y\,B\,T$ (or $X\,A\,T \leftrightarrow Y\,B\,T$) can be derived from G, and $X\,T \Rightarrow Y$ (or $X\,T \rightarrow Y\,T$) and $Y\,B\,T \Rightarrow A$ (or $Y\,B\,T \rightarrow A$) can be derived from G-$\{X\,T \Rightarrow A$ (or $X\,T \rightarrow A$, respectively)$\}$.

The proof is shown in Appendix C.

Now, we show that the problem of finding a substitute dependency for a given dependency is NP-complete. We define the problem of finding a substitute dependency as follows:

Substitute dependency problem (SUB):

Input: A relation scheme R, a minimal cover G on R, a dependency $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$), where X or A are non-homogeneous, and a set of attributes H.

Question: Find a dependency $Y\,T \Rightarrow B$ (or $Y\,T \rightarrow B$) of $X\,T \Rightarrow A$ (or $X\,T \rightarrow A$) such that $Y \subseteq H$, $B \in H$, and $G = G - \{X\,T \Rightarrow A$ (or $X\,T \rightarrow A)\} \cup \{Y\,T \Rightarrow B$ (or $Y\,T \rightarrow B)\}$.

We prove that SUB is NP-complete by showing a polynomial-time reduction of EQ to a special case of SUB, where the minimal cover contains only 1-1 substitutable or non-substitutable P-dependencies.

Lemma 5.5: SUB is NP-complete.

Proof:

First, we show that a special case of SUB, where G contains only 1-1 substitutable or non-substitutable P-dependencies, is NP-complete. If we non-deterministically pick Y and B in H, it takes polynomial time to determine whether $G \equiv G - \{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$. Therefore, the special case of SUB is in NP.

The input R, G, and H of SUB and EQ are the same. The input $X\ T \Rightarrow A$ of SUB can be derived from X and G of EQ, as shown in T1 in Figure 5.5, by finding a P-dependency in G whose left-hand side is $X\ T$. This can be done in polynomial time. According to Lemma 5.4, $X\ T \Rightarrow A$ is a 1-1 substitutable if and only if there are Y and B such that $X \neq Y$ or $A \neq B$, $X\ T \Leftrightarrow Y\ T$ and $X\ A\ T \Leftrightarrow Y\ B\ T$. And, $Y\ T \Rightarrow B$ is a substitute dependency of $X\ T \Rightarrow A$. Then, for the output, if $Y\ T \Rightarrow B$ is a substitute dependency of $X\ T \Rightarrow A$ such that $X \neq Y$, $X\ T \Leftrightarrow Y\ T$. Thus, there is a polynomial-time reduction of EQ to the special case of SUB. Therefore, the special case of SUB is NP-complete and SUB is also NP-complete.□

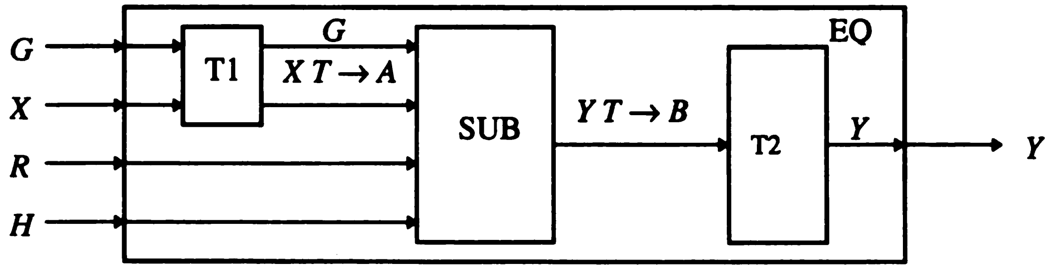

Figure 5.5: The reduction of *EQ* to a special case of *SUB*.

5.6 Normalization into P-Consistency Normal Form and 3NF for Temporal Relations

In this section, we discuss the decomposition of a relation into PCNF and 3NFT relations. The decomposition algorithm for 3NF also decomposes a temporal relation into 3NFT relations. In Section 5.6.1, a normalization algorithm for PCNF and 3NFT is presented. However, this algorithm does not guarantee the minimum number of normalized relations. In Section 5.6.2, we show that finding the decomposition that gives the minimum number of normalized relations is an NP-complete problem. Finally, in Section 5.6.3, we present a heuristic that gives the minimum number of PCNF and 3NFT relations in polynomial time if there is no 1-n substitutable dependency and all equivalent sets of attributes are disjoint.

5.6.1 A Normalization Algorithm

The normalization into PCNF and 3NFT can be done by first decomposing a relation into 3NFT relations, using the 3NF decomposition algorithm. Then, each 3NFT relation is decomposed into PCNF relations. PCNF decomposition can be done by grouping dependencies according to the three items in Definition 5.4. A normalization algorithm is shown below.

Algorithm 1: Normalization into PCNF and 3NFT relations

Input: a set of dependencies *G*.

Output: a set of relation schemes.

Method:

1. Find a minimal cover, *MG*, of *G*.
2. From *MG*, find a set of dependencies, *N*, with equivalent left-hand side.
3. From dependencies in *N*:
   - 3.1. Find functional dependencies with non-homogeneous attributes and the same types of sets of attributes on the left-hand side of the dependencies.
   - 3.2. Find dependencies in which all attributes are contained in every function dependency found in Step 3.1.
   - 3.3. Create one relation scheme from attributes in the dependencies found in Step 3.1 and 3.2.
4. From dependencies in *N*, find dependencies without non-homogeneous attributes. Create one relation scheme from attributes in these dependencies.
5. From dependencies in *N*, find a P-dependency with non-homogeneous attributes and dependencies in which all attributes are contained in the P-dependency. Create one relation scheme from all attributes in these dependencies.
6. Repeat Step 5 for all P-dependencies with non-homogeneous attributes found in *N*.
7. Repeat Step 2-6 until all dependencies in *MG* are considered.
8. If the key of *R* is not contained in any relation created in Step 3-6, create a relation scheme R_0 from the key of *R*.

It is obvious that Algorithm 1 has polynomial-time complexity. Next, we prove that Algorithm 1 gives a dependency-preserving and lossless decomposition for PCNF and 3NFT.

Lemma 5.6: Algorithm 1 gives a dependency-preserving and lossless decomposition for PCNF and 3NFT.

The proof is given in Appendix D.

Algorithm 1 may give a different number of PCNF and 3NFT relations if a different minimal cover is derived in Step 1. Consider the minimal covers G_1 and G_2 in Example 5.6. If G_1 is derived in Step 1, Algorithm 1 decomposes the relation *TaxInterest* into the following relations: R_1(*MinimumBalance AccountType T*), R_2(*MinimumBalance PercentTaxWithheld T*), R_3(*MinimumBalance InterestRate T*), and R_4(*AveBalance InterestRate Interest T*). However, this relation can be decomposed into fewer relations. If G_2 is derived in Step 1, Algorithm 1 decomposes the relation *TaxInterest* into relations R_5(*MinimumBalance AccountType T*), R_6(*AccountType PercentTaxWithheld InterestRate T*), and R_7(*AveBalance InterestRate Interest T*).

Algorithm 1 does not always give the best decomposition, i.e., the decomposition which gives the fewest normalized relations. We show in the next section that the problem of finding such a decomposition is NP-complete.

5.6.2 FINDING A MINIMUM DECOMPOSITION

It is desirable to achieve a decomposition that gives the fewest PCNF and 3NFT relations because it reduces the number of join operations required in queries. We call a decomposition which gives the minimum number of PCNF-3NFT normalized relations a *minimum decomposition*. We will show that finding a minimum decomposition is an NP-complete problem. First, we define the problem as follows:

Minimum decomposition problem (MD):

Input: A relation scheme *R*, a set of dependencies *G* on *R*, and a set of homogeneous attributes *H*.

Question: Find a minimum decomposition of R into PCNF and 3NFT relations.

We show that MD is NP-complete by reducing SUB to a special case of MD, where there are only non-substitutable or 1-1 substitutable P-dependencies on the relation, and no two dependencies have the same substitute dependency. First, we show that, in this special case, a decomposition is a minimum decomposition if, in the minimal cover of each normalized relation, there is no P-dependency with non-homogeneous attributes, that can be replaced by a P-dependency without non-homogeneous attributes.

Lemma 5.7: Let R be a relation scheme, and G be a minimal cover on R, where G contains only P-dependencies which are either non-substitutable or 1-1 substitutable, and no two dependencies can have the same substitute dependency. A decomposition of a relation scheme R into $\{R_1, R_2,\ldots, R_n\}$, where G_1, $G_2,\ldots$, and G_n are minimal covers on $R_1, R_2,\ldots, R_n$, respectively, is a minimum decomposition if and only if, $R_1, R_2,\ldots, R_n$ are created from Algorithm 1, and for each 1-1 substitutable dependency $X\ T \Rightarrow A$ in $G_1 \cup G_2 \cup \ldots \cup G_n$ such that X or A are non-homogeneous, there is no substitute dependency $Y\ T \Rightarrow B$ of $X\ T \Rightarrow A$ such that Y and B are homogeneous.

The proof is shown in Appendix D.

Next, we prove that MD is NP-complete by giving a polynomial-time reduction of SUB to the special case of MD, where the minimal cover contains only 1-1 substitutable or non-substitutable P-dependencies, and no two dependencies can be replaced by the same dependency.

Theorem 5.3: MD is NP-complete.

The proof is shown in Appendix E.

Now that we have proven that finding a minimum decomposition is NP-complete, we need a heuristic that finds a "good" decomposition in polynomial time. In the next section, we present a heuristic that finds a minimum decomposition in most cases.

5.6.3 A Heuristic for Reducing the Number Relations in the Decomposition

In this section, we present a heuristic for PCNF and 3NFT decomposition. This heuristic reduces the number of normalized relations by replacing 1-1 substitutable P-dependencies with non-homogeneous attributes by their substitute dependencies without non-homogeneous attributes. To find 1-1 substitutable dependencies and their substitute dependencies, we must first find equivalent sets of attributes for some sets of attributes in the dependencies. Since finding equivalent sets of attributes is also NP-complete, we first present a heuristic that finds equivalent sets of attributes in polynomial time.

5.6.3.1 A Heuristic for Finding Equivalent Sets of Attributes

In this section, we present a heuristic that finds sets of attributes which are equivalent to a given set of attributes. The heuristic presented here has polynomial-time complexity. However, it cannot find some equivalent sets of attributes that are not disjoint.

To find equivalent sets of attributes for a given set X, we first find the closure of X, called

Y. We then remove one attribute at a time from *Y*. If the remainder of *Y* after removing an attribute is not equivalent to *X*, put the attribute back into *Y* and repeat the process until the smallest set of attributes which is equivalent to *X* is found. In general, there may be more than one equivalent set of attributes. To find another equivalent set, repeat the same process, but remove each attribute in the equivalent set previously found, one at a time. The heuristic is shown below.

```
Algorithm 2: Finding functionally equivalent (or P-equivalent) sets of attributes
Input: A set of dependencies $G$ and a set of attributes $X$.
Output: $EQ=\{Y \mid Y\ T \leftrightarrow X\ T$ (or $Y\ T \Leftrightarrow X\ T$) under $G\}$.
Method:
1.  $XF=\{$attribute $A \mid X\ T \rightarrow A\}$ (or $XP\{$attribute $A \mid X\ T \Rightarrow A\}$); $Z=$ LIST($X$)||LIST($XF$ (or
    $XP$) $-X$); $EQ=\{X\}$; $SUCCESS=T$.
2.  Find if there is any set of attributes $W \neq X$ and $W\ T \leftrightarrow X\ T$ (or $W\ T \Leftrightarrow X\ T$).)
    While ($SUCCESS$)
    2.1  $Y=XF$ (or $XP$);
    2.2  (If $Y$-$A$ is not equivalent to $X$, keep $A$ in $Y$.)
          For each attribute $A$ in $Z$:
         2.2.1 If $(Y$-$A)\ T \leftrightarrow X$ (or $(Y$-$A)\ T \Leftrightarrow X$), then $Y=Y$-$A$.
    2.3  (If new equivalent set of $X$ is found, repeat Step 2.)
         If ($Y \notin EQ$), then $EQ=EQ \cup \{Y\}$; $Z=$LIST($Y$)||LIST($Z$-$Y$).
    2.4  (If there is no new equivalent set of $X$ is found, stop here.)
         If ($EQ=\{X\}$), $SUCCESS=$F.
3.  return($EQ$).
Note: || denotes the concatenation of lists.
```

It is obvious that the sets of attributes in EQ, produced in Algorithm 2, is equivalent to X. In Step 2.2, $Y=Y$-A if $(Y$-$A)\ T \leftrightarrow X$ (or $(Y$-$A)\ T \Leftrightarrow X$). In Step 2.3, Y is added into EQ.

Therefore, each set of attributes in EQ is equivalent to X. Next, we prove that Algorithm 2 has polynomial-time complexity.

Lemma 5.8: Algorithm 2 has polynomial-time complexity.

Proof:

Obviously, Step 1 and 3 in Algorithm 2 take linear time. First, we prove that Step 2 is repeated at most n times, where n is the number of time-varying attributes in the relation scheme R. Let $R\ T$ be the relation scheme. Step 2 is repeated until no new equivalent set of attributes is found. No new attribute is found when the equivalent set of attributes found in the last repetition of Step 2 is Y and all sets of attributes that contain at least one attributes in R-Y are found in previous repetitions. The number of sets of attributes that contain at least one attributes in R-Y that can be found in Step 2 is no greater than n. Therefore, Step 2 is repeated at most n times.

Since Step 2.1, 2.2 and 2.3 take polynomial time, n repetitions of Step 2 takes polynomial time. Thus, Algorithm 2 takes polynomial-time. $\square$

The following example shows the derivation of equivalent sets of attributes, using Algorithm 2.

Example 5.8: Consider a set of dependencies $\{A\ T \rightarrow C,\ A\ B\ T \rightarrow D,\ C\ D\ T \rightarrow A,\ D\ T \rightarrow B\ E,\ E\ T \rightarrow B\}$. The sets of attributes which are functionally equivalent to $X = (A\ B)$ can be derived by using Algorithm 2, as follows. First, compute the functional closure of X, which is $Y = (A\ B\ C\ D\ E)$. In the first try, Z = list of A, B, C, D, E, and we try to remove each attribute in *Z*, starting from the head of the list. *Y-A*, which is (*B C D E*), does functionally determine *X*. Thus, *A* is removed from *Y*, and *Y* = (*B C D E*). Similarly, *B* is removed from *Y* and *Y* = (*C D E*). Next, we try to remove *C*. Since *Y-C* does not functionally determine *X*, *C* cannot be removed. Similarly, *D* cannot be removed from *Y*, but *E* is removed from *Y*. Then, *Y* = (*C D*), and is functionally equivalent to *X*. To find another equivalent set of attributes, the same procedure is repeated with the same *Y* = (*A B C D E*). However, we first try to remove each attribute in (*C D*) which is previously found to be equivalent to *X* first. Thus, *Z* = list of *C*, *D*, *A*, *B*, *E*. We find (*A B*) is equivalent to (*A B*). Since no new equivalent set of attributes is found, we stop. Notice that (*A E*), which is functionally equivalent to (*A B*), cannot be derived in this approach because they are not disjoint.

Next, we present a heuristic for PCNF and 3NFT decomposition that uses Algorithm 2 to reduce the number of relations.

5.6.3.2 Using Equivalent Sets of Attributes to Reduce the Number of Relations

The heuristic presented in this section reduces the number of relations in a decomposition by replacing P-dependencies with non-homogeneous attributes with their substitute dependencies without non-homogeneous attributes.

Algorithm 3: Finding a decomposition for PCNF and 3NFT, and reducing the number of relations.

Input: a set of dependencies *G*.

Output: a set of relation schemes.

Method:

1. Find MG, which is a minimal cover of G.
2. (Replace a 1-1 substitutable P-dependency with non-homogeneous attributes by a P-dependency with only homogeneous attributes.)

   For each dependency $X\ T \Rightarrow A$ in MG such that X or A are non-homogeneous:

   2.1. Find $Y \subseteq H$ and $B \in H$ such that $X\ T \Leftrightarrow Y\ T$ and $X\ A\ T \Leftrightarrow Y\ B\ T$ can be derived from MG, and $X\ T \Rightarrow Y$ and $Y\ B\ T \Rightarrow A$ can be derived from MG-$\{X\ T \Rightarrow A\}$ (use Algorithm 2).

   2.2. If there are such Y and B, let MG=MG-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$.
3. Create relation schemes $\{R_0, R_1, \ldots, R_n\}$, according to Step 2-8 in Algorithm 1

It is clear that Algorithm 3 has polynomial-time complexity. Step 1 takes polynomial time because a minimal cover can be created in polynomial time. Step 2 takes polynomial time because it is based on Algorithm 2 which, according to Lemma 5.8, has polynomial-time complexity, and is performed at most $|G|$ times. Step 3 takes polynomial time because it is based on Algorithm 1 which has polynomial-time complexity. Next, we prove that Algorithm 3 also gives a dependency-preserving and lossless decomposition for PCNF and 3NFT.

Lemma 5.9: Algorithm 3 gives a dependency-preserving and lossless decomposition for PCNF and 3NFT.

Proof:

In Step 2, 1-1 substitutable dependencies are replaced by their substituted dependencies. According to Lemma 5.4, the set of dependencies MG produced in Step 2 is equivalent to the input G. Step 3 is Algorithm 1, and it is proved, in Lemma 5.6, to produce a dependency-preserving and lossless decomposition for PCNF and 3NFT. Therefore, Algorithm 3 gives a dependency-preserving and lossless decomposition for PCNF and 3NFT. □

The following example shows a decomposition of the relation *TaxInterest* which is produced by Algorithm 3 using the minimal cover G_2 in Example 5.6.

Example 5.9: Consider the relation *TaxInterest* in Example 5.6. Suppose the minimal cover G_1 in Example 5.6 is derived in Step 1 in Algorithm 3. In Step 2, ***MinimumBalance T*** $\Rightarrow$ ***PercentTaxWithheld*** and ***MinimumBalance T*** $\Rightarrow$ ***InterestRate*** are substituted by *AccountType T* $\Rightarrow$ *PercentTaxWithheld* and *AccountType T* $\Rightarrow$ *InterestRate*. In Step 3, the relation *TaxInterest* is decomposed into relations R_5*(MinimumBalance AccountType T)*, R_6*(AccountType PercentTaxWithheld InterestRate T)*, and R_7*(AveBalance InterestRate Interest T)*. This decomposition is a minimum decomposition.

Algorithm 3 reduces the number of normalized relations. However, it does not guarantee the minimum decomposition because it only replace 1-1 substitutable dependencies, but not 1-n substitutable dependencies, and Algorithm 2 does not guarantee to find all equivalent sets of attributes. The following example shows a situation when Algorithm 3 does not produce a minimum decomposition because of a 1-n substitutable dependency.

Example 5.10: Let A, B, C, D, E, F, G, and H be homogeneous attributes and J be a non-homogeneous attribute. Consider the relation $S(A\ B\ C\ D\ E\ F\ G\ H\ J\ T)$ and the minimal cover M shown in Figure 5.6. From Algorithm 3, no dependency is replaced in Step 2, and the dependencies are grouped as follows: $\{A\ B\ T \rightarrow G, A\ B\ T \rightarrow H, G\ H\ T \rightarrow A, G\ H$

$T \rightarrow B, G\,H\,T \rightarrow D, G\,H\,T \rightarrow E, D\,E\,T \Rightarrow G, D\,E\,T \Rightarrow H\}$, $\{G\,J\,T \Rightarrow C\}$, $\{H\,J\,T \Rightarrow F\}$, $\{B\,C\,D\,F\,T \Rightarrow J\}$, $\{G\,H\,T \Rightarrow J\}$. In Step 3, S is decomposed into $S_1(A\ B\ D\ E\ G\ H\ T)$, $S_2(C\ G\ J\ T)$, $S_3(F\ H\ J\ T)$, $S_4(B\ C\ D\ F\ J\ T)$, and $S_5(G\ H\ J\ T)$, which is not a minimum decomposition.

$$\begin{aligned}
M = \{&A\,B\,T \rightarrow G, & &A\,B\,T \rightarrow H,\\
&D\,E\,T \Rightarrow G, & &D\,E\,T \Rightarrow H,\\
&G\,H\,T \Rightarrow J, & &G\,J\,T \Rightarrow C,\\
&H\,J\,T \Rightarrow F, & &G\,H\,T \rightarrow A,\\
&G\,H\,T \rightarrow B, & &G\,H\,T \rightarrow D,\\
&G\,H\,T \rightarrow E, & &B\,C\,D\,F\,T \Rightarrow J\}.
\end{aligned}
\qquad
\begin{aligned}
M' = \{&A\,B\,T \rightarrow G, & &A\,B\,T \rightarrow H,\\
&D\,E\,T \Rightarrow G, & &D\,E\,T \Rightarrow H,\\
&A\,B\,T \Rightarrow C, & &D\,E\,T \Rightarrow F,\\
&G\,J\,T \Rightarrow C, & &H\,J\,T \Rightarrow F,\\
&G\,H\,T \rightarrow A, & &G\,H\,T \rightarrow B,\\
&G\,H\,T \rightarrow D, & &G\,H\,T \rightarrow E,\\
&B\,C\,D\,F\,T \Rightarrow J\}.
\end{aligned}$$

Figure 5.6: Minimal covers M and M' on a relation S.

The dependency $G\,H\,T \Rightarrow J$ in M is 1-n substitutable and it can be replaced by $A\,B\,T \Rightarrow C$ and $D\,E\,T \Rightarrow F$. A minimal cover M', shown in Figure 5.6, can be created from M by replacing $G\,H\,T \Rightarrow J$ in M with $A\,B\,T \Rightarrow C$ and $D\,E\,T \Rightarrow F$. Using M', the relation r can be decomposed into the relations $S_6(A\ B\ D\ E\ G\ H\ T)$, $S_7(C\ G\ J\ T)$, $S_8(F\ H\ J\ T)$, and $S_9(B\ C\ D\ F\ J\ T)$, which is a minimum decomposition.

5.7 Conclusions

Existing design principles for temporal databases are not applicable when non-homogeneous data are involved. In this chapter, we address an inconsistency problem in temporal relations with homogeneous and non-homogeneous data, called P-inconsistency. P-inconsistency occurs when there are both a P-dependency and another dependency with non-homogeneous attributes in the same relation. We propose a normal form called PCNF which avoids P-inconsistency. It is desirable to avoid both P-inconsistency and anomalies caused by transitive and partial dependencies. We study the decomposition for PCNF as well as 3NFT, which avoids transitive and partial dependencies, and prove that the problem of finding a minimum decomposition is NP-complete. In the process, we also prove that the problem of finding equivalent sets of attributes is an NP-complete problem. A polynomial-time heuristic which decomposes a relation into PCNF and 3NFT relations is presented.

Chapter 6

EXTENSION OF DATALOG FOR PROPERTY AND REPRESENTATIVE RELATIONS

6.1 Introduction

There are a few extensions of Datalog [81] to support temporal data, such as Templog [1, 7], Datalog$_{1s}$ [19, 20], and Temporal DATALOG [63]. In these deductive databases, temporal data are associated with points of time. Representative relations cannot be represented in these data models. For example, the highest temperature over a time interval is associated with the whole interval, but not with each point in the interval, and thus it cannot be represented in these extensions of Datalog. In this chapter, we propose an extension of Datalog, called *TDatalog*, which supports both property and representative relations.

In TDatalog, a temporal data is associated with a time interval. The implicit relationship between time intervals associated with temporal data in an IDB rule is adopted from a type of if-statement called a generalization if-statement [31]. In a generalization if-statement, it is assumed that the consequence is true when all the conditions are true (unless specified otherwise). Thus, in an IDB rule, the time interval associated with the head can be derived from the intervals associated with predicates in the body, based on this assumption, and the types of temporal data in the body. Furthermore, explicit relationship between intervals associated with temporal data in an IDB rule can be specified by temporal predicates. Because of the decomposability of the intervals associated with the two types of temporal data, the interpretations of these temporal predicates also depend on types of the temporal data, as shown in Chapter 4.

The focus in this chapter is to extend Datalog to capture both property and representative relations. In this extension of Datalog, the relationship between time in IDB rules is implicit, and can be derived from the rules, based on types of temporal data in the rules. The rest of this chapter is organized as follows. In Section 6.2, the relationship between time in if-statements are examined, and applied in the extension of Datalog. Based on this implicit relationship between time, TDatalog is presented in Section 6.3. Section 6.4 discusses the translation of TDatalog to relational algebra presented in Section 4.4. Section 6.5 presents the conclusion.

6.2 Relationships between Time in If-statements

The similarity between if-statements and IDB rules in Datalog is obvious. However, the relationship between time in if-statements and IDB rules in temporal deductive databases has not been previously addressed. Based on the tense patterns in if-statements, if-statements can be classified into 3 types: *hypothetical*, *generalization*, and *conditional if-statements* [31]. In this section, the relationships between time in hypothetical and generalization if-statements are examined, and they are compared to the relationship between time in IDB rules in temporal databases. Conditional if-statements are not discussed here because they involve modality which is not in the scope of this study. The relationship between time in generalization if-statements is adopted for TDatalog.

6.2.1 Hypothetical If-statements

In a hypothetical if-statement, "if" acts as conjunction, and thus the condition and the consequence are two independent clauses. The time of the conditions and the time of the consequences are independent, and are indicated by tenses or time adverbial phrases. An example of a hypothetical if-statement is shown in Example 6.1.

Example 6.1: The following if-statement is a hypothetical if-statement because the tense of the condition and the tense of the consequence are independent.

If John left on Monday, he will arrive on Friday.

The time of the consequence, which is indicated by future tense "*will arrive*" and the phrase "*on Friday*," does not depend on the time of the condition, which is indicated by past tense "*left*" and the phrase "*on Monday*."

The relationship between time in a hypothetical if-statement is explicitly stated. Similarly, the relationship between time in an IDB rule can be explicit. For example, in Datalog_{1s} rules, the relationship between time is specified explicitly. The if-statement in Example 6.1 can be expressed in Datalog_{1s}, as follows:

arrive("John", T+4) :- leave("John", T), T = Monday.

Next, we examine another type of if-statement, called generalization if-statement.

6.2.2 Generalization If-statements

In a generalization if-statement, the if clause acts as an adverbial modifier, and the time of the consequence depends on the time of the condition. A generalization if-statement describes "the generalization about the occasion of condition's satisfaction" [31]. Thus, the time when the consequence is true depends on the time when the condition is true. The following example shows a generalization if-statement.

Example 6.2: The following is a generalization if-statement.

A student is a senior if he/she completes more than 100 credits. [S1]

In this sentence, the tenses in the condition and the consequence are the same. Thus, the time of the consequence is the same as the time of the condition, and the explicit relationship between time can be omitted. However, the relationship between time in this type of if-statement also depends on types of temporal data and temporal modifiers in the statement. These factors are discussed next.

6.2.2.1 Effects of Types of Temporal Data

As mentioned earlier, the consequence in a generalization if-statement is true when all the conditions are true. However, the characteristic of a temporal data determines when the temporal data is true. A property data is true for an interval as well as its sub-intervals, while a representative data is true for an interval and not necessarily for its sub-intervals. Thus, in a generalization if-statement, the time when the consequence is true also depends on the types of temporal data in the condition. Consider the following three generalization if-statements.

A stock is stable if the difference between the highest and the lowest price is less than 20 points. [S2]

A student is a senior if he/she completes more than 100 credits and his/her GPA is higher than 1.0. [S3]

A stock is unstable if the difference between the highest and the lowest price is more than 100 points and it is an entertainment business. [S4]

The highest price and the lowest price of a stock are representative data. The number of credits completed, GPA, and the type of business are property data. In [S2], the highest price and the lowest price are applied to the whole interval. They are true at the same time if the time intervals associated with both of them are exactly the same. And, the consequence "*A stock is stable*" is true for that time interval. Now, consider [S3] in which both conditions are property data. Since the number of credits and GPA are applied to all sub-intervals associated with the data, they are both true during the intersection of the intervals. Thus, the consequence "*A student is a senior*" is true during the intersection of the two intervals. Finally, for [S4], two conditions are representative data and another is a property data. All conditions are true when the time intervals associated with representative data are equal, and are contained in the interval associated with the property data. This type of implicit relationship between time is adopted for the extension of Datalog presented in Section 6.3.

Another factor that effects the relationship between time in generalization if-statements is the roles of conditions in if-statements, which are examined next.

6.2.2.2 Effects of Roles of Conditions

In any statements, clauses have different roles. For example, the statement "*John works for a company which is located in Canada*" is composed of two clauses; "*John works for a company*" and "*which* (*the company*) *is located in Canada*." The first clause is the main clause, and the second is the modifier of the main clause. In generalization if-statements, main clauses and modifiers have different effects on the relationship between time. Consider the following generalization if-statement.

> *A student's tuition for a semester is pre-paid if it is paid for before he/she registers for the semester.* [S5]

There are two clauses in the if part of this statement; "*it* (*a student's tuition for a semester*) *is paid for*" and "*before he/she* (*the student*) *registers for the semester.*" The first clause is the main condition, and the second clause is a modifier which specifies when the main condition is true. The first clause is called a ***main condition***, and the second a ***modifier***. Only the first clause directly determines when the consequence is true. The time when the consequence "*a student's tuition for a semester is pre-paid*" is the time when the main condition "*the tuition is paid for*" is true. According to the modifier "*before the student registers for the semester*", the time when the main condition itself is true must be before the time that the student registers for the semester.

The relationship between time in generalization if-statements is adopted in TDatalog, which is an extension of Datalog presented in the next section.

6.3 TDatalog: an Extension of Datalog

In this section, an extension of Datalog, called TDatalog, is presented. Similar to Datalog, there are two types of databases in TDatalog; an extensional database (EDB) and an intensional database (IDB). Predicates defined in EDB and IDB represent temporal data, and these predicates are called *data predicates*. A major difference between TDatalog and Datalog$_{1s}$ is the way the relationship between time in IDB rules is defined. The relationship between time in a Datalog$_{1s}$ rule, which is analogous to the relationship between time in a hypothetical if-statement, is explicitly defined through the temporal arguments in the rule. On the other hand, the relationship between time in a TDatalog rule, which is analogous to a generalization if-statement, is implied from the rule, and types of temporal data in the rule. Furthermore, temporal data in Datalog$_{1s}$ are associated with time points, while temporal data in TDatalog are associated with time intervals. This allows TDatalog to capture representative relations. Next, data predicates defined in EDB and IDB are discussed.

6.3.1 Extensional Databases

EDB predicates in TDatalog represent temporal data. An EDB predicate in TDatalog is similar to that in Datalog, except for the additional argument, i.e. the valid time, associated with an EDB predicate in TDatalog. In Datalog$_{1s}$, a time point is added as another argument in a predicate. In TDatalog, a time interval, instead of a point, is associated with a predicate because, in a representative relation, a tuple is associated with the whole interval, but not its sub-intervals. The syntax of an EDB predicate is shown below.

> *predicate-symbol*(*arguments*): *t*., where *arguments* is the list of values of the arguments for the predicate, and *t* is a time interval.

The type of a temporal data represented by an EDB predicate is defined in the *type* predicate, as shown below.

> *type*(*predicate-symbol*, *data-type*)., where *predicate-symbol* is a predicate symbol representing a temporal data, and *data-type* is the type of the temporal data, i.e. *property* or *representative*.

The following example shows an extensional database.

Example 6.3: Consider the temporal data in [S2] and [S4]. The predicate *company* represents a property data "the type of business of a company", and the predicates *highestprice* and *lowestprice* represent representative data "a stock's highest price" and "a stock's lowest price", respectively.

> *type*(*company*, *property*).
>
> *type*(*highestprice*, *representative*).
>
> *type*(*lowestprice*, *representative*).
>
> *company*(*microsoft*, *software*): [*1/1/75*, *12/31/97*].

company(*pentax*, *lenses*): [*1/1/19*, *12/31/97*].

company(*pentax*, *camera*): [*1/1/71*, *12/31/97*].

highestprice(*microsoft*, *150*):[*7/1/97*, *7/31/97*].

lowestprice(*microsoft*, *120*): [*7/1/97*, *7/31/97*].

highestprice(*pentax*, *70*):[*7/1/97*, *7/31/97*].

lowestprice(*pentax*, *60*): [*7/1/97*, *7/31/97*].

Notice that time intervals are distinguished from other arguments. The time intervals have pre-determined meaning according to the type of the data predicates associated with the intervals. On the other hand, other arguments can have any user-defined meaning. Furthermore, the time interval can be omitted when the predicate is used in IDB rules, which will be discussed later, while other arguments cannot. Next, IDB rules are defined.

6.3.2 Intensional Databases

Some data predicates in TDatalog are defined by rules in intensional databases. Similar to EDB predicates, types of these predicates must also be defined in a type predicate in an IDB. For example, the type of the predicates *stable* and *unstable*, representing the property fact "a stock is stable" in [S2] and the representative fact "a stock is unstable" in [S4], can be defined in TDatalog as follows:

type(*stable*, *property*).

type(*unstable*, *representative*).

Similar to Datalog, a rule in TDatalog is composed of the head and the body. A head contains an IDB predicate, and the body is the conjunction of data predicates and temporal predicates. The temporal predicates defined in Section 4.2, e.g. *before*, *after*, etc., are adopted in TDatalog. The syntactical difference between the temporal predicates in Section 4.2 and those in TDatalog is the arguments in the temporal predicates. An argument of a temporal predicate in Section 4.2 is either a time constant or a valid time. In TDatalog, an argument of a temporal predicate is either a time constant or a data predicate with the omission of the time. A data predicate in a temporal predicate refers to the time interval associated with the data predicate. The semantics of temporal predicates in TDatalog also depends on the types of temporal data in the temporal predicate, as shown in Chapter 4. The following example shows temporal predicates in TDatalog.

Example 6.4: Consider the EDB predicates in Example 6.3. The following are examples of temporal predicates in TDatalog.

equal (*company* (*pentax, camera*), *highestprice* (*pentax, 70*)) [P1]

equal (*highestprice* (*microsoft, 150*), *lowestprice* (*microsoft, 120*)) [P2]

A temporal predicate in a rule specifies the relationship between intervals associated with two temporal data. For example, [P1] indicates that the predicates *company* (*pentax, camera*) and *highestprice* (*pentax, 70*) are true at the same time, and [P2] indicates that the predicates *highestprice* (*microsoft, 150*) and *lowestprice* (*microsoft, 120*) are true at the same time. As a result, the valid time of a data predicate is restricted by a temporal predicate. For example, the time interval associated with the predicate *company* (*pentax, camera*) that satisfies [P1] is the sub-interval which is equal to the time interval associated with the predicate *highestprice* (*pentax, 70*). Thus, the valid time of the predicate *company* (*pentax, camera*) that satisfies [P1] is not the whole interval of

[1/1/*71-12/31/97*], but a sub-interval [*7/1/97-7/31/97*]. On the other hand, in [P2], the predicate ***highestprice*** is a representative data and the interval associated with the predicate cannot be decomposed. Thus, the whole interval of time associated with the predicate ***highestprice*** (*pentax, 70*) must satisfy [P2].

The relationship between time in a rule can be derived, based on the relationship between time in generalization if-statements discussed in Section 6.2.2. First, find main conditions in the rule. In an IDB rule, if a data predicate is an argument of a temporal predicate in the body, but not present outside temporal predicates, the data predicate is a modifier. If a data predicate is present outside temporal predicates in an IDB rule, it is a main condition. For example, consider the following TDatalog rule.

> *tuition(Student, Semester, pre-paid) :- paid(Student, Semester), before (paid (Student, Semester), register(Student, Semester)).*

The predicate *paid* (*Student, Semester*) is a main condition, and the predicate *register*(*Student, Semester*) is a modifier.

Second, if a main condition has a modifier, find the valid time of the main condition which satisfies the modifier. The time associated with the head of the rule is determined by the time associated with main conditions in the body, as follows:

> Let P_1 and P_2 be predicates representing property data, R_1 and R_2 be predicates representing representative data, H be the head of an IDB rule, and TP_1, TP_2, TR_1, TR_2, and TH be the time intervals associated with P_1, P_2, R_1, R_2, and H, respectively.

- For the rule H :- P_1, P_2, $TH = TP_1 \cap TP_2$ if $TP_1 \cap TP_2 \neq \emptyset$.

- For the rule H :- R_1, R_2, $TH = TR_1$ if $TR_1 = TR_2$.

- For the rule H :- P_1, R_2, $TH = TR_2$ if TR_2 is contained in TP_1.

The following example shows the relationship between time derived from TDatalog rules.

Example 6.5: [S2], [S3], and [S4] can be expressed in TDatalog as follows:

stable(X) :- highestprice(X, Y), lowestprice(X, Z), Y-Z<20. [R1]

status(X, "senior") :- credit(X, Y), Y>100, GPA(X, Z), Z>1.0. [R2]

unstable(X) :- highestprice(X, Y), lowestprice(X, Z), Y-Z>100, company(X, "entertainment"). [R3]

In [R1], *highestprice* and *lowestprice* are representative data. The time associated with *stable(X)* is the interval associated with *highestprice(X, Y)*, and the intervals associated with *highestprice(X, Y)* and *lowestprice(X, Z)* are equal. In [R2], *credit* and *GPA* are representative data. The time associated with *status(X, "senior")* is the intersection of the intervals associated with *credit(X, Y)* and *GPA(X, Z)*, where the intersection is not empty. In [R3], the time associated with *unstable(X)* is the time interval associated with *highestprice(X, Y)*, if the intervals associated with *highestprice(X, Y)* and *lowestprice(X, Z)* are equal, and contained in the interval associated with *company(X, "entertainment")*.

It is clear from the discussion in this section that TDatalog is closely related to the extension of relational algebra presented in Chapter 4. Next, the translation from TDatalog into the algebra is presented.

6.4 Translation from TDatalog to Relational Algebra

A relation defined by a Datalog rule can be calculated from a relational algebra expression, which is translated from the rule [81]. Similarly, a temporal relation defined in a TDatalog rule can be calculated by translating the rule into the relational algebra presented in Chapter 4. Then, the relational algebra is used to calculate the relation. The translation is similar to the translation of Datalog into relational algebra, except for the valid time and the temporal predicates which are not present in Datalog. Time-varying attributes are handled the same way as the translation from Datalog to relational algebra, and will be omitted here.

An IDB rule can be translated into a relational algebra expression as follows. For each temporal predicate $p(Q, R)$ in a rule, create a new relation from Q and R, i.e. $Q \bowtie_{p(Q.T,\ R.T)} R$, where T denotes the valid time. Notice that the valid time in this new relation is created from the valid time in Q and R, as shown in Table 4.2. Then, join the relations created for temporal predicates and the relations that represent the main conditions in the rule. The following example shows the translation of a TDatalog rule into a relational expression.

Example 6.6: Consider the rule P :- Q, $p(Q, R)$, S., where P, Q, R, and S are temporal relations and p is a temporal predicate. The relation P can be created by first creating a relation $Q' = Q \bowtie_{p(Q.T,\ R.T)} R$. Then, create a relation Q'' by joining Q' and Q when $equal(Q'.T, Q.T)$. That is, $Q'' = Q' \bowtie_{equal(Q'.T,\ Q.T)} Q$. Finally, P is created by joining Q'' and S when $equal(Q''.T, S.T)$. That is, $P = Q'' \bowtie_{equal(Q''.T,\ S.T)} S$.

A relation defined by a recursive rule in TDatalog can be calculated similar to the evaluation of a recursive rule in Datalog. A relation defined by a recursive rule is created by repeatedly calculating the relation from the relational algebra expression until no new tuple is created.

6.5 Conclusions

In this chapter, an extension of Datalog, called TDatalog, is presented. The relationship between time in TDatalog rules is adopted from the relationship between time in generalization if-statements. The characteristics of property and representative data are incorporated in TDatalog rules, as shown in Section 6.2.2.1. The calculation of relations defined in TDatalog rules is discussed in Section 6.4.

There are a few advantages of TDatalog rules. First, different characteristics of property and representative data are taken care of automatically in the rules. Second, Datalog rules can also serve as TDatalog rules since the valid time in TDatalog rules is omitted. The valid time of a predicate defined in a rule can be derived based on types of temporal data in the rule.

Imprecise valid time is another important aspect in temporal databases. In the next chapter, temporal predicates, defined in Section 4.2, are extended to handle imprecise valid time in property and representative relations.

Chapter 7

RELATIONAL ALGEBRA FOR TEMPORAL RELATIONS WITH IMPRECISE VALID TIME

7.1 Introduction

Imprecise information has been addressed in many database research [15, 43, 51, 84, 85, 88][51]. In temporal databases, the valid time itself can be imprecise. For example, the exact year that King Tutankhamen reign is unknown. It is said that he reigned during 1347-1339 BC, 1334-1325 BC, or 1336-1327 BC. However, this imprecise information is useful in many queries. For example, we can find Egyptian Kings after Tutankhamen. However, the answer for the query for Hittite kings whose reign overlaps Tutankhamen's is not precise. If Tutankhamen reigned during 1347-1339 BC, the answer to the query is Suppiluliuma I, who reigned during 1380-1340 BC, and Arnuwanda II, who reigned during 1340-1339. If Tutankhamen reigned during 1334-1325 BC or 1336-1327 BC, the answer for the query is Mursili II, who reigned during 1339-1306 BC. It is possible that any one of the answers is true. Even though the answer to this query is not precise, it is useful. Imprecise valid time is an important issue in temporal databases [15, 32, 33]. Relational algebra is extended to capture imprecise valid time in [15]. In [32], probability is associated with an imprecise valid time. However, in most situations, this probability is unknown. For example, the probability that King Tutankhamen's reign began in 1347 BC (or 1334 BC or 1336 BC) is unknown. Although it is possible to assign arbitrary probability when the actual value is unknown, the computational cost for operations involving probability is unnecessarily high. In this chapter, a simpler approach is adopted. Operators and temporal predicates on imprecise intervals are defined. Then, interpretations of temporal predicates on imprecise intervals are extended to handle imprecise valid time in property and representative relations.

First, the representation of precise intervals is extended for imprecise intervals. An interval can be represented by the two end points. However, an end point of an imprecise interval is an unknown point in a given set. For example, there are three possible beginning points and three possible ending points of the interval of King Tutankhamen's reign, as shown earlier. Notice also that, given the possibility that Tutankhamen was throned in 1347 BC, or 1334 BC, it is possible that he was throned at any time during 1347-1334 BC. Thus, the beginning point of Tutankhamen's reign can be any point between 1347-1334 BC. The possible end point is in a set of points between, and including, two given points. This set is called ***a range***, and imprecise intervals can be defined on ranges. In Section 7.2, operators and temporal predicates on ranges are defined. Then, in Section 7.3, operators and temporal predicates, which are an extension of Allen's temporal predicates [2], on imprecise intervals are defined, based on operators and temporal predicates on ranges. Temporal predicates for imprecise intervals can be classified into two types; precise and imprecise predicates. A precise predicate, e.g.

before, *after*, etc., indicates a certain relationship definitely holds for any possible intervals of two imprecise intervals. For example, an imprecise interval whose ending point is between 1347 BC and 1334 BC is definitely before an imprecise interval whose beginning point is between 1330 BC and 1300 BC. An imprecise predicate, e.g. *pbefore* (probably before), indicates a certain relationship probably holds for two imprecise intervals. For example, an imprecise interval whose ending point is between 1347 BC and 1334 BC is probably before an imprecise interval whose beginning point is between 1339 BC and 1300 BC. In Section 7.4, interpretations of temporal predicates for imprecise valid time, based on characteristics of property and representative relations, are studied. The interpretations of precise predicates are similar to those in Section 4.2; however, the interpretations of imprecise predicates must be included.

7.2 Ranges

A *range* is a set of points between, and including, two given points, called an *upper bound* and a *lower bound*. For example, a range R = <1990, 1993> is a set of points between, and including, 1990 and 1993, where the lower bound of R is 1990 and the upper bound is 1993. Let *lower*(R) denote the lower bound of R, and *upper*(R) denote the upper bound of R. For example, *lower*(<1990, 1993>) = 1990 and *upper*(<1990, 1993>) = 1993.

Next, operators on ranges, which are required for the operators on imprecise intervals, are defined. The *min* operator on two ranges finds a range whose lower and upper bounds are the smaller of the lower and the upper bounds of the two given ranges. Similarly, the *max* operator on two ranges finds a range whose lower and upper bounds are the bigger of the lower and the upper bounds of the two given ranges. These operators are formally defined as follows.

Definition 7.1: min and max operators

Let R_1 and R_2 be ranges. The *min* and the *max* operators can be defined as follows:

$$min(R_1, R_2) = < min(lower(R_1), lower(R_2)), min(upper(R_1), upper(R_2)) >.$$

$$max(R_1, R_2) = < max(lower(R_1), lower(R_2)), max(upper(R_1), upper(R_2)) >.$$

The following example shows the operators *min* and *max* on ranges.

Example 7.1: Let R_1 = <1990, 1993> and R_2 = <1987, 1996>. $min(R_1, R_2)$ = <1987, 1993> and $max(R_1, R_2)$ = <1990, 1996>, as shown in Figure 7.1.

```
       R1              <----->
       R2         <----------------->
min(R1, R2)       <----------->
max(R1, R2)            <----------->
                  |-----|-----|-----|
                 1987  1990  1993  1996
```

Figure 7.1: The *min* and *max* operators.

Allen's temporal predicates between intervals are defined on the relationships between end points [2]. The temporal predicates between imprecise intervals can be defined similarly. However, an end point of an imprecise interval is a point in a range, and the relationships between points in ranges can be imprecise. For example, if a point A is in a range <1990, 1993> and a point B is in a range <1992, 1996>, then A probably is before, after, or the same point as B. On the other hand, the relationships between points in ranges can sometimes be precise. For example, if a point A is in a range <1990, 1992> and a point B is in a range <1993, 1996>, A is definitely before B. It is necessary to capture both situations. In Definition 7.2, the predicates $<_r$, $>_r$, and $=_r$ indicate "definitely" before, after, and equal, while the predicates $p{<_r}$, $p{>_r}$, and $p{=_r}$ indicate "probably" before, after, and equal. The former type of temporal predicate is called *precise predicate*, and the latter, *imprecise predicate*.

Definition 7.2: $<_r$, $>_r$, $=_r$, $p{<_r}$, $p{>_r}$ ***and*** $p{=_r}$

Let R_1 and R_2 be ranges. Precise and imprecise predicates for ranges are defined in Table 7.1 below.

Table 7.1: Definitions of precise and imprecise predicates on ranges.

Precise Predicate	***Definition***	***Imprecise Predicate***	***Definition***
$<_r(R_1, R_2)$	$upper(R_1) < lower(R_2)$.	$p{<_r}(R_1, R_2)$	$lower(R_1) < upper(R_2)$.
$>_r(R_1, R_2)$	$lower(R_1) > upper(R_2)$.	$p{>_r}(R_1, R_2)$	$upper(R_1) > lower(R_2)$.
$=_r(R_1, R_2)$	$lower(R_1) = upper(R_1) = lower(R_2) = upper(R_2)$.	$p{=_r}(R_1, R_2)$	$upper(R_1) \geq lower(R_2)$ or $lower(R_1) \leq upper(R_2)$.

From definitions of predicates in Table 7.1, it is clear that the computational cost of these predicates are lower than that of the predicates in [32], which require the summation of products, e.g., $\Sigma_{I<j}$ P(I)·P(j). Furthermore, in Table 7.1 the precise predicates are distinguished from the imprecise predicates, while in [32] they are distinguished by the probability indicated in the predicates.

There are a few observations that need to be indicated here. First, if $<_r(R_1, R_2)$ is true, $p<_r(R_1, R_2)$ is true, and similarly for $>_r$ and $=_r$. Second, if $<_r(R_1, R_2)$ is true, $>_r(R_1, R_2)$ and $=_r(R_1, R_2)$, are false, and similarly for $>_r$ and $=_r$. Third, if $<_r(R_1, R_2)$ is false but $p<_r(R_1, R_2)$ is true, either $>_r(R_1, R_2)$ or $=_r(R_1, R_2)$ are true, and similarly for $>_r$ and $=_r$. Finally, $=_r(R_1, R_2)$ is true only when both ranges contain only one point, i.e. they are precise, and the two points are equal.

Next, imprecise intervals are defined on ranges. Based on the operators and the temporal predicates on ranges, operators and temporal predicates on imprecise intervals are defined.

7.3 Imprecise Intervals

A precise interval can be defined by its end points, and an imprecise interval can be defined similarly. While an end point of a precise interval is a point, an end point of an imprecise interval is an unknown point in a range. Thus, an imprecise interval can be defined, based on ranges, as follows:

Definition 7.3: Imprecise Interval

Let B and E be ranges. An ***imprecise interval*** I, denoted by $[B, E]$, is an interval of which the beginning point is in B and the ending point is in E, where $lower(B)<lower(E)$ and $upper(B)<upper(E)$. Let ***begin***(I) denote the range of the beginning point of I, i.e. B, and ***end***(I) denote the range of the ending point of I, i.e. E.

The following example shows an imprecise interval.

Example 7.2: The interval of Tutankhamen's reign is the imprecise interval *TI* = [<1347 BC, 1334 BC>, <1339 BC, 1325 BC >]. *begin*(*TI*) is the range <1347 BC, 1334 BC> and *end*(*TI*) is the range <1339 BC, 1325 BC >. Notice that *lower*(<1347 BC, 1334 BC>) < *lower*(<1339 BC, 1325 BC >) and *upper*(<1347 BC, 1334 BC>) < *upper*(<1339 BC, 1325 BC >).

In Definition 7.3, there is a restriction on the relationship between the ranges of the beginning and the ending points of an interval. First, the range of the ending point cannot contain any point before the lower bound of the range of the beginning point, as shown in the range *E* in Figure 7.2, because the ending point cannot come before the beginning point. That is, *lower*(*B*) < *lower*(*E*), where *B* is the range of the beginning point and *E* is the range of the ending point. Second, the range of the beginning point cannot contain any point after the upper bound of the range of the ending point, as shown in the range *B* in Figure 7.2, because the beginning point cannot come after the ending point. That is, *upper*(*B*) < *upper*(*E*).

```
B            <------xxxxx>
E      <xxxxx------>
```

Figure 7.2: The restriction on the relationship between the ranges of the beginning and the ending points of an interval. x denotes the part of the range which violates the restriction in Definition 7.3.

In [32], it is assumed that the ranges of the beginning and the ending points must be disjoint, which is more restricted than Definition 7.3. However, the assumption of disjoint ranges of beginning and ending points is too limited. Consider an imprecise interval with overlapping ranges of the beginning and ending points shown in Figure 7.3. From the definition of imprecise intervals, the beginning point of this imprecise interval can be any point in the range *B*. Thus, it is possible that *lower*(*B*) is the beginning point of the interval. Then, the ending point of the interval can be any point after *lower*(*B*). Thus, there is no reason to prevent overlapping ranges of the beginning and the ending points.

```
<-----B----->
       <----E----->
```

Figure 7.3: An imprecise interval with overlapping ranges of the beginning and the ending points.

Temporal predicates and operators on imprecise intervals are necessary for queries on temporal databases with imprecise valid time. In the next section, the temporal predicates on imprecise intervals are examined.

7.3.1 Temporal predicates on Imprecise Intervals

Allen's temporal predicates on intervals are defined on the temporal predicates on the end points [2]. For example, an interval *A* is before an interval *B* if the ending of *A* is before the beginning of *B*. Similarly, the temporal predicates on imprecise intervals can be defined on the temporal predicates on ranges. For example, an imprecise interval *C* is definitely before an imprecise interval *D* if the range of the ending point of *C* is definitely before the range of the beginning point of *D*. However, the end points of imprecise intervals are ranges, and the temporal predicates on ranges can be imprecise as shown in Section 7.2. Thus, the temporal predicates on imprecise intervals can be imprecise. For example, the imprecise interval [<1990, 1993>, <1995, 1997>] probably is after or met by or overlaps the imprecise interval [<1980, 1982>, <1990, 1993>]. Based on the temporal predicate on ranges, the definitions of precise predicates, such as *BEFORE*, *AFTER*, *MEET*, etc., can be easily extended to capture imprecise relationships. Precise predicates, e.g., *BEFORE*, *AFTER*, *MEET*, etc., and imprecise predicates, e.g., *PBEFORE*, *PAFTER*, *PMEET*, etc., on imprecise intervals are defined as follows.

Definition 7.4: Temporal predicates between imprecise intervals

Let I_1 and I_2 be imprecise intervals. Precise temporal predicates and imprecise temporal predicates on imprecise intervals are defined in Table 7.2 and Table 7.3.

Table 7.2: Definitions of precise temporal predicates on imprecise intervals.

Precise Predicate on (I_1, I_2)	***Definition***
BEFORE	$<_r(end(I_1), begin(I_2))$
AFTER	$>_r(begin(I_1), end(I_2))$
MEET	$=_r(begin(I_1), end(I_2))$
MET_BY	$=_r(end(I_1), begin(I_2))$
OVERLAP	$<_r(begin(I_1), begin(I_2)) \wedge <_r(begin(I_2), end(I_1)) \wedge <_r(end(I_1), end(I_2))$
OVERLAPPED_BY	$>_r(begin(I_1), begin(I_2)) \wedge >_r(end(I_2), begin(I_1)) \wedge >_r(end(I_1), end(I_2))$
START	$=_r(begin(I_1), begin(I_2)) \wedge <_r(end(I_1), end(I_2))$
STARTED_BY	$=_r(begin(I_1), begin(I_2)) \wedge >_r(end(I_1), end(I_2))$
FINISH	$=_r(end(I_1), end(I_2)) \wedge >_r(begin(I_1), begin(I_2))$
FINISHED_BY	$=_r(end(I_1), end(I_2)) \wedge <_r(begin(I_1), begin(I_2))$
DURING	$>_r(begin(I_1), begin(I_2)) \wedge <_r(end(I_1), end(I_2))$
CONTAIN	$<_r(begin(I_1), begin(I_2)) \wedge >_r(end(I_1), end(I_2))$
EQUAL	$=_r(begin(I_1), begin(I_2)) \wedge =_r(end(I_1), end(I_2))$

Table 7.3: Definitions of precise and imprecise predicates on imprecise intervals.

Imprecise Predicate on (I_1, I_2)	***Definition***
PBEFORE	$p{<_r}(end(I_1), begin(I_2))$
PAFTER	$p{>_r}(begin(I_1), end(I_2))$
PMEET	$p{=_r}(begin(I_1), end(I_2))$
PMET_BY	$p{=_r}(end(I_1), begin(I_2))$
POVERLAP	$p{<_r}(begin(I_1), begin(I_2)) \wedge p{<_r}(begin(I_2), end(I_1)) \wedge p{<_r}(end(I_1), end(I_2))$
POVERLAPPED_BY	$p{>_r}(begin(I_1), begin(I_2)) \wedge p{>_r}(end(I_2), begin(I_1)) \wedge p{>_r}(end(I_1), end(I_2))$
PSTART	$p{=_r}(begin(I_1), begin(I_2)) \wedge p{<_r}(end(I_1), end(I_2))$
PSTARTED_BY	$p{=_r}(begin(I_1), begin(I_2)) \wedge p{>_r}(end(I_1), end(I_2))$
PFINISH	$p{=_r}(end(I_1), end(I_2)) \wedge p{>_r}(begin(I_1), begin(I_2))$
PFINISHED_BY	$p{=_r}(end(I_1), end(I_2)) \wedge p{<_r}(begin(I_1), begin(I_2))$
PDURING	$p{>_r}(begin(I_1), begin(I_2)) \wedge p{<_r}(end(I_1), end(I_2))$
PCONTAIN	$p{<_r}(begin(I_1), begin(I_2)) \wedge p{>_r}(end(I_1), end(I_2))$
PEQUAL	$p{=_r}(begin(I_1), begin(I_2)) \wedge p{=_r}(end(I_1), end(I_2))$

Similar to the temporal predicates on end points in ranges, *PBEFORE*(I_1, I_2) is true and other precise predicates, such as *AFTER*, *MEET*, etc., between I_1 and I_2 are false if *BEFORE*(I_1, I_2) is true. Furthermore, if *PBEFORE*(I_1, I_2) is true but *BEFORE*(I_1, I_2) is false, at least one of other imprecise predicates, such as, *PAFTER*, *PMEET*, etc., between I_1 and I_2 is true.

Based on predicates in Definition 7.4, we define four more temporal predicates, which are frequently used in queries and other definitions. They are shown in Table 7.4 and Table 7.5.

Table 7.4: Additional precise temporal predicates on imprecise intervals.

Precise Predicate on (I_1, I_2)	***Definition***
COMMON	*OVERLAP*$(I_1, I_2) \vee$ *OVERLAPPED_BY*$(I_1, I_2) \vee$ *START*$(I_1, I_2) \vee$ *STARTED_BY*$(I_1, I_2) \vee$ *FINISH*$(I_1, I_2) \vee$ *FINISHED_BY*$(I_1, I_2) \vee$ *DURING*$(I_1, I_2) \vee$ *CONTAIN*$(I_1, I_2) \vee$ *EQUAL*(I_1, I_2)
DISJOINT	*BEFORE*$(I_1, I_2) \vee$ *AFTER*(I_1, I_2) or *MEET*$(I_1, I_2) \vee$ *MET_BY*(I_1, I_2).

Table 7.5: Additional imprecise temporal predicates on imprecise intervals.

Imprecise Predicate on (I_1, I_2)	***Definition***
PCOMMON	*POVERLAP*$(I_1, I_2) \vee$ *POVERLAPPED_BY*$(I_1, I_2) \vee$ *PSTART*$(I_1, I_2) \vee$ *PSTARTED_BY*$(I_1, I_2) \vee$ *PFINISH*$(I_1, I_2) \vee$ *PFINISHED_BY*$(I_1, I_2) \vee$ *PDURING*(I_1, I_2) or *PCONTAIN*$(I_1, I_2) \vee$ *PEQUAL*(I_1, I_2)
PDISJOINT	*PBEFORE*$(I_1, I_2) \vee$ *PAFTER*$(I_1, I_2) \vee$ *PMEET*$(I_1, I_2) \vee$ *PMET_BY*(I_1, I_2).

These temporal predicates on imprecise intervals can be used in queries on temporal databases with imprecise valid time. For example, *after* can be used in the query for Egyptian kings who reigned "definitely" after Tutankhamen, and *pcommon* can be used in the query for Hittite kings whose reign "probably" overlaps Tutankhamen's reign. Next, operators on imprecise intervals are defined.

7.3.2 Operators on Imprecise Intervals

Operators *union* ($\cup$), *intersection* ($\cap$), and *difference* (-) for precise intervals must be extended to apply for imprecise intervals. Figure 7.4 shows operators $\cup$, $\cap$, and - on imprecise intervals in a simple situation. $I_1 \cup I_2$ = [*min*(*begin*(I_1), *begin*(I_2)), *max*(*end*(I_1), *end*(I_2))], $I_1 \cap I_2$ = [*max*(*begin*(I_1), *begin*(I_2)), *min*(*end*(I_1), *end*(I_2))], and $I_1 - I_2$ = [*begin*(I_1), *begin*(I_2)] or [(*end*(I_2), *end*(I_1)].

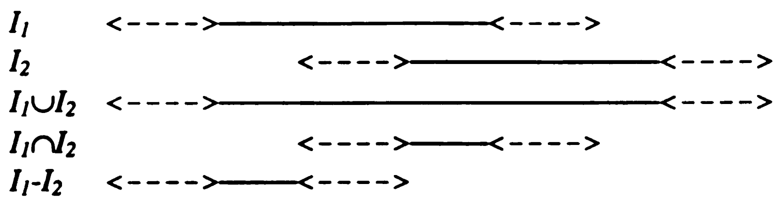

Figure 7.4: Union, Intersection, and difference of two imprecise intervals in a simple situation.

However, complication may arise when the imprecise intervals created by these expressions violate the restriction in Definition 7.3. For example, in Figure 7.5, $I_1 \cap I_2$= [begin(I_2), end(I_1)], but *lower*(begin(I_2)) > *lower*(end(I_1)) and *upper*(begin(I_2)) > *upper*(end(I_1)). Furthermore, in this situation $I_1 \cup I_2$ cannot be represented by one imprecise interval, and thus is undefined. More complex operations must be added to accommodate these details, and the operators are defined in Definition 7.5.

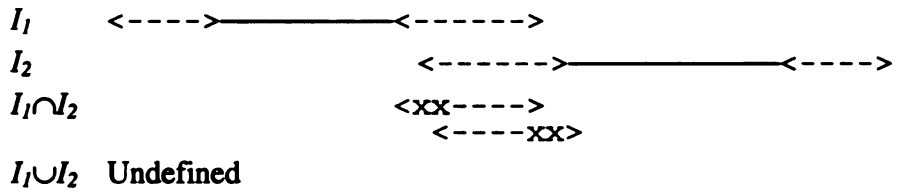

Figure 7.5: Union and Intersection of two imprecise intervals in a complicated situation.

Definition 7.5: Union, intersection, and difference on imprecise Intervals

Let $I_1 = [B_1, E_1]$ and $I_2 = [B_2, E_2]$ be imprecise intervals, $B = max(B_1, B_2)$ and $E = min(E_1, E_2)$. Operators on imprecise intervals can be defined in Table 7.6.

Table 7.6: Operators on imprecise intervals.

Operator	*Condition*	*Result*
$I_1 \cup I_2$	$COMMON(I_1, I_2)$	$[min(B_1, B_2), max(E_1, E_2)]$
$I_1 \cup I_2$	NOT $COMMON(I_1, I_2)$	undefined
$I_1 \cap I_2$	$PCOMMON(I_1, I_2)$	$[<lower(B), min(upper(B), upper(E))>, <max(lower(E), lower(B)), upper(E)>]$
$I_1 \cap I_2$	NOT $PCOMMON(I_1, I_2)$	$\varnothing$
$I_1 - I_2$	$lower(B_1) \leq upper(B_2)$ and $upper(E_1) > lower(E_2)$	$[<lower(B_1), min(upper(B_1), upper(B_2))>, <max(lower(B_1, B_2)), upper(B_2)>]$
$I_1 - I_2$	$lower(B_1) > upper(B_2)$ and $upper(E_1) \leq lower(E_2)$	$[<lower(E_2), min(upper(E_1), upper(E_2))>, <max(lower(E_1, E_2)), upper(B_1)>]$
$I_1 - I_2$	$lower(B_1) \leq upper(B_2)$ and $upper(E_1) \leq lower(E_2)$	$[<lower(B_1), min(upper(B_1), upper(B_2))>, <max(lower(B_1, B_2)), upper(B_2)>]$ and $[<lower(E_2), min(upper(E_1), upper(E_2))>, <max(lower(E_1, E_2)), upper(B_1)>]$

These operators on imprecise intervals can be used to calculate the valid time in temporal databases with imprecise valid time.

7.4 Interpretations of Temporal Predicates for Imprecise Valid Time in Property and Representative Relations

In Section 4.2, different interpretations of temporal predicates for property and representative relations with precise valid time are examined. Similarly, different interpretations of temporal predicates for imprecise intervals are applied when the intervals are the valid times in property and representative relations. For example, consider the interpretations of the predicate *pduring* in the following two queries:

> Q'_1: *Find all Hittite kings who probably reigned during King Tutankhamen's reign.*
>
> Q'_2: *Find all Hittite kings who reigned longer than 5 years and probably reigned during King Tutankhamen's reign.*

For Q'_1, the valid time when a Hittite king reigned, t_1, and the valid time when King Tutankhamen reigned, t_2, are obtained. Then, the interpretation that some sub-interval of t_1 is in t_2 is applied to the predicate *pduring* because the reign of a king is a property relation and the valid time is decomposable. For Q'_2, the valid time of the length of a Hittite king reign, t_3, and the valid time when King Tutankhamen reigned, t_4, are obtained. In this query, the interpretation that the whole interval of t_3 is in t_4 is applied to the predicate *pduring* because the length of a reign is a representative relation and t_3 is not decomposable. Similar to the interpretations of temporal predicates in Section 4.2, the first interpretation is called $pduring_{\exists}$, and the second, $pduring_{\forall}$. These two interpretations can be defined in terms of temporal predicates for imprecise intervals, which is denoted by italic capital letters. The followings are the two interpretations.

Let A and B be sets of time-varying attributes, T_A, and T_B be imprecise valid time, and $R_A(A\ T_A)$ and $R_B(B\ T_B)$ be two temporal relations.

$$pduring_{\exists}(T_A, T_B) \;=\; \exists\, t_A\; t_A \subseteq T_A\; PDURING(t_A, T_B)$$

$$= PBEFORE(T_A,T_B) \vee PMEET(T_A,T_B) \vee POVERLAP(T_A,T_B) \vee$$

$$PFINISHED\_BY(T_A,T_B) \vee PCONTAIN(T_A,T_B).$$

$$pduring_{\forall}(T_A,T_B) = PDURING(T_A, T_B).$$

As mentioned in Section 4.2, the predicates *meet, pmeet, start, pstart, finish, pfinish, overlap, poverlap,* and their inverses, are independent of types of temporal relations, and each of them has only one interpretation. The interpretations of predicates *before*, *after*, *during*, *contain* and *equal* are similar to those in Section 4.2, and the interpretations of predicates *pbefore*, *pafter*, *pduring*, *pcontain* and *pequal* are presented in Table 7.7.

7.5 Conclusions

Valid time in temporal databases can be imprecise. Though the result of a query on imprecise valid time can be imprecise, it is still useful. In this chapter, a representation of imprecise intervals is presented. This representation imposes less restriction than the representation proposed in [32]. Based on this representation, the operators *union*, *intersection*, and *difference* on imprecise intervals are defined. The relationships between intervals, presented in Chapter 4, are extended to capture both precise and imprecise relationships between imprecise intervals. The representation of imprecise intervals, the operators and relationships on imprecise intervals can be incorporated in temporal databases to allow queries on imprecise valid time. Finally, interpretations of temporal predicates for imprecise valid time in different types of temporal relations are defined on the temporal predicates for imprecise intervals.

Table 7.7: Temporal predicates for imprecise valid time.

Let A and B be sets of time-varying attributes, T_A and T_B be valid time and $R_A(A\ T_A)$ and $R_B(B\ T_B)$ be temporal relations. The column ***Predicate*** contains names of temporal relationship between R_A and R_B. The columns $\boldsymbol{R_A}$ and $\boldsymbol{R_B}$ are types of R_A and R_B, where P and R denote property and representative relations respectively. The column ***Interpretation*** contains the interpretation of the corresponding predicate and the column ***Definition*** contains the definitions of interpretation of the predicates.

Predicate	$\boldsymbol{R_A}$	$\boldsymbol{R_B}$	***Interpretation***	***Definition***
$pbefore(T_A,T_B)$	R	P/R	$pbefore_{\forall}(T_A,T_B)$	$PBEFORE(T_A, T_B)$
$pbefore(T_A,T_B)$	P	P/R	$pbefore_{\exists}(T_A,T_B)$	$PBEFORE(T_A, T_B) \vee PMEET(T_A, T_B) \vee POVERLAP(T_A, T_B) \vee PCONTAIN(T_A, T_B)$
$pafter(T_A,T_B)$	R	P/R	$pafter_{\forall}(T_A,T_B)$	$PAFTER(T_A, T_B)$
$pafter(T_A,T_B)$	P	P/R	$pafter_{\exists}(T_A,T_B)$	$PAFTER(T_A, T_B) \vee PMET\_BY(T_A, T_B) \vee POVERLAPPED\_BY(T_A, T_B) \vee PDURING(T_A, T_B)$
$pduring(T_A,T_B)$	R	P/R	$pduring_{\forall}(T_A,T_B)$	$PDURING(T_A, T_B)$
$pduring(T_A,T_B)$	P	P/R	$pduring_{\exists}(T_A,T_B)$	$POVERLAP(T_A, T_B) \vee POVERLAPPED\_BY(T_A, T_B) \vee PCONTAIN(T_A, T_B) \vee PDURING(T_A, T_B) \vee PSTART(T_A, T_B) \vee PSTARTED\_BY(T_A, T_B) \vee PFINISH(T_A, T_B) \vee PFINISHED\_BY(T_A, T_B) \vee PEQUAL(T_A, T_B)$
$pcontain(T_A,T_B)$	P/R	R	$pcontain_{\forall}(T_A,T_B)$	$PCONTAIN(T_A, T_B)$
$pcontain(T_A,T_B)$	P/R	P	$pcontain_{\exists}(T_A,T_B)$	$POVERLAP(T_A, T_B) \vee POVERLAPPED\_BY(T_A, T_B) \vee PCONTAIN(T_A, T_B) \vee PDURING(T_A, T_B) \vee PSTART(T_A, T_B) \vee PSTARTED\_BY(T_A, T_B) \vee PFINISH(T_A, T_B) \vee PFINISHED\_BY(T_A, T_B) \vee PEQUAL(T_A, T_B)$
$pequal(T_A, T_B)$	R	R	$equal_{\forall\forall}(T_A, T_B)$	$PEQUAL(T_A, T_B)$
$pequal(T_A, T_B)$	R	P	$equal_{\forall\exists}(T_A, T_B)$	$PDURING(T_A, T_B) \vee PSTART(T_A, T_B) \vee PFINISH(T_A, T_B) \vee PEQUAL(T_A, T_B)$
$pequal(T_A, T_B)$	P	R	$equal_{\exists\forall}(T_A, T_B)$	$PCONTAIN(T_A, T_B) \vee PSTARTED\_BY(T_A, T_B) \vee PFINISHED\_BY(T_A, T_B) \vee PEQUAL(T_A, T_B)$
$pequal(T_A, T_B)$	P	P	$equal_{\exists\exists}(T_A, T_B)$	$POVERLAP(T_A, T_B) \vee POVERLAPPED\_BY(T_A, T_B) \vee PCONTAIN(T_A, T_B) \vee PDURING(T_A, T_B) \vee PSTART(T_A, T_B) \vee PSTARTED\_BY(T_A, T_B) \vee PFINISH(T_A, T_B) \vee PFINISHED\_BY(T_A, T_B) \vee PEQUAL(T_A, T_B)$

Chapter 8

CONCLUSIONS

8.1 Discussions

Both homogeneous and non-homogeneous data have been addressed in many researches in AI [3, 58, 73]. These are focused on automatic planning by using temporal facts, and database issues such as design principles are not addressed to the studies. On the other hand, non-homogeneous data are not considered in existing temporal databases, and design principles for temporal databases containing both homogeneous and non-homogeneous data have not been addressed. The goal in this dissertation is to define a data model for both homogeneous and non-homogeneous data, and to provide design principle for temporal databases that contain both homogeneous and non-homogeneous data.

In Chapter 3, two types of temporal relations–property relations and representative relations, which represent homogeneous data and non-homogeneous data are defined. In Chapter 4, a temporal relational data model which supports both homogeneous and non-homogeneous data is presented. Relational operators on temporal relations depend on the characteristics of temporal relations. The valid time in temporal relations are manipulated according to the characteristics of the temporal data represented by the relations. Furthermore, temporal predicates for time intervals, defined in [2], are extended to apply on valid time of property and representative relations. The difference between the extension of temporal predicates and the original temporal predicates is a result of the decomposable valid time for property relations and non-decomposable valid time for representative relations. As a result, users are relieved from the task of manipulating the valid time. This also prevents errors in the manipulation of the valid time.

P-inconsistency can occur in temporal databases with homogeneous and non-homogeneous data. In Chapter 5, PCNF, which avoids P-inconsistency, is defined. PCNF is based on P-dependencies, functional dependencies, and types of time-varying attributes. A normalization algorithm for PCNF is presented. However, this algorithm may not give the minimum number of normalized relations. We prove that the problem of finding the minimum number of relations in PCNF is NP-complete. In the process, we also prove that finding equivalent sets of attributes is an NP-complete problem.

For traditional normal forms, such as 3NF, the numbers of normalized relations created from different minimal covers are the same. However, for PCNF, different minimal covers can give different numbers of normalized relations. We present a heuristic for reducing the number of normalized relations for PCNF.

Based on this temporal relational data model, a temporal deductive database, called TDatalog, is defined in Chapter 6. The manipulation of time in a TDatalog rule is implied from types of temporal relations in the rules. Since the manipulation of valid time is implicit in TDatalog rules, Datalog rules are also applicable in TDatalog, which is an advantage of TDatalog.

Finally, in Chapter 7, the temporal relational data model presented in Chapter 4 is extended to support imprecise valid time in property and representative relations. Imprecise valid time in temporal databases is addressed in [15, 32, 33]. However, imprecise valid time in non-homogeneous data is not considered in these works.

8.2 Future Work

There are two issues that need to be addressed in the future work. First, the heuristic for PCNF normalization can be improved. This heuristic is based on the substitution of a set of attributes in a dependency by an equivalent set of attributes. However, it is not possible to find all equivalent sets of attributes. If more can be found, a better decomposition can be generated. Thus, the heuristic for finding equivalent sets of attributes should be further examined.

Second, design principles for temporal databases based on other characteristics of temporal data need to be studied. In this dissertation, a type of inconsistency, based on homogeneous and non-homogeneous data, is identified. It is necessary to examine if other type of inconsistencies, based on other characteristics of temporal data, in temporal databases can be identified.

Appendices

Appendix A

Appendix A

PROOF OF THEOREM 5.1.

***Theorem** 5.1:* I1-I8 are complete.

Proof:

Let $r(R)$ be a relation, G be a set of dependencies on R, and W, X, Y, and Z be sets of attributes in R. We show that there exists a relation r that satisfies G^+ and no other dependencies which are not in G^+.

Let $r(R)$ be a temporal relation with four tuples τ_1, τ_2, τ_3, and τ_4.

$\tau_1(T) = \tau_2(T)$, $\tau_3(T) \cap \tau_4(T) \neq \varnothing$, $\tau_1(T) \cap \tau_3(T) = \varnothing$, $\tau_1(T) \cap \tau_4(T) = \varnothing$,

$\tau_1(A) = \tau_2(A)$ if A is in X_F^+, $\tau_1(A) \neq \tau_2(A)$ if A is not in X_F^+,

$\tau_3(A) = \tau_4(A)$ if A is in X_P^+, $\tau_3(A) \neq \tau_4(A)$ if A is not in X_P^+.

First, we show that r satisfies G^+. For any functional dependency $W\,T \rightarrow Z$ in G^+, we show that r satisfies $W\,T \rightarrow Z$.

Case 1: $X_P^+ \not\subseteq W \subseteq X_F^+$. From I6-I7 and $W \subseteq X_F^+$, $X\,T \rightarrow W$ is in G^+. From $X\,T \rightarrow W$ and $W\,T \rightarrow Z$, through I8, $X\,T \rightarrow Z$ is also in G^+. Then, $Z \subseteq X_F^+$. Thus, r satisfies $W\,T \rightarrow Z$.

Case 2: $W \not\subseteq X_F^+$. Since $W \not\subseteq X_F^+$, r satisfies $W\,T \rightarrow Z$.

Case 3: $W \subseteq X_P^+$. From I6-I7 and $W \subseteq X_P^+$, $X\,T \Rightarrow W$ is in G^+. From $X\,T \Rightarrow W$ and $W\,T \rightarrow Z$, through I1 and I8, $X\,T \rightarrow Z$ is in G^+. Then, $Z \subseteq X_F^+$. Thus, r satisfies $W\,T \rightarrow Z$.

For any P-dependency $W\,T \Rightarrow Z$ in G^+, we show that r satisfies $W\,T \Rightarrow Z$.

Case 1: $W \subseteq X_P^+$. From I3-I4 and $W \subseteq X_P^+$, $X\,T \Rightarrow W$ is in G^+. From $X\,T \Rightarrow W$ and $W\,T \Rightarrow Z$, through I5, $X\,T \Rightarrow Z$ is in G^+. Then, $Z \subseteq X_P^+$. Thus, r satisfies $W\,T \Rightarrow Z$.

Case 2: X and Z are homogeneous, and $X_P^+ \not\subseteq W \subseteq X_F^+$. From I3-I4 and $W \subseteq X_F^+$, $X\,T \rightarrow W$ is in G^+. From $X\,T \rightarrow W$ and $W\,T \Rightarrow Z$, $X\,T \Rightarrow Z$ is in G^+ by I1, I2 and I5. Then, $Z \subseteq X_P^+$. Thus, r satisfies $W\,T \Rightarrow Z$.

Case 3: X or Z are non-homogeneous, and $X_P^+ \not\subseteq W \subseteq X_F^+$. From I3-I4 and $W \subseteq X_F^+$, $X\,T \rightarrow W$ is in G^+. From $X\,T \rightarrow W$ and $W\,T \Rightarrow Z$, through I1 and I8, $X\,T \rightarrow Z$ is in G^+. Then, $Z \subseteq X_F^+$. Thus, r satisfies $W\,T \Rightarrow Z$.

Case 4: $W \not\subseteq X_F^+$. Because $W \not\subseteq X_F^+$, r satisfies $W\,T \Rightarrow Z$.

Therefore, r satisfies any dependencies in G^+.

Second, we show that r does not satisfy any dependency which is not in G^+. Let $X\,T \rightarrow Y$ and $X\,T \Rightarrow Z$ be dependencies which are not in G^+. Since $X\,T \rightarrow Y$ is not in G^+, $Y\text{-}X_F^+ \neq \emptyset$. $\tau_1(Y\text{-}X_F^+) \neq \tau_2(Y\text{-}X_F^+)$ and $\tau_3(Y\text{-}X_F^+) \neq \tau_4(Y\text{-}X_F^+)$. Then, r does not satisfy $X\,T \rightarrow Y$. Similarly, since $X\,T \Rightarrow Z$ is not in G^+, $Z\text{-}X_P^+ \neq \emptyset$. $\tau_3(Z\text{-}X_P^+) \neq \tau_4(Z\text{-}X_P^+)$. Then, r does not satisfy $X\,T \Rightarrow Z$.

Then, r satisfies G^+, and does not satisfy $X\,T \rightarrow Y$ or $X\,T \Rightarrow Z$ which are not derived from G. Thus, I1-I8 are complete. $\square$

Appendix B

Appendix B

PROOF OF LEMMA 5.3

Lemma 5.3: EQ is NP-complete.

Proof:

To prove that EQ is NP-complete, we show that a special case of EQ is NP-complete. We consider a special case of EQ such that G contains only P-dependencies, and for all dependencies $X\ T \Rightarrow A$ in G, X is a single attribute.

First, we show that the special case of EQ is in NP. If we non-deterministically choose $Y \subseteq H$, it takes polynomial time to determine whether $X\ T \Leftrightarrow Y\ T$ under G. Thus, EQ is in NP.

Next, we show a polynomial-time reduction of the hitting set problem (HS), which is NP-complete, to the special case of EQ. Given the input C_i, for i=1, 2,…,n, of HS, we can create the input of G, H and X for EQ such that C_i is a determinative of an attribute A_i in X in G, for i=1, 2, …, n, and all equivalent sets of attributes of X is smaller than an integer K. R, G, H, and X can be created as follows:

1. Let $G' = \varnothing$; $X = \varnothing$; $H = C_1 \cup C_2 \cup \ldots \cup C_n$; $R = H \cup \{T\}$.

```
2.  For $i = 1, 2, ..., n$:
    2.1.  create a new attribute $A_i$ such that $A_i$ is not in $H$; $R=R\cup\{A_i\}$.
    2.2.  $G'=G'\cup\{B\ T\Rightarrow A_i|\ B$ is an attribute in $C_i\}$.
    2.3.  $X=X\cup A_i$.
3.  $G$ = minimal cover of $G'$.
```

It is clear that the algorithm above has polynomial-time complexity. The set of dependencies G' is created in Step 2 by first creating an attribute A_i, which is not in H, and create G' such that $C_i = \text{DP}(A_i, X, G')\cap H$. Since A_i is not in H, A_i is not in $\text{DP}(A_i, X, G')\cap H$. G' created in Step 2 may not be a minimal cover. Thus, G, the minimal cover of G', is created in polynomial time, in Step 3. Therefore, $C_i = \text{DP}(A, X, G)\cap H$, where A is an attribute in X.

The output Y of EQ is a set of attributes, in H, which is equivalent to X. The output S' for HS is the set of attributes Y, derived from EQ. Since $|Y|\leq K$, then $|S'|\leq K$. Therefore, there is a polynomial-time reduction of HS to a special case of EQ.

Next, we show that S' is the output of HS if and only if Y is the output of EQ. From the reduction, $C_i = \text{DP}(A_i, X, G)\cap H$, $|S'|\leq K$ and S' contains at least one attribute from C_i, for i=1, 2,..., n. Then, S' is the hitting set of C_i, for i=1, 2,..., n. From Lemma 5.2, $X\ T \Leftrightarrow Y\ T$ under G if and only if $Y=\cup_{A\in X} Y_A$ and $Y_A\in \text{DP}(A_i, X, G)$. $C_i = \text{DP}(A_i, X, G)\cap H$. Thus, $X\ T \Leftrightarrow Y\ T$ under G if and only if Y is a hitting set of C_i, for i=1, 2,..., n. S' is Y such that $|Y|\leq K$. Therefore, S' is the hitting set of C_i, for i=1, 2,..., n, such that $|S'|\leq K$ if and only if $Y\subseteq H$ and $X\ T \Leftrightarrow Y\ T$ under G. Thus, the special case of EQ is NP-complete. Since a special case of EQ is NP-complete, EQ is NP-complete. □

Appendix C

Appendix C

PROOF OF LEMMA 5.4

Lemma 5.4: Let R be a relation scheme, G be a minimal cover of a set of dependencies on R, X and Y be sets of attributes in R, A and B be attributes in R, and $X\ T \Rightarrow A$ (or $X\ T \rightarrow A$) be in G. G - $\{X\ T \Rightarrow A$ (or $X\ T \rightarrow A)\} \cup \{Y\ T \Rightarrow B$ (or $Y\ T \rightarrow B)\}$ is a minimal cover of G if and only if there are Y and B such that $Y\ T \Rightarrow B$ (or $Y\ T \rightarrow B$) is reduced, $X\ T \Leftrightarrow Y\ T$ (or $X\ T \leftrightarrow Y\ T$) and $X\ A\ T \Leftrightarrow Y\ B\ T$ (or $X\ A\ T \leftrightarrow Y\ B\ T$) can be derived from G, and $X\ T \Rightarrow Y$ (or $X\ T \rightarrow Y\ T$), and $Y\ B\ T \Rightarrow A$ (or $Y\ B\ T \rightarrow A$) can be derived from G-$\{X\ T \Rightarrow A$ (or $X\ T \rightarrow A$, respectively)$\}$.

Proof:

First, prove the "if" part. Let Y and B be a set of attributes and an attribute such that $X{\neq}Y$ or $A{\neq}B$, $Y\ T \Rightarrow X$ and $X\ A\ T \Rightarrow B$ can be derived from G, and $X\ T \Rightarrow Y$ or $Y\ B\ T \Rightarrow A$ can be derived from G-$\{X\ T \Rightarrow A\}$. Let G'=G-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$. Since $Y\ T \Rightarrow B$ is in G' and $X\ T \Rightarrow Y$ and $Y\ B\ T \Rightarrow A$ can be derived from G - $\{X\ T \Rightarrow A\}$, $X\ T \Rightarrow A$ can be derived from G'. Since $X\ T \Rightarrow A$ is in G and $Y\ T \Rightarrow X$ and $X\ A\ T \Rightarrow B$ can be derived from G, $Y\ T \Rightarrow B$ can be derived from G. Thus, $G'{\equiv}G$. If $Y\ T \Rightarrow B$ is redundant in G', $G{\equiv}G$-$\{X\ T \Rightarrow A\}$, which contradicts to the fact that G is a minimal cover. Therefore, $Y\ T$

$\Rightarrow B$ is non-redundant in G'. Since $Y\ T \Rightarrow B$ is reduced and non-redundant, G' is a minimal cover of G. Similarly, we can prove that G - $\{X\ T \rightarrow A\} \cup \{Y\ T \rightarrow B\}$ is a minimal cover of G if there are Y and B such that $X\ T \leftrightarrow Y\ T$ and $X\ A\ T \leftrightarrow Y\ B\ T$ can be derived from G, and $X\ T \rightarrow Y\ T$ and $Y\ B\ T \rightarrow A$ can be derived from G-$\{X\ T \rightarrow A\}$.

Second, we prove the "only-if" part. Let $X\ T \Rightarrow A$ be a dependency in G, there are Y and B such that $X \neq Y$ or $A \neq B$, and G-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$ is a minimal cover of G. Thus, $Y\ T \Rightarrow B$ is reduced. To prove that $Y\ T \Rightarrow X$ and $X\ A\ T \Rightarrow B$ can be derived from G, we first show that $X\ T \Rightarrow A$ is used in the derivation of $Y\ T \Rightarrow B$ from G. Assume $X\ T \Rightarrow A$ is not used in the derivation of $Y\ T \Rightarrow B$ from G. That is, $Y\ T \Rightarrow B$ can be derived from G-$\{X\ T \Rightarrow A\}$. Since $X\ T \Rightarrow A$ can be derived from G-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$, and $Y\ T \Rightarrow B$ can be derived from G-$\{X\ T \Rightarrow A\}$, $X\ T \Rightarrow A$ can be derived from G-$\{X\ T \Rightarrow A\}$. Therefore, $X\ T \Rightarrow A$ is redundant in G, and G is not a minimal cover, which is a contradiction. Thus, $X\ T \Rightarrow A$ is used in the derivation of $Y\ T \Rightarrow B$ from G. Since $X\ T \Rightarrow A$ is used in the derivation of $Y\ T \Rightarrow B$ from G, $Y\ T \Rightarrow X$ and $X\ A\ T \Rightarrow B$ can be derived from G.

Next, we prove that $X\ T \Rightarrow Y$ and $Y\ B\ T \Rightarrow A$ can be derived from G-$\{X\ T \Rightarrow A\}$. Since $G \equiv G$-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$, $Y\ T \Rightarrow B$ can be derived from G and $X\ T \Rightarrow A$ can be derived from G-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$. Since G is a minimal cover, $X\ T \Rightarrow A$ cannot be derived from G-$\{X\ T \Rightarrow A\}$, but it can be derived from G-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$. That is, $Y\ T \Rightarrow B$ is used in the derivation of $X\ T \Rightarrow A$ from G-$\{X\ T \Rightarrow A\} \cup \{Y\ T \Rightarrow B\}$. Thus, $X\ T \Rightarrow Y$ and $Y\ B\ T \Rightarrow A$ can be derived from G-$\{X\ T \Rightarrow A\}$, and also from G.

Therefore, $Y\ T \Rightarrow B$ is reduced, $X\ T \Leftrightarrow Y\ T$ and $X\ A\ T \Leftrightarrow Y\ B\ T$ can be derived from G, and $X\ T \Rightarrow Y$ and $Y\ B\ T \Rightarrow A$ can be derived from G-$\{X\ T \Rightarrow A\}$. Similarly, we can prove that if G - $\{X\ T \Rightarrow A$ (or $X\ T \rightarrow A)\} \cup \{Y\ T \Rightarrow B$ (or $Y\ T \rightarrow B)\}$ is a minimal cover of G, then there are Y and B such that $Y\ T \rightarrow B$ is reduced, $X\ T \leftrightarrow Y\ T$ and $X\ A\ T \leftrightarrow Y\ B\ T$ can be derived from G, and $X\ T \rightarrow Y$ and $Y\ B\ T \rightarrow A$ can be derived from G-$\{X\ T \rightarrow A\}$. $\square$

Appendix D

Appendix D

PROOF OF LEMMA 5.6.

Lemma 5.6: Algorithm 1 gives a dependency-preserving and lossless decomposition for PCNF and 3NFT.

Proof:

Let R be a temporal relation scheme, and G be a set of dependencies on R. Let R_0, R_1, R_2, …, and R_n be relation schemes obtained from the decomposition by Algorithm 1, and G_0, G_1, G_2, …, and G_n be sets of dependencies on R_0, R_1, R_2, …, and R_n, respectively.

It is obvious that the decomposition is dependency-preserving and lossless. From Step 1, MG is a minimal cover of G. Thus, $MG \equiv G$. From Step 2-7, $G_1 \cup G_2 \cup \ldots \cup G_n = MG$, and $G_0 = \emptyset$ because R_0 contains only the key of R. Therefore, $G \equiv G_1 \cup G_2 \cup \ldots \cup G_n$. Therefore, the decomposition is dependency-preserving. From Step 8, R_0 contains the universal key of R or there is R_i, for $1 \leq i \leq n$, such that R_i contains the universal key. A decomposition is lossless if there is a relation scheme in the decomposition that contains the universal key. Therefore, Algorithm 1 gives a lossless decomposition.

Finally, we prove that the relations created in Algorithm 1 are in 3NFT and PCNF. First, consider 3NFT. From Step 8, R_0 contains the key of R. Then, there is no dependency between attributes in R_0. Thus, R_0 is in 3NFT. From Step 1 and 2, dependencies on each relation created from Algorithm 1 are dependencies in the minimal cover with equivalent sets of attributes on the left-hand side. Similar to the decomposition of 3NF, there is no non-prime attribute in R_i which is transitively or partially dependent on the key of R_i. Thus, each R_i is in 3NFT.

Second, we prove that the relations created by Algorithm 1 are in PCNF. There is no non-homogeneous attribute in R_i created in Step 4. Thus, R_i is in PCNF according to Item 1 in Definition 5.4. For a relation scheme R_i created in Step 3, there are only functional dependencies on R_i and P-dependencies $X\ T \Rightarrow Y$ such that X, Y and T are in a functional dependency. Since there is no transitive dependency, according to 3NFT, no P-dependency can be derived. Thus, R_i created in Step 4 is in PCNF according to Item 2 in Definition 5.4. For a relation scheme R_i created in Step 5, there are only P-dependencies with non-homogeneous attributes. Thus, R_i created in Step 5 is in PCNF according to Item 3 in Definition 5.4. If R_0 is created in Step 8, it contains only the universal key. Since R_0 contains only the universal key, there is no non-trivial dependency on R_0. Thus, it is in PCNF. Therefore, the relation schemes created by Algorithm 1 are also in PCNF.

In conclusion, Algorithm 1 gives a dependency-preserving and lossless decomposition for PCNF and 3NFT. □

Appendix E

Appendix E

PROOF OF LEMMA 5.7.

***Lemma** 5.7:* Let R be a relation scheme, and G be a minimal cover on R, where G contains only P-dependencies which are either non-substitutable or 1-1 substitutable, and no two dependencies can have the same substitute dependency. A decomposition of a relation scheme R into $\{R_1, R_2,\ldots, R_n\}$, where G_1, $G_2,\ldots$, and G_n be minimal covers on R_1, $R_2,\ldots$, R_n, respectively, is a minimum decomposition if and only if, R_1, $R_2,\ldots$, R_n are created from Algorithm 1, and for each 1-1 substitutable dependency $X\ T \Rightarrow A$ in $G_1 \cup G_2 \cup \ldots \cup G_n$ such that X or A are non-homogeneous, there is no substitute dependency $Y\ T \Rightarrow B$ of $X\ T \Rightarrow A$ such that Y and B are homogeneous.

Proof:

Let $\{R_1, R_2,\ldots, R_n\}$ be a decomposition of R created from G in Algorithm 1, and G_1, $G_2,\ldots$, G_n be minimal covers on R_1, $R_2,\ldots$, R_n, respectively. Each dependency in $G_1 \cup G_2 \cup \ldots \cup G_n$ is either a non-substitutable dependency or a 1-1 substitutable. According to Algorithm 1, there is at most one relation which contains no non-homogeneous attribute. Then, let R_n be the relation which contains no non-homogeneous attributes. Thus, each of G_1, $G_2,\ldots,G_{n-1}$ contains a P-dependency with non-homogeneous attributes.

First, we prove the "if" direction. Let the decomposition of R into $\{R'_1, R'_2,\ldots, R'_m\}$ be another decomposition of R. Let $G'_1, G'_2,\ldots, G'_m$ be minimal covers on $R'_1, R'_2,\ldots, R'_m$, respectively, and $G'_1 \cup G'_2 \cup \ldots \cup G'_m \equiv G$. Let $G'_1, G'_2,\ldots, G'_i$ be sets of dependencies with non-homogeneous attributes, and $G'_{i+1}, G'_{i+2},\ldots, G'_m$ be sets of dependencies without non-homogeneous attributes. For each dependency in $G_1, G_2,\ldots,$ and G_n, there is a corresponding dependency in $G'_1, G'_2,\ldots, G'_m$, which is either the same dependency itself or its substitute dependency. For a P-dependency with non-homogeneous attributes, there is no substitute dependency without non-homogeneous attribute. Thus, for each dependency in G_n on R_n, there is a corresponding P-dependency in $G'_1, G'_2,\ldots$ or G'_m. For a P-dependency with non-homogeneous attributes, there is possibly a substitute dependency with or without non-homogeneous attributes. Thus, for each P-dependency with non-homogeneous attributes in $G_1, G_2,\ldots,G_{n-1}$, there is a corresponding P-dependency in $G'_1, G'_2,\ldots$ or G'_i. Since two P-dependencies with non-homogeneous attributes are not allowed in the same PCNF relation unless one contains the other, and no two dependencies can be replaced by the same dependency, $i \geq n-1$. There is at least one minimal cover among $G'_1, G'_2,\ldots, G'_m$ which contains a substitute dependency of a dependency in G_n or a dependency in G_n. Thus, $n \leq m$.

Now, we prove the "only if" direction. Let the decomposition of R into $\{R_1, R_2,\ldots, R_n\}$ be a minimal decomposition, and the decomposition of R into $\{R'_1, R'_2,\ldots, R'_m\}$ be a different decomposition of R, where $G'_1, G'_2,\ldots, G'_m$ be minimal covers on $R'_1, R'_2,\ldots,$

R'_m, respectively, and $G'_1 \cup G'_2 \cup \ldots \cup G'_m \equiv G$. Since the decomposition of R into $\{R_1, R_2, \ldots, R_n\}$ is a minimal decomposition, $n \leq m$. Since G contains only 1-1 and non-substitutable P-dependencies, for each $X\ T \Rightarrow A$ in $G_1, G_2, \ldots,$ and G_n, there are $X\ T \Rightarrow A$ or $Y\ T \Rightarrow B$, which is a substitute dependency of $X\ T \Rightarrow A$, in $G'_1, G'_2, \ldots, G'_m$. If $n \leq m$, there is no substitute dependency $Y\ T \Rightarrow B$ of $X\ T \Rightarrow A$ in G such that X or A are non-homogeneous and Y and B are homogeneous. Thus, for each 1-1 substitutable dependency $X\ T \Rightarrow A$ in G such that X or A are non-homogeneous, there is no substitute dependency $Y\ T \Rightarrow B$ of $X\ T \Rightarrow A$ such that Y and B are homogeneous. □

Appendix F

Appendix F

PROOF OF THEOREM 5.3

***Theorem** 5.3:* MD is NP-complete.

Proof:

First, we show that a special case of MD, where the minimal cover G contains only 1-1 substitutable or non-substitutable P-dependencies, is NP-complete.

For each dependency $X\ T \Rightarrow A$ in G such that X or A are non-homogeneous, if we non-deterministically guess a substitute dependency $Y\ T \Rightarrow B$ of $X\ T \Rightarrow A$ such that Y and B are homogeneous. Then, a new minimal cover G' is created by replacing $X\ T \Rightarrow A$ in G by $Y\ T \Rightarrow B$. A decomposition of R into $\{R_1, R_2,\ldots, R_n\}$ by using Algorithm 2 and G' is the minimum decomposition, as shown in Lemma 5.7, because G' does not contain $X\ T \Rightarrow A$, where X or A are non-homogeneous, and $X\ T \Rightarrow A$ can be replaced by $Y\ T \Rightarrow B$ such that Y and B are homogeneous. Since Algorithm 2 has polynomial-time complexity and G' can be created in polynomial time, the special case of MD is in NP.

Next, we show the polynomial-time reduction of SUB, which is NP-complete according to Lemma 5.5, to the special case of MD. The input R, G, and H of SUB and MD are the same. The output of MD is a minimum decomposition of R into $\{R_1, R_2,\ldots, R_n\}$. For $X\ T$

$\Rightarrow A$, which is an input of SUB, if there is R_i=(X A T), there is no substitute of X $T \Rightarrow A$ which contains only homogeneous attributes. Otherwise, there is a substitute Y $T \Rightarrow B$ of X $T \Rightarrow A$ which contains only homogeneous attributes. We can find Y $T \Rightarrow B$ by finding R_i such that X is in R_i and R_i contains only homogeneous attributes. Then, find the minimal cover G_i on R_i. G_i is a minimal cover of {Z $T \Rightarrow C$| Z $T \Rightarrow C$ is in G, and Z and C are in R_i.} G_i can be calculated in polynomial time. Then, for each Z $T \Rightarrow C$ is in G'-G, testing if Z $T \Rightarrow C$ is a substitute of X $T \Rightarrow A$ in G can be done in polynomial time. Thus, there is a polynomial-time reduction of SUB to the special case of MD. Therefore, the special case of MD is NP-complete, and MD is NP-complete. □

Bibliography

Bibliography

[1] Abadi, M., and Z. Manna. Temporal Logic Programming. *International Symposium on Logic Programming*, pp. 4-16, September 1987.

[2] Allen, J.F. Maintaining knowledge about temporal intervals. *Communication of the ACM*, Vol. 26, No. 11, pp. 832-843, 1983.

[3] Allen, J.F. Towards a General Theory of Action and Time. *Artificial Intelligence* Vol. 23, pp. 123-154, 1984.

[4] Aref, W.G. and H. Samet. Extending a DBMS with spatial operations. In O. Gunther and H.J. Schek (eds), *Advances in Spatial Databases,* Springer-verlag, 1991.

[5] Ariav, G. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, Vol. 11, No. 4, pp. 499-527, December 1986.

[6] Bassiouni, M.A., and M.J. Llewellyn. A relational-calculus query language for historical databases, *Comput. Lang.* Vol. 17, No. 3, pp. 185-197, 1992.

[7] Baudinet, M. Temporal Logic Programming is Complete and Expressive. *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages*, pp. 267-280, January 1989.

[8] Baudinet, M., J. Chomicki, and P. Wolper. Temporal Deductive Databases. In [80], pp.294-319.

[9] Beeri, C., and P.A. Bernstein. Problems in Design of Normal Form Relational Schemes. *ACM Transactions on Database Systems*, Vol. 4, No. 1, pp. 30-59, March 1979.

[10] Beeri, C., R. Fagin, and J. H. Howard. A Complete Axiomatization for Functional

and Multivalued Dependencies in Database Relations. *Proceedings of the ACM SIGMOD International Conference of Management of Data*, pp. 47-61, 1979.

[11] Ben-Zvi, J. *The Time Relational Model*. Ph.D. Thesis, Computer Science Department, UCLA, 1982.

[12] Bertino, E., E. Ferrari, and G. Guerrini. A Formal Temporal Object-Oriented Data Model. *Proceedings of the 5th International Conference on Extending Database Technology (EDBT)*, p. 342-356, 1996.

[13] Bolour, A., T.L. Anderson, L.J. Dekeyser, and H. K.T. Wong. The Role of Time in Information Processing: A Survey. *SIGMOD Record*, Vol. 12 No. 3, pp. 27-50, April 1982.

[14] Brent, M. R. Causal/Temporal Connectives: Syntax and Lexicon. *Technical Memo AIM-1122*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1989.

[15] Brusoni, V., L. Console, P. Terenziani, and B. Pernici. Extending Temporal Relational Databases to Deal with Imprecise and Qualitative Temporal Information. *Proceedings of the International Workshop on Temporal Databases*, pp. 3-22, September 1995.

[16] Bruce, B. A Model for Temporal Reference and Its Application in a Question Answering Program, *Artificial Intelligence* Vol. 3, No. 1, 1972.

[17] Chamberlain, D.D., et al. SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control. *IBM Journal of Research and Development*, Vol. 20, No. 6, pp. 560-575, 1976.

[18] Cheng, S., and S.K. Gadia. An Object-Oriented Model for Temporal Databases, *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, June 1993.

[19] Chomicki, J., and T. Imielinski. Temporal Deductive Databases and Infinite Objects. *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp.61-73, March 1988.

[20] Chomicki, J., and T. Imielinski. Relational Specifications of Infinite Query Answers. *Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data*, pp. 174-183, June 1989.

[21] Chomicki, J. Polynomial-Time Computable Queries in Temporal Deductive Databases. *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp.379-391, April 1990.

[22] Chomicki, J. *Functional deductive databases: Query processing in the Presence of Limited Function Symbols*, Ph.D. Thesis, Rutgers University, New Brunswick, NJ, 1990.

[23] Clifford, J., and A. Croker. The Historical Data Model (HRDM) and Algebra Based on Lifespans. *Proceedings of the International Conference on Data Engineering*, pp. 528-537, 1987.

[24] Clifford, J., and A. Tuzhilin. On algebra for historical relational databases: Two views. *Proceedings of the ACM SIGMOD International Conference of Management of Data*, pp. 247-265, 1985.

[25] Clifford, J., A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, Vol. 19, No. 1, pp. 64-116, March 1994.

[26] Clifford, J., and D.S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, Vol. 8, No. 2, pp. 214-254, June 1983.

[27] Codd, E.F. A Relational Model of Data for Large Shared Data Banks. *Communication of the ACM*, Vol. 13, No. 6, June 1970.

[28] Codd, E.F. Extending the Data Base Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, Vol. 4, No. 4, pp. 397-434, December 1979.

[29] Date, C.J. and H. Darwen. *A Guide to the SQL Standard*, 3rd Edition, Addison-Wesley Publishing Company, 1992.

[30] Dayal, U., and G. Wuu. Extending Existing DBMS's to Manage Temporal Data: An Object-Oriented Approach. *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, June 1993.

[31] Dudman, V. Conditional Interpretations of If-Sentences. *Australian Journal of Linguistics* Vol. 4, 1984.

[32] Dyreson, C.E. and R.T. Snodgrass. Valid Time Indeterminacy. *Proceedings of the International Conference on Data Engineering*, pp. 335-343, April 1993.

[33] Dyreson, C. E. and R.T. Snodgrass. Temporal Indeterminacy in TSQL2. In [75], pp. 327-346, Kluwer Academic Publishers, 1995.

[34] Edelweiss, N., J. Oliveira, and B. Pernici. An Object-Oriented Approach to a Temporal Query Language. *Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA)*, pp. 225-235, September 1994.

[35] Egenhofer, M.J. Reasoning about Binary Topological Relations. *Proceedings of the Second Symposium on the Design and Implementation of Very Large Spatial Databases*, 1991.

[36] Egenhofer, M.J. Spatial SQL: A Query and Presentation Language. *IEEE Transaction on Knowledge and Data Engineering*, Vol. 6 No. 1, pp. 86-95, 1994.

[37] Elmasri, R., V. Kouramajian, and S. Fernando. Temporal Database Modeling : An Object-Oriented Approach. *Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM)*, pp. 574-585, November 1993.

[38] Fillmore, C.J. The Case for Case. In Bach and Harms (Eds.), *Universals in Linguistic Theory*, Holt, Rinehart and Winston, New York, 1968.

[39] Gadia, S. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, Vol. 13, No. 4, pp. 418-448, December 1988.

[40] Gadia, S. A Seamless Generic Extension of SQL for Querying Temporal Data. *Technical Report TR-92-02*, Computer Science Department, Iowa State University, 1992.

[41] Gadia, S., and C. Yeung. A Generalized Model for a Relational Temporal Database. *Proceedings of the ACM SIGMOD International Conference of Management of Data*, pp. 251-259, 1988.

[42] Garey, M.R., and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[43] Golshani, F. Growing Certainty With Null Values. *Information Systems*, Vol. 10, No. 2, pp. 289-297, 1985.

[44] Guting, R.H. Geo-relational algebra: A Model and Query Language for Geometric Database Systems. *Proceedings of the International Conference on Extending Database Technology*, Venice, Italy, March 14-18, 1988.

[45] Jackendoff, R. Toward an Explanatory Semantic Representation. *Linguistic Inquiry*, Vol. 7, No. 1, pp. 89-150, 1976.

[46] Jensen, C.S., J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, and S. Jajodia. A Consensus Glossary of Temporal Database Concepts. *SIGMOD Record*, Vol. 23, No.1, pp. 52-64, March 1994.

[47] Jensen, C.S., R.T. Snodgrass, and M.D. Soo. The TSQL2 Data Model. In [75], pp. 157-240, Kluwer Academic Publishers, 1995.

[48] Jensen, C.S., R.T. Snodgrass, and M.D. Soo. Extending Existing Dependency Theory to Temporal Databases. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 4, pp. 563-581, August 1996.

[49] Jensen, C.S., M.D. Soo, and R.T. Snodgrass. Unification of Temporal Data Models. *Proceedings of 9th International Conference on Data Engineering*, pp. 262-271, April 1993.

[50] Jensen, C.S., M.D. Soo, and R.T. Snodgrass, Unifying Temporal Data Models via a Conceptual Model. *Technical Report TR-93-31*, Department of Computer Science, University of Arizona, September 1993.

[51] Lerat, N. Query Processing in Incomplete Logical Databases. *Proceedings of the International Conference on Database Theory*, pp. 260-277, Springer-Verlag, September 1986.

[52] Lorentzos, N.A. Management of Intervals and Temporal Data in the Relational Model. *Technical Report 49*, Agricultural University of Athens, Greece, 1991.

[53] Lorentzos, N.A., and R.G. Johnson. Extending Relational Algebra to Manipulate Temporal Data. *Information Systems*, Vol. 13, No. 3, pp. 289-296, 1988.

[54] Lorentzos, N.A., and R.G. Johnson. An Extension of the Relational Model to Support Generic Intervals. *Proceedings of the Extending Data Base Technology Conference*, 1988.

[55] Lorentzos, N.A., and Y.G. Mitsopoulos. SQL Extension for Interval Data. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 3, pp. 480-499, May/June 1997.

[56] McCarthy, J., and P.J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie (Eds.), *Machine Intelligence*,

Vol. 4, Edinburgh University Press, 1969.

[57] McCawley, J. Tense and Time Reference in English. In C. Fillmore and D. T. Langedoen (Eds.), *Studies in Linguistic Semantics*, Holt, New York, 1971.

[58] McDermott, D.A. Temporal Logic for Reasoning about Processes and Plans. *Cognitive Sciences*, Vol. 6, pp.101-155, December 1982.

[59] McKenzie, E., and R.T. Snodgrass. Extending Relational Algebra to Support Transaction Time. *Proceedings of the ACM SIGMOD International Conference of Management of Data*, pp. 467-478, May 1987.

[60] Melton, J. and A.R. Simon. *Understanding the New SQL: A Complete Guide*, Morgan-Kaufmann Publishers, 1992.

[61] Mourelatos, A.P.D. Events, Processes, and States. *Linguistics and Philosophy*, Vol. 2, pp.415-434, 1978.

[62] Navathe, S.B., and R. Ahmed. A Temporal Relational Model and a Query Language. *Information Sciences*, Vol. 49, pp.147-175, 1989.

[63] Orgun, M.A. On Temporal Deductive Databases. *Computational Intelligence*, Vol.12, No.2, 1996.

[64] Papadias, D., Y. Theodoridis, T. Sellis, and M.J. Egenhofer. Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees. *Proceedings of the 1995 ACM SIGMOD International Conference on the Management of Data*, pp. 92-103, 1995.

[65] Pissinou, N., K. Makki, and Y. Yesha. On Temporal Modeling in the Context of Object Databases. *SIGMOD Record*, Vol. 22, No. 3, pp. 8-15, September 1993.

[66] Prior, A. *Past, Present, Future*, Oxford University Press, 1967.

[67] Rescher, N.C., and A. Urquhart. *Temporal Logic*, Springer-Verlag, 1971.

[68] Roper, P. Interval and Tenses. *Journal of Philosophical Logic*, Vol. 9, No. 4, pp. 451- 469, November 1980.

[69] Rose, E., and A. Segev. TOOA: A Temporal Object-Oriented Algebra. *Proceedings of the ECOOP '93 European Conference on Object-oriented Programming*, pp. 297-

325, July 1993.

[70] Sarda, N.L. Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 2, No. 2, pp. 220-230, July 1990.

[71] Segev, A., and A. Shoshani. Logical Modeling of Temporal Data. *Proceeding of the 1987 ACM SIGMOD International Conference on the Management of Data*, pp. 454-466, 1987.

[72] Segev, A., and A. Shoshani. The Representation of a Temporal Data Model in the Relational Environment. *Proceeding of the 4th International Conference on Statistical and Scientific Database Management*, 1988.

[73] Shoham, Y. Temporal Logics in AI: Semantical and Ontological Considerations. *Artificial Intelligence*, Vol. 33, pp. 89-104, 1987.

[74] Snodgrass, R.T. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, Vol. 12, No. 2, pp. 247-298, June 1987.

[75] Snodgrass, R.T. *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, 1995.

[76] Snodgrass, R.T., and I. Ahn. A Taxonomy of Time in Databases. *Proceedings of the ACM SIGMOD International Conference of Management of Data*, pp. 236-246, May 1985.

[77] Snodgrass, R.T., and I. Ahn. Temporal databases. *IEEE Computer*, Vol. 19, No. 9, pp. 35-42, September 1986.

[78] Stonebraker, M., E. Wong, P. Kreps, and G. Held. The Design and Implementation if INGRES. *ACM Transactions on Database Systems*, Vol. 1, No. 3, pp. 189-2220, 1979.

[79] Tansel, A. Temporal Relational Data Model. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 9, No. 3, pp. 464-479, May/June 1997.

[80] Tansel, A., et al. *Temporal Databases: Theory, Design and Implementation*, The Benjamin/Cummings Publishing Company, Inc., 1993.

[81] Ullman, J.D. *Principles of Database and Knowledge-Base Systems*, Vol. 1, Computer Science Press, Inc., 1988.

[82] Wang, X.S., C. Bettini, A. Brodsky, and S. Jajodia. Logical Design for Temporal Databases with Multiple Granularities. *ACM Transactions on Database Systems*, Vol. 22, No. 2, pp. 115-170, June 1997.

[83] Wijsen, J. Design of Temporal Relational Databases Based on Dynamic and Temporal Functional Dependencies. *Recent Advances in Temporal Databases*, Springer-Verlag, pp.61-76, September 1995.

[84] Wijsen, J., J. Vandenbulcke and H. Olive. Functional Dependencies Generalized for Temporal Databases that Include Object-Identity. *Proceedings of the International Conference on the Entity-Relationship Approach*, pp. 100-114, 1993.

[85] Winslett, M. Updating Logical Databases Containing Null Values. *Proceedings of the International Conference on Database Theory*, pp. 421-435, Springer-Verlag, September 1986.

[86] Wuu, G.T.J. and U. Dayal. A Uniform Model for Temporal Object-Oriented Databases. *Proceedings of the International Conference on Data Engineering*, pp. 584-593, IEEE Computer Society Press, February 1992.

[87] Yip, K. M. Tense, Aspect and Cognitive Representation of Time. *Technical Memo AIM-815*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1984.

[88] Zaniolo, C. Database Relations with Null Values. *Journal of Computer and System Sciences*, Vol. 28, pp. 142-166, 1984.

MICHIGAN STATE UNIV. LIBRARIES
31293016826426