1993

A relational model for incomplete information in temporal databases

Sunil S. Nair
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation
Nair, Sunil S., "A relational model for incomplete information in temporal databases " (1993). Retrospective Theses and Dissertations. 10175.
https://lib.dr.iastate.edu/rtd/10175

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A relational model for incomplete information in temporal databases

Nair, Sunil S., Ph.D.
Iowa State University, 1993
A relational model for incomplete information in temporal databases

by

Sunil S. Nair

A Dissertation Submitted to the Graduate Faculty in Partial Fulfillment of the Requirements for the Degree of DOCTOR OF PHILOSOPHY

Department: Computer Science
Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa
1993
# TABLE OF CONTENTS

1. **INTRODUCTION**  
   1.1. Thesis Organization  
   1.2. The Basic Relational Model  

2. **SURVEY OF LITERATURE**  
   2.1. A Survey of Temporal Databases  
   2.2. Relational Databases with Null Values  

3. **A MODEL FOR TEMPORAL DATABASES**  
   3.1. Universe of Time and Temporal Elements  
   3.2. Attribute Values  
   3.3. $\theta$–navigation  
   3.4. Tuples and Relations  
   3.5. The Nature of Keys in the Model  
   3.6. Algebra for Complete Temporal Information  
      3.6.1. Temporal expressions  
      3.6.2. Boolean expressions  
      3.6.3. Relational expressions  

4. **OPTIMIZATION**  
   4.1. Algebraic Identities for the Model  
   4.2. Algorithm for Algebraic Optimization of Relational Queries  
   4.3. Algorithm for Cross Product  

5. **MODEL FOR INCOMPLETE TEMPORAL INFORMATION**  
   5.1. Partial Temporal Elements  
   5.2. Attributes  
   5.3. $\theta$–navigation  
   5.4. Tuples and Relations  
   5.5. Snapshots of Partial Temporal Relations  


### LIST OF FIGURES

<p>| Figure 2.1 | Codd's Union and Difference operators | 8 |
| Figure 2.2 | Codd's Projection operator | 8 |
| Figure 2.3 | Codd's TRUE and MAYBE Select operators | 9 |
| Figure 2.4 | Codd's TRUE and MAYBE Equijoin operators | 9 |
| Figure 2.5 | A relation with marked nulls | 11 |
| Figure 3.1 | A database | 14 |
| Figure 3.2 | Snapshot of the emp relation at t = 50 | 15 |
| Figure 3.3 | The management relation with MANAGER as the key | 16 |
| Figure 3.4 | Converting a temporal expression to an instantaneous condition | 19 |
| Figure 3.5 | Result of query in Example 3.5 | 20 |
| Figure 3.6 | Result of query in Example 3.7 | 20 |
| Figure 5.1 | Partial temporal elements and partial temporal assignments | 36 |
| Figure 5.2 | The emp relation | 39 |
| Figure 5.3 | Snapshot of emp relation at t = 50 | 41 |
| Figure 5.4 | Snapshot of emp relation at t = 55 | 41 |
| Figure 6.1 | Three valued truth tables for ( \land ), ( \lor ) and ( \neg ). | 43 |
| Figure 6.2 | Tables for ( \cap ), ( \cup ) and ( \neg ) of ( \langle 1, I \rangle ), ( \langle \emptyset, I \rangle ), ( \langle \emptyset, I \rangle ) where ( I = [0, NOW] ) | 44 |
| Figure 6.3 | The relation ( emp' ) | 47 |
| Figure 6.4 | Result of ( emp - emp' ) | 47 |
| Figure 6.5 | Result of selection in Example 6.3 | 49 |
| Figure 6.6 | Result of selection in Example 6.4 | 49 |
| Figure 6.7 | Result of query in Example 6.6 | 50 |
| Figure 7.1 | A completion of the emp relation | 55 |
| Figure 8.1 | A counterexample for ( \sigma(x; [B=b]) ) | 100 |</p>
<table>
<thead>
<tr>
<th>Figure 8.2</th>
<th>A counterexample for $\sigma(\sigma(r; [B&lt;b]); [B&gt;b])$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 9.1</td>
<td>Mary's tuple in emp relation</td>
</tr>
<tr>
<td>Figure 9.2</td>
<td>Mary's tuple after change operation</td>
</tr>
<tr>
<td>Figure 9.3</td>
<td>Mary's tuple after further change</td>
</tr>
<tr>
<td>Figure 9.4</td>
<td>Mary's tuple after still further change</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

In classical database systems only the current state of the database is maintained. As attributes change, their old values are discarded and replaced with new values. However, there are several applications in which it is important that historical information be maintained and a mechanism be provided to query the historical information. Databases which provide such features are called temporal databases.

In temporal database systems the time varying aspects of data are captured by time-stamping data values. A relational model for temporal databases should be a consistent extension of the classical relational database model. Thus, if all the information in a database describes a single snapshot of time then the temporal database model must give the same results as the classical relational model, at least up to an isomorphism. At the same time, a temporal database system must be able to capture the more complex nature of temporal data.

Research in temporal databases has concentrated on developing models in which it is essential that all the information be known. However, in the case of temporal databases the likelihood of missing information increases because of the vast amount of information being stored. Furthermore, users may want to maintain only selective portions of history, for instance only the salary history of certain employees. Therefore, there is a need to develop data models in which partial historical information can be stored and queried.

For the classical relational model there has been considerable research in the area of incomplete information (which exists but is unknown). In most models unknown values are marked by a special symbol called a null value, denoted \( \perp \). Since some attribute values are unknown a selection expression does not always evaluate to TRUE or FALSE for a given tuple. A third truth-value which we call UNDEFINED is sometimes introduced. Thus, in the result of a selection some tuples will be marked as definite tuple and some tuples as maybe tuples. A similar situation arises for the case of \( \theta \)-join operations.
In this work we present a relational model for temporal databases with incomplete information. The model consists of a storage model and an algebra for incomplete information. We show that our model satisfies certain desirable properties.

There are two points that must be considered in the storage model for incomplete information in temporal databases. First, for a given object, we may know the values for a given attribute at some points in time but the values at other points in time may be unknown. Second, at some points in time we are sure that the object must exist in the relation but at other points in time the existence of the object in the relation is not a certainty. Thus at some points in time the tuple is a definite tuple but at other points it is a maybe tuple.

Apart from the storage model to maintain partial histories, we also define a powerful algebra to query the incomplete historical information. The selection operation is an especially interesting operator in our model. It allows us to ask questions that have no counterpart in the classical case and it represents a departure of temporal databases from the classical snapshot databases. As a consequence, our results for incomplete temporal information cannot be obtained directly from corresponding results on incomplete snapshot information. Our model has the following properties:

1. The incomplete information model presented here generalizes the model for temporal databases with complete information described in Chapter 3. Thus, if our relation had no incomplete information our operators would give the same results as the operators in described in Chapter 3. Our generalization is seamless in the sense that queries which could be presented to a database with complete information can also be presented to our model without any change in syntax. Some remarks about further extending the querying capability of our model, without altering the structure of the syntax, are made in the Conclusions.

2. Our algebraic expressions produce results that are reliable in the sense that they never report incorrect information. This is shown by introducing the notion of completions of relations and databases.

3. Except for certain cases of selection, if the definition of the operators were strengthened to give more information, we may obtain results that are not
reliable. This result is obtained by first introducing the concepts of extensions of relations and more informative relations. This theorem does not extend to (i) certain cases of selection and (ii) arbitrary algebraic expressions. These limitations parallel Biskup's findings in classical databases with null values.

We also describe the syntax and semantics of update operations for the model so that the state of the database can be changed to reflect changes in the real world or to correct previous errors. The update operations we define for the model are create, change and changekey operations. Deletion of an object from a relation can be performed using the change operation.

1.1. Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2 we review some of the existing literature in the area of temporal databases and in the area of classical relations with null values. In Chapter 3 we focus our attention on a single model for temporal databases with complete information [GY88] which we use as a basis to develop our model for incomplete information. In Chapter 4 we develop algebraic identities that hold for the model and give an optimization algorithm for the model.

In Chapter 5 we introduce our model for incomplete temporal databases. In this chapter the concepts of temporal element, temporal assignments, tuples, relations and databases in the model for complete temporal information are generalized to capture incomplete information. The algebra to query the databases is given in Chapter 6. We show that the model is a consistent extension of the model for complete information.

In Chapter 7 we prove that our algebraic expressions produce reliable results. This is done by introducing the concept of completions of relations. Informally, a completion of a relation r is a relation which is consistent with r but having complete information. In Chapter 8 we introduce the notion of extensions of relations and the notion of more informative relations. We use these ideas to show that, except for certain cases of selection, the results of our operators are maximal in the sense that if our operators were defined so that the results were more informative we would lose the property of reliability.

In Chapter 9 we describe the syntax and semantics of update operations for the
Chapter 10 gives the conclusions and some possibilities for future research. We conclude this chapter with some basic definitions and notation concerning the relational model.

1.2. The Basic Relational Model

In this section we give some basic definitions and notation concerning the basic relational model [Co70]. Throughout the thesis we consider a fixed finite set of attributes. Attributes are usually denoted $A$, $B$, $C$ and sets of attributes are denoted $X$, $Y$, $Z$ possibly with subscripts. A set of attributes, say $\{A,C\}$ is usually written as $AC$. Associated with each attribute $A$ is an attribute domain $D(A)$. A domain is simply a set of values, not unlike a data type. By a tuple over $X$ we mean any mapping $\tau$ that associates a value $\tau(A) \in D(A)$ with every $A \in X$. A relation $r$ over a set of attributes $R$ is a finite set of tuples over $R$. It helps to view a relation as a table where each row is a tuple and each column corresponds to an attribute. The set of attribute names $R$ is called the \textit{scheme} of $r$. A database is a finite set of relations.

Lower case letters $r$, $s$ (possibly with subscripts) are often used to denote relations and the corresponding upper case letters $R$, $S$ (possibly with subscripts) are often used to denote the scheme of the corresponding relations. A relation $r$ having scheme $R$ is often denoted $r(R)$.

Let $\tau$ be a tuple in relation $r(R)$. If $A \in R$ then $\tau(A)$ is the value for the attribute $A$ in $\tau$. Similarly, if $X \subseteq R$ then $\tau[X]$ is the value for the attributes of $X$ in $\tau$.

Suppose $X$, $Y \subseteq R$. We say that $X \rightarrow Y$ (read $X$ functionally determines $Y$) holds in $r$ if for all $\tau_1, \tau_2 \in r$ ($\tau_1[X] = \tau_2[X] \Rightarrow \tau_1[Y] = \tau_2[Y]$).

Suppose $K \subseteq R$. We say $K$ is a key of $r$ if for all $\tau_1, \tau_2 \in r$ ($\tau_1[K] = \tau_2[K] \Rightarrow \tau_1 = \tau_2$).
2. SURVEY OF LITERATURE

There have been several attempts at defining relational models for temporal databases. In this section we briefly describe some of these approaches. We also survey the research conducted in the area of missing information for classical (snapshot) databases.

2.1. A Survey of Temporal Databases

One of the first attempts to incorporate a concept of time in relational databases was the work of Clifford and Warren [CW81]. In their model, each relation has two special attributes called \textit{STATE} and \textit{EXISTS?}. The attribute \textit{STATE} denotes the instant of time for which the tuple is valid. The model requires that all key values be entered at every state. The attribute \textit{EXISTS?} is then used to denote whether a particular object exists within the scope of interest at the instant denoted by \textit{STATE}. If an object does not exist at a given instant, the \textit{EXISTS?} column is set to 0 and the non-key attributes are set to null. Clearly there is a large amount of redundancy in the model.

In [Sn84,Sn87] Snodgrass presents a model in which a temporal domain is appended to each tuple of a relation. The value of the domain specifies the period of validity. There are two kinds of relations. \textit{Event relations} consist of tuples representing instantaneous occurrences. These relations have a single time value as the temporal domain, which is modeled by a special attribute called \textit{START}. \textit{Interval relations} consist of tuples valid during a time interval. Thus, the temporal domain contains two time values modeled by two special attributes called \textit{START} and \textit{STOP}. Since each attribute value of a tuple is a single atomic value this is a 1NF approach towards temporal databases. The author proposes a query language called TQuel to query a temporal database. This language is an extension of QUEL to handle temporal information. A "when clause" is introduced to make selections along the temporal
dimension. The "valid clause" specifies the value of the temporal domain in the derived relation.

Navathe and Ahmed [NA87] propose a model in which both time-varying and non-time-varying relations are allowed. Each time-varying relational schema has two special attributes called Time-Start ($T_S$) and Time-End ($T_E$). The authors define an extension of SQL called TSQL to handle temporal data. A **WHEN** clause is added which allows selections by specifying temporal relationships such as **BEFORE**, **AFTER**, **DURING**, **OVERLAP** etc. The **TIME-SLICE** clause allows restricting of the output to an interval or instant of interest.

A major problem with 1NF approaches to temporal databases is that the history of an object is not collected in a single tuple. Hence several queries cannot be expressed in a natural manner by languages developed for such models. Furthermore, each time any attribute value changes, a new tuple must be created in which all other attribute values are merely duplicated. This causes a high degree of redundancy.

Gadia and Vaishnav [GV85] introduce a non-1NF model for temporal databases in which the timestamping is at the attribute level and is based on temporal elements rather than intervals. A temporal element is a finite union of intervals. Temporal elements are closed under union, intersection and complementation and form a Boolean algebra. The use of temporal elements simplifies the query languages used to access temporal databases [GY91]. Furthermore, the use of the non-1NF model reduces data redundancy and allows the entire history of an object to be accessed in a single tuple.

Clifford [CT85] describes a model in which attributes can be of three types: Constant attributes, Time-varying attributes and Temporal attributes. Constant attributes are time invariant e.g. GENDER. Time-varying attributes can vary over time and are modeled as functions from time to some simple domain. Temporal attributes are those which have domain Time e.g. BIRTHDATE. A special symbol called **NULL** is used to terminate a field. The Comprehension Principle is the idea that a relation $r$ has complete information about the objects it models over its lifespan. The Continuity Assumption allows the interpolation of values not explicitly stored in the database.

Tansel [CT85] assumes $T = \{0,1,\ldots, NOW\}$ together with an ordering $\lt$. Intervals are of the type $[t_1,t_2)$. In his model attributes can be of four types: atomic, set-valued,
triplet-valued and set-triplet-valued. Atomic attributes are assigned atomic values, set-valued attributes are assigned values that are sets of atomic values, triplet-valued attributes are assigned triplets of the form $\langle t_1, t_2, a \rangle$ where $a$ is an atomic value, and set-triplet-valued attributes are assigned a finite set of triplets. Several new operations are introduced in the algebra. A PACK operation converts atomic and triplet-valued attributes to set-valued and set-triplet-valued attributes respectively. The UNPACK operation does the reverse. A triplet decomposition (T-DEC) operator breaks a triplet-valued attribute into its components, namely $T_{\text{START}}$, $T_{\text{END}}$, and the value. The triplet formation (T-FORM) operation does the reverse.

Other approaches can be found in Sarda [Sa90] and in Lorentzos and Johnson [LJ88]. Snodgrass and Ahn [SA85] give a taxonomy of time. Models with multi-dimensional timestamps can be found in [GY88, BG89b]. Tansel and Garnett [TG89] develop a nested historical database model. In their zero-information loss model, Bhargava and Gadia [BG89a] develop a mechanism in which even circumstantial information surrounding a transaction can be stored and queried. A 1NF proposal for incomplete temporal databases appears in [DS91]. Soo [So91] gives a bibliography on temporal databases.

2.2. Relational Databases with Null Values

For classical relational databases there has been considerable research in the area of relational databases with null values [Co79, Co86, Co87, Bi83, Ge90, Li81, IL84, Re86, Va79, Za84]. In this section we briefly survey some of the research in that area.

In [Co79] Codd proposes an extension to the relational model to incorporate null values. The meaning given to the null values is "value at present unknown." An occurrence of a null value is denoted by $\perp$. Because of missing information, it may not always be possible to determine if a given expression such as $A = B$ is TRUE or FALSE. Hence, an additional truth value which we call UNDEFINED is introduced. Then an expression like $A = B$ yields the truth value UNDEFINED if $A$ or $B$ or both are nulls. The truth-value UNDEFINED is assigned to an expression using the so-called null substitution principle. According to this principle, an expression is assigned the truth value UNDEFINED if both of the following conditions hold:
1. Each occurrence of \( x \) in the expression can be replaced by a non-null value (possibly a distinct one for every occurrence) so as to yield the value TRUE for the expression.

2. Each occurrence of \( x \) in the expression can be replaced by a non-null value (possibly a distinct one for every occurrence) so as to yield the value FALSE for the expression.

Duplicate removal, however, is done by a different rule. The null value \( x \) in one tuple is treated as being the same as an \( x \) in another tuple. This rule is used to define the Union, Difference and Projection operations. Figure 2.1 gives an example of Union and Difference. Figure 2.2 gives an example of Projection. The Cartesian Product operation remains unaffected.

![Figure 2.1. Codd's Union and Difference operators](image)

![Figure 2.2. Codd's Projection operator](image)

The Selection operation comes in two flavors: the TRUE Select and the MAYBE Select. The result of a TRUE Select operation contains only tuples for which the selection expression evaluates to TRUE. The MAYBE Select operation contains those tuples where the selection expression evaluates to UNDEFINED. An example of TRUE
Select and MAYBE Select is shown in Figure 2.3. Similarly, the Theta-join operator comes in two flavors: the TRUE Theta-join and the MAYBE Theta-join. Example of these are shown in Figure 2.4. Hurson and Miller [HM87] describe attribute MAYBE operators. These operators allow a user to retrieve more restrictive results than pure MAYBE operations, yielding better results and improved performance.

![Figure 2.3. Codd's TRUE and MAYBE Select operators](image1)

![Figure 2.4. Codd's TRUE and MAYBE Equijoin operators](image2)

In Codd's approach null values can be handled within the framework of a relational database system. However, there are no proposals of how maybe-results must be subsequently treated in relational expressions. Also, there is no common basis for the two different rules for processing \(\perp\), namely the null substitution principle and the duplicate removal rule. Furthermore, there are no results that show that the proposals are best in some sense.

Biskup [Bi83] tries to solve some of these problems in Codd's approach. Each relational scheme has a special attribute called STATUS, which can take on a value from...
the set \{d,m\}. If a tuple has a d in the STATUS column, it is a definite tuple. A tuple with an m in the STATUS column is a maybe tuple. Thus, in a single relation there may be some definite tuples and some maybe tuples. Maybe tuples are allowed as part of base relations. The operations Natural join, Projection, Union and Difference are extended to operate on relations that contain definite and maybe tuples. Selection operations are restricted to be of the type \( \sigma_{A=b}(r) \) and \( \sigma_{A=B}(r) \), where \( A, B \) are attributes and \( b \) is a constant. Theoretical soundness of the operations is shown through the notions of adequacy and restrictedness of operations. Informally, adequacy means that only correct information is provided. Restrictedness means that the operations "give as much information as possible" in the sense that if we try to strengthen the result of an operation to be more informative, then we lose the property of adequacy. All operations listed above are shown to be adequate and restricted. We call the corresponding results for temporal databases reliability and maximality respectively.

Lipski [Li84] describes a system based on marked nulls in which the null values are represented as variables rather than the generic null \( \bot \). This allows more information to be stored in the tables. For instance, Figure 2.5 shows that the teacher of the databases course, although unknown, is the same on Monday and Wednesday. This cannot be represented using the generic null symbol \( \bot \) of Codd tables. Imielinski and Lipski [IL84] introduce condition tables in which they use marked nulls and an additional column to store conditions which must be satisfied by each tuple. They then obtain a theorem which states that all valid conclusions are derivable in their system.

Besides the "value unknown" kind of null, several other kinds of null values have been introduced in the literature. Grant [Gr77] suggests a null value that can be interpreted as "non-existent". For instance, for an employee with no phone the attribute TEL# may be assigned the non-existent null. In that case the expressions TEL# = 2876543 and TEL# \neq 2876543 would both evaluate to \text{FALSE}. The problem of non-existent nulls was also studied by Lien [Li82].

Vassiliou [Va79] uses a denotational semantics approach to study the problem of both types of nulls occurring at once. Codd [Co86, Co87] and Gessert [Ge90] suggest a four-valued logic approach for database systems when both types of nulls are present. However, Codd indicates that the extra complexity is not currently justified.
Figure 2.5. A relation with marked nulls

<table>
<thead>
<tr>
<th>NAME</th>
<th>TEACHER</th>
<th>DAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>x</td>
<td>Monday</td>
</tr>
<tr>
<td>Languages</td>
<td>Tom</td>
<td>Tuesday</td>
</tr>
<tr>
<td>Databases</td>
<td>x</td>
<td>Wednesday</td>
</tr>
</tbody>
</table>

Zaniolo [Za84] introduces the "no-information" interpretation of null values. If the attribute TEL# of an employee tuple has a "no-information" null assigned to it, then either the employee has no telephone or the employee has a telephone but the number is unknown. Thus, the "no information" null is even more generic than the "non-existent" and "value unknown" nulls. Yue [Yu91] indicates that the presence of all three types of nulls leads to a 7-valued logic, which is too complicated. Other interpretations of null values are listed in [AP82].

We end this chapter by remarking that there is no completely satisfactory solution to the problem of relations with null values. Some problems with proposed solutions are discussed in [Ko89].
3. A MODEL FOR TEMPORAL DATABASES

In this chapter we describe a model for temporal databases [GY88]. In Chapter 4 we develop algebraic identities and an algebraic optimizer for the model. We also use the model of [GY88] as a basis to develop our model for incomplete temporal information.

3.1. Universe of Time and Temporal Elements

We assume a universe \([0, \text{NOW}]\) of time instants together with a linear order \(\preceq\) on it. Although it is not necessary, we assume for simplicity that \([0, \text{NOW}]\) is the discrete set \(\{0,1,\ldots, \text{NOW}\}\). NOW denotes the current time according to the system clock. An interval is a subset \(I\) of \([0, \text{NOW}]\) such that any instant between two instants in \(I\) is also an instant in \(I\). Intervals are not adequate to model history of an object in a single tuple and lead to query languages that are difficult to use [GY91]. Hence, a temporal element is defined to be a finite union of intervals. An interval is a temporal element. An instant \(t\) may be identified with the interval \([t,t]\); thus it may be regarded as a temporal element. The set of all temporal elements is closed under \(\cup\), \(\cap\) and \(\neg\) (complementation) and forms a boolean algebra (see Theorem 3.1 below).

**Theorem 3.1.** The set of all temporal elements together with \(\cup\), \(\cap\), \(\neg\), \(\emptyset\), and \([0, \text{NOW}]\) forms a boolean algebra (for a definition of boolean algebra see [TM75]).

**Proof:**
Let \(\mu_1\), \(\mu_2\) and \(\mu_3\) be temporal elements. It can be easily shown that the following identities hold. Hence the set of temporal elements together with \(\cup\), \(\cap\), \(\neg\), \(\emptyset\), and \([0, \text{NOW}]\) forms a boolean algebra.

**Commutativity**
\[
\mu_1 \cap \mu_2 = \mu_2 \cap \mu_1 \\
\mu_1 \cup \mu_2 = \mu_2 \cup \mu_1
\]
Associativity

\[(\mu_1 \cap \mu_2) \cap \mu_3 = \mu_1 \cap (\mu_2 \cap \mu_3)\]
\[(\mu_1 \cup \mu_2) \cup \mu_3 = \mu_1 \cup (\mu_2 \cup \mu_3)\]

Absorption

\[\mu_1 \cap (\mu_1 \cup \mu_2) = \mu_1\]
\[\mu_1 \cup (\mu_1 \cap \mu_2) = \mu_1\]

Complementation

\[\mu_1 \cap \neg \mu_1 = \emptyset\]
\[\mu_1 \cup \neg \mu_1 = [0, \text{NOW}]\]

Distributivity

\[\mu_1 \cap (\mu_2 \cup \mu_3) = (\mu_1 \cap \mu_2) \cup (\mu_1 \cap \mu_3)\]
\[\mu_1 \cup (\mu_2 \cap \mu_3) = (\mu_1 \cup \mu_2) \cap (\mu_1 \cup \mu_3)\]

3.2. Attribute Values

To capture the changing values of an attribute a temporal assignment to an attribute A is defined to be a function from a temporal element into the domain of A. An example of a temporal assignment to the attribute COLOR is ([25,32] red, [33, NOW] blue). If \(\xi\) is a temporal assignment, \([\xi]\) denotes its domain. Thus \([([25,32] \text{ red}, [33, \text{NOW}] \text{ blue}]) = [25, \text{NOW}]. \(\xi_1|_\mu\) denotes the restriction of \(\xi\) to the temporal element \(\mu\). Thus (([25,32] red, [33, NOW] blue)'[25,30] = ([25,30] red).

3.3. \(\theta\)-navigation

The counterpart of the construct \(A \theta B\) of the conventional relational model that we use is \([A \theta B]\), which captures the time when A is in \(\theta\)-relationship to B. This is introduced through \([\xi_1 \theta \xi_2] = \{t: \xi_1 \text{ and } \xi_2 \text{ are defined at } t, \text{ and } \xi_1(t) \theta \xi_2(t) \text{ is TRUE}\}\). For example \([([25,32] \text{ red}, [33, \text{NOW}] \text{ blue}) = ([0, \text{NOW}] \text{ blue}]) = [33, \text{NOW}]. The construct \([A \theta b]\), where \(b\) is a constant, is also allowed, and is evaluated by identifying the constant \(b\) with the assignment \([0, \text{NOW}]\) b.
3.4. Tuples and Relations

A *tuple* is simply a concatenation of assignments whose temporal domains are the same. The assumption that all temporal assignments in a tuple have the same domain is called the *homogeneity* assumption [GV85, Ga88]. The temporal domain of a tuple \( \tau \), denoted \( \langle \tau \rangle \), is simply the temporal domain of any of its temporal assignments. If \( \tau \) is a tuple, \( \tau \upharpoonright \mu \) is obtained by restricting each assignment in \( \tau \) to the temporal element \( \mu \).

A *relation* \( r \) over a scheme \( R \), with \( K \subseteq \mathcal{R} \) as its *key*, is a finite set of non-empty tuples such that no key attribute value in a tuple changes with time, and no two tuples agree on all their key attributes. If \( r \) is a relation, then \( r \upharpoonright \mu \) is obtained by restricting each tuple in \( r \) to the temporal element \( \mu \). Figure 3.1 shows a database with a relation \( \text{emp}(\text{NAME \ SALARY \ DEPT}) \) with \( \text{NAME} \) as its key, and a relation \( \text{management}(\text{DEPT \ MANAGER}) \) with \( \text{DEPT} \) as its key. The *snapshot* of a relation \( r \) at an instant \( t \), denoted \( r(t) \), is defined in a natural manner. For example, the snapshot at \( t = 50 \) of the \( \text{emp} \) relation in Figure 3.1 is shown in Figure 3.2.

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[8,44] Toys</td>
</tr>
<tr>
<td>[40,52]</td>
<td></td>
<td>[45,52] 20K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shoes</td>
</tr>
<tr>
<td>[48,NOW]</td>
<td>Doug</td>
<td>[48,NOW] 20K</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[48,NOW] Auto</td>
</tr>
</tbody>
</table>

The \( \text{emp} \) relation

<table>
<thead>
<tr>
<th>DEPT</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>[40,NOW]</td>
<td></td>
</tr>
<tr>
<td>[6,39] Auto</td>
<td>[48,NOW] Doug</td>
</tr>
</tbody>
</table>

The \( \text{management} \) relation

Figure 3.1. A database
In Figure 3.1, all the information about a given employee is captured in a single tuple. If an employee could work in more than one department at the same time then it would not be possible to capture all the information about a given employee in a single tuple. Instead, all the information about a NAME DEPT pair would be captured in a single tuple. Thus, the notion of an object in the model is closely tied to the idea of the key of the relation.

The temporal domain of a relation, denoted $[[r]]$, is the union of the temporal domain of its tuples. For example, for the management relation in Figure 3.1, $[[management]] = [6, NOW]$.

3.5. The Nature of Keys in the Model

Keys play a critical role in the model. A key provides a persistent identity to an object. In classical databases the choice of a key has no effect on the structure of the relation. However, this is not so in the case of temporal databases. Two relations $r$ and $s$ are said to be weakly equal if for all instants $t$ in $[0, NOW]$ we have $r(t) = s(t)$ [Ga86a]. In temporal databases two relations $r$ and $s$ may be weakly equal but their structures may still be different. For instance, the management relation in Figure 3.1 is weakly equal to the management relation in Figure 3.3, but the structure of the two relations are different. In fact, even the number of tuples in the two relations is not the same. In the management relation the key is DEPT but in the management relation the key is MANAGER. The following theorem [GY88] is easily proved.

**Theorem 3.2.** Suppose $r$ is a relation over $R$ with key $K$. Suppose $K' \subseteq R$ such that for each instant $t$ in $[0, NOW]$ we have $K' \rightarrow R$ in $r(t)$. Then there is unique relation $s$ weakly equal to $r$ but with key $K'$.
3.6. Algebra for Complete Temporal Information

The set of all algebraic expressions can be divided into three mutually exclusive groups: \textit{temporal expressions}, \textit{boolean expressions}, and \textit{relational expressions}.

3.6.1. Temporal expressions

Temporal expressions are the syntactic counterpart of temporal elements. They are formed using temporal elements, $[A]$, $[A \theta B]$, $[A \theta_b]$, and $[E]$ where $E$ is a relational expression. More complex temporal expressions are formed using $\cup$, $\cap$, and $\neg$. If $\mu$ is a temporal expression and $\tau$ is a tuple, then $\mu(\tau)$ evaluates to a temporal element and is defined in a natural way.

The construct $[E]$ in the model makes the system highly recursive since temporal expressions can contain relational expressions. As we shall see in Section 3.6.3, relational expressions can contain temporal expressions.

\textbf{Example 3.1}. Consider the emp relation of Figure 3.1. Suppose $\tau$ denotes John's tuple. Then $[\text{SALARY} \leq 20K](\tau)$ evaluates to $[8,39]$. Similarly, $[\text{DEPT}=\text{Shoes}](\tau) = [45,52]$. Also, the result of substituting $\tau$ in the temporal expression $[\text{SALARY} \leq 20K] \cup [\text{DEPT}=\text{Shoes}]$ is $[8,39] \cup [45,52]$. The result of substituting $\tau$ in the temporal expression $[\text{SALARY}]$ is $[8,52]$. Similarly, $[\text{management}](\tau) = [6,\text{NOW}]$.

3.6.2. Boolean expressions

Atomic boolean expressions are of the form TRUE, FALSE and $\mu \subseteq \nu$, where $\mu$ and $\nu$ are temporal expressions. More complex expressions are formed using the boolean
operators $\land, \lor$ and $\neg$. Note that expressions of the form $\mu = \nu, \mu \neq \nu$ etc. can be derived using the above constructs.

#### 3.6.3. Relational expressions

Relational expressions are the syntactic counterpart of temporal relations. Before we describe the relational operators we first introduce the idea of weakly invariant operators. A relational operator is said to be *weakly invariant* [GY88] if it transforms weakly equal relations to weakly equal relations. We now describe the relational operators.

**Restructuring.** The purpose of the restructuring operator is to change the key of a relation. Suppose $r$ is a relation over $R$ with $K$ as its key. Then if $K' \subseteq R$ such that $K' \rightarrow R$ in all snapshots of $r$, then $I_{K'}^r(r)$ is the unique relation weakly equal to $r$ but with key $K'$. For example, $I_{\text{MANAGER}}^r(\text{management})$ is the management relation shown in Figure 3.2. Since $I_{K'}^r(r)$ is weakly equal to $r$, $(I_{K'}^r(r))(t) = r(t)$ for all $t$. An example of a query using the restructuring operation is given in Example 3.4. In the literature $I_{K'}^r(r)$ is sometimes denoted as $r:K'$.

**Union.** If $r$ and $s$ are relations over the same scheme $R$ and the same key $K$, then $r \cup s$ also has the same scheme and key. To arrive at $r \cup s$, we first compute the union of $r$ and $s$ treating them as sets, and then collapse each pair of tuples of $r$ and $s$ which agree on all key attributes into a single tuple. Note that this may give a run-time error if the two tuples being collapsed have different values at some non-key attribute at the same instant of time. A study of this phenomenon is beyond the scope of this thesis. The union operation is weakly invariant and hence $(r \cup s)(t) = r(t) \cup s(t)$.

**Difference.** Suppose $r$ and $s$ are temporal relations with the same scheme and key. Then $r - s$ also has the same scheme and key. The relation $r - s$ is computed as follows. We start with $r$. Then for each tuple of $r$ we check to see if there is a tuple in $s$ which agrees on all key attributes. If there is no such tuple then $\tau$ does not change. If $s$ has such a tuple $\tau'$ then those instants where $\tau$ and $\tau'$ agree on all attributes are removed from the domain of $\tau$. If the resulting domain of $\tau$ is $\emptyset$ then $\tau$ is rejected. This basically amounts to removing the overlap between $\tau$ and $\tau'$ from $\tau$. It is easy to see that difference is a weakly invariant operation. Hence, $(r - s)(t) = r(t) - s(t)$.
Projection. To define $\Pi_X(r)$, it is required that the key of $r$ be a subset of $X$. Then $\Pi_X(r)$ is defined to be $\{\tau(X): \tau \in r\}$. The projection operation is weakly invariant and hence $(\Pi_X(r))(t) = \Pi_X(r(t))$.

Selection. Selection is a powerful operator in temporal databases. If $r$ is a relation, $f$ is a boolean expression and $\mu$ is a temporal expression then the selection $\sigma(r; f; \mu)$ evaluates to $\{\tau \mid \mu(\tau) \land f(\tau) \land T(\tau) \neq \emptyset\}$. If $f$ evaluates to TRUE for a tuple, $\sigma$ allows us to select only a relevant part of it, which is specified by $\mu$. For example, consider the database in Figure 3.1. If the parameter $f$ is omitted in $\sigma(r; f; \mu)$ it defaults to TRUE. If $\mu$ is omitted it defaults to $[0, \text{NOW}]$. The key of $\sigma(r; f; \mu)$ is the same as the key of $r$. Because of the non-1NF nature of the relations in the model, some queries which can be expressed naturally as a selection in the model would require a join if the relations were required to be in 1NF form [Ga92]. For instance, the query give information about employees while they were in Toys or Shoes if they are currently employed can be expressed as $\sigma(\text{emp}; [\text{NOW}, \text{NOW}] \subseteq [\text{name}] ; [\text{dept}=\text{Toys}] \cup [\text{dept}=\text{Shoes}])$. The query would have to be expressed as a join if the relations were required to be in 1NF.

The selection operator is not a weakly invariant operator (see Example 3.2). Thus, the selection operator represents a departure of temporal databases from classical snapshot databases. The fact that temporal expressions, boolean expressions and relational expressions can contain each other makes the system highly recursive and represents a further departure of temporal databases from classical snapshot databases.

Example 3.2. Consider the management relation in Figure 3.1 and the management relation in Figure 3.3. The management relation is weakly equal to the management relation. However, $\sigma(\text{management}; [8, \text{NOW}] \subseteq [\text{dept}=\text{Toys}] ; [0, \text{NOW}])$ is not weakly equal to $\sigma(\text{management}; [8, \text{NOW}] \subseteq [\text{dept}=\text{Toys}] ; [0, \text{NOW}])$. The first expression yields the relation with the Toys tuple from management relation while the second expression yields the empty relation.

Example 3.2 shows that $\sigma(r; f; \mu)$, in general, is not weakly invariant. However, when the parameter $f$ is omitted, the selection operator is weakly invariant. In other words, $\sigma(r; ; \mu)$ is weakly invariant. A tuple $\tau$ belongs to the snapshot $\sigma(r; ; \mu)(t)$ if and only if $\tau$ belongs to the snapshot $r(t)$ and $\tau$ satisfies a condition $C_{\mu,t}$ which is obtained as shown in Figure 3.4.
Example 3.3. For any instant $t$, $\tau \in \sigma(\text{emp} ; [\text{DEPT}=\text{Toys}] \cup [\text{DEPT}=\text{Auto}]) (t)$ if and only if $\tau \in \text{emp}(t)$ and $\tau$ satisfies the formula $(\text{DEPT}=\text{Toys} \lor \text{DEPT}=\text{Auto})$. This can be easily verified using the emp relation from Figure 3.1.

<table>
<thead>
<tr>
<th>$\mu$</th>
<th>$C_{\mu,t}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>temporal element</td>
<td>$t \in \mu$</td>
</tr>
<tr>
<td>$[[A]]$</td>
<td>TRUE</td>
</tr>
<tr>
<td>$[[A \land b]]$</td>
<td>$A \land b$</td>
</tr>
<tr>
<td>$[[A \lor B]]$</td>
<td>$A \lor B$</td>
</tr>
<tr>
<td>$[[E]]$</td>
<td>$t \in [[E]]$</td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$</td>
<td>$C_{\mu_1,t} \lor C_{\mu_2,t}$</td>
</tr>
<tr>
<td>$\mu_1 \cap \mu_2$</td>
<td>$C_{\mu_1,t} \land C_{\mu_2,t}$</td>
</tr>
<tr>
<td>$\mu_1 - \mu_2$</td>
<td>$C_{\mu_1,t} \land \neg C_{\mu_2,t}$</td>
</tr>
</tbody>
</table>

Figure 3.4 Converting a temporal expression to an instantaneous condition

Cross product. A tuple in $r \times s$ is obtained by concatenating a tuple in $r$ and a tuple in $s$, and only preserving the instants where both the tuples are defined. This assures the homogeneity of $r \times s$. The key of $r \times s$ is the union of the keys of $r$ and $s$. Let $\tau_r$, $\tau_s$ be two tuples. Define $\text{hom}(\tau_r \circ \tau_s)$ to be the homogeneous part of the concatenation of $\tau_r$ and $\tau_s$. We may then define $r \times s = \{\text{hom}(\tau_r \circ \tau_s) : \tau_r \in r \land \tau_s \in s \land \text{hom}(\tau_r \circ \tau_s) \text{ is not empty}\}$. We also note that $(r \times s)(t) = r(t) \times s(t)$.

The following theorem summarizes the weakly invariant operators in the framework [GY88].

**Theorem 3.3.** The union, difference, projection, restructuring, cross-product operators and selection operators of the form $\sigma(r ; \mu)$ are weakly invariant.

We end this chapter with a few examples of queries in the model. The examples use the database shown in Figure 3.1.
Example 3.4. This example illustrates the use of the restructuring operation and the selection operation. The query give information from management relation for managers who were managers at least during [11,40] is expressed below. The query retrieves Jack's tuple from the management relation.

\[
\sigma(I_{\text{MANAGER}}(\text{management}); [11,40] \subseteq [\text{MANAGER}]); \]

Example 3.5. The query give details about employees while they were working in the Shoes department is expressed below, and the result is shown in Figure 3.5.

\[
\sigma(\text{emp}; \text{TRUE}; [\text{DEPT} = \text{Shoes}]).
\]

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
</table>

Figure 3.5. Result of query in Example 3.5

Example 3.6. The query give details about employees while they had a salary greater than 24K and worked in the either the Clothing or the Shoes department is expressed below. The expression below shows a direct relationship between "and", "or" and \( \cap \), \( \cup \) respectively. For the emp relation, the query retrieves the empty relation.

\[
\sigma(\text{emp}; \text{TRUE}; ([\text{SALARY} > 24K] \cap ([\text{DEPT} = \text{Shoes}] \cup [\text{DEPT} = \text{Clothing}])).
\]

Example 3.7. The query find the managers of employees and display the result with employee name as the key is expressed below, and the result is shown in Figure 3.6.

\[
\Pi_{\text{NAME}, \text{MANAGER}}(I_{\text{NAME}}(\sigma(\text{emp} \times \text{management}; ;[\text{emp.DEPT} = \text{management.DEPT}])).
\]

<table>
<thead>
<tr>
<th>NAME</th>
<th>MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>[40,44] Jack</td>
<td></td>
</tr>
<tr>
<td>[48,NOW] Doug</td>
<td>[48,NOW] Doug</td>
</tr>
</tbody>
</table>

Figure 3.6. Result of query in Example 3.7
4. OPTIMIZATION

Most of the research in temporal databases has concentrated on the development of models to store and query the historical information. There has been relatively less work done in the area of optimization for temporal queries.

In snapshot databases, there have been two general approaches to query optimization. The first approach consists of the use of index structures to improve performance [Th88, Sa87, Wi87]. The second is the use of equivalences between relational expressions [SC75, Sc86, Ul88]. Such equivalences have been extensively studied for snapshot databases. These equivalences are used in query optimization to convert a given user query to another equivalent query that would require fewer disk accesses. Heuristics such as "perform selections as early as possible" are used in this approach [SC75, Ul88].

While there has been some work in the area of index structures for temporal databases [EWK90, GS91, KS89], there has been little effort to develop algebraic optimization techniques for models in temporal databases. Since the operators in our algebra for temporal databases are different from those in traditional snapshot databases, there is a need to study equivalences among algebraic expressions and use them to optimize temporal queries.

In this chapter we develop algebraic identities for the model for temporal databases described in Chapter 3. The selection operator in the model represents a substantial departure from classical snapshot databases. The result of a selection operator cannot always be evaluated by considering snapshots of a temporal relation. Furthermore, the restructuring operator in the model has no counterpart in the classical case. This departure of the selection operator from the classical selection and the introduction of the restructuring operator make it necessary to develop new algebraic identities for the temporal algebra. We present an algorithm that uses the identities to convert a given relational expression to another equivalent expression which would execute more efficiently. We also give an algorithm for computing the cross-product of
two relations. This algorithm yields a substantial saving over the brute force algorithm when (i) the relations store history spanning a long period of time relative to the length of history of individual tuples and (ii) the tuples in the relation are sorted by increasing order of the first instants in their time domains.

4.1. Algebraic Identities for the Model

In this section we develop algebraic identities for the temporal database model. These identities can be used to convert a given user query to an equivalent query which will be more efficient to execute. We note that while we use \( r \) and \( s \) as the operands in the identities, the operands could also be algebraic expressions. We give the proof for some of these identities but omit the proof for others. The proofs of some of the identities use the following theorem, which follows immediately from Theorem 3.2.

**Theorem 4.1.** Suppose \( r \) and \( s \) are relations with the same scheme and key. If \( r \) and \( s \) are weakly equal (i.e. \( \forall t \in [0, \text{NOW}] \) we have \( r(t) = s(t) \)), then \( r = s \).

1. **Commutativity of cross-product.**
   \[ r \times s = s \times r \]
   Since our columns are named the above identity holds.

2. **Associativity of cross-product.**
   \[ q \times (r \times s) = (q \times r) \times s \]

3. **Cascade of projections.** A cascade of projections can be combined into a single projection by using the following rule:
   \[ \Pi_{A_1, \ldots, A_n}(\Pi_{B_1, \ldots, B_m}(r)) = \Pi_{A_1, \ldots, A_n}(r) \]
   For the left-hand side in the above equality to be defined, we need \( A_1, \ldots, A_n \subseteq B_1, \ldots, B_m \).

4. **Cascade of selections.** In the most general case a cascade of selections cannot be combined into a single selection i.e. in general, \( \sigma(\sigma(r; f_1, \mu_1); f_2, \mu_2) \neq \sigma(r; f_1, f_2, \mu_1, \mu_2) \) as shown in Example 4.1. However, there are several important cases in which two or more selections can be combined into a single selection. These are listed below.
(a) \( \sigma(\sigma(r; f_1; ) ; f_2; ) = \sigma(r; f_1 \land f_2; ) \)

**Proof:**
\[
\tau \in \sigma(\sigma(r; f_1; ) ; f_2; )
\implies \tau \in \sigma(r; f_1; ) \land f_2(\tau)
\implies \tau \in r \land f_1(\tau) \land f_2(\tau)
\implies \tau \in r \land (f_1 \land f_2)(\tau)
\implies \tau \in \sigma(r; f_1 \land f_2; ).
\]
Hence, the result follows. 

Since \( f_1 \land f_2 = f_2 \land f_1 \), we have the following commutativity rule for selection.

(b) \( \sigma(\sigma(r; f_1; ) ; f_2; ) = \sigma(\sigma(r; f_2; ) ; f_1; ) \)

The cascade \( \sigma(r; \mu_1); \mu_2 \) can also be converted to a single selection using the following rule.

(c) \( \sigma(\sigma(r; \mu_1); \mu_2) = \sigma(r; \mu_1 \cap \mu_2) \)

**Proof:**
Let \( t \) be an instant in \([0, NOW]\).
\[
\tau \in \sigma(\sigma(r; \mu_1); \mu_2)(t)
\implies \tau \in \sigma(r; \mu_1)(t) \land C_{\mu_2,t}(\tau)
\implies \tau \in r(t) \land C_{\mu_1,t}(\tau) \land C_{\mu_2,t}(\tau)
\implies \tau \in \sigma(r; \mu_1 \cap \mu_2)(t)
\]
Hence, for all \( t \) in \([0, NOW]\), \( \sigma(\sigma(r; \mu_1); \mu_2)(t) = \sigma(r; \mu_1 \cap \mu_2)(t) \)

Also, the key of both sides of the equation in (c) is the same.

By I, II and Theorem 4.1 the result follows. 

Since \( \mu_1 \cap \mu_2 = \mu_2 \cap \mu_1 \), we have the following commutativity rule for selection.

(d) \( \sigma(\sigma(r; \mu_1); \mu_2) = \sigma(\sigma(r; \mu_2); \mu_1) \)

We can also easily prove the following rule.

(e) \( \sigma(\sigma(r; f; ); \mu) = \sigma(r; f; \mu) \)

**Proof:**
\[
\tau \in \sigma(r; f; \mu)
\implies \exists \tau' \in r \text{ such that } f(\tau') \land (\tau' \vdash \mu(\tau') = \tau) \text{ and } \tau' \vdash \mu(\tau') \text{ is not empty.}
\]
\[ \Rightarrow \exists \tau' \in \sigma(r; f_1) \text{ such that } \tau' \models \mu(\tau') = \tau \text{ and } \tau' \models \mu(\tau') \text{ is not empty} \]

\[ \Rightarrow \tau \in \sigma(r; f_1); i, \mu \]

**Example 4.1.** This counterexample shows that the equality \( \sigma(\sigma(r; f_1; \mu_1); f_2; \mu_2) = \sigma(r; f_1 \land f_2; \mu_1 \cap \mu_2) \) does not hold. Let \( r(AB) \) be a relation with the single tuple \([0,10]a; [0,10]b\), let \( f_1 = f_2 = [0,10][[B=b]] \), and let \( \mu_1 = \mu_2 = [0,5] \). Then, the left hand side of the equality evaluates to the empty relation while the right hand side evaluates to the relation with the single tuple \(([0,5]a; [0,5]b)\).

**Example 4.2.** This counterexample shows that the equality \( \sigma(r; f; \mu) = \sigma(\sigma(r; \mu); f) \) does not hold. Let \( r(AB) \) be a relation with the single tuple \(([0,10]a; [0,10]b)\), let \( f = [0,10] [[B=b]] \), and let \( \mu = [0,5] \). Then, the right hand side of the equality evaluates to the empty relation while the left hand side evaluates to the relation with the single tuple \(([0,5]a; [0,5]b)\).

5. **Commutativity of selection and projection.** Under certain conditions selections and projections can be commuted as described in the following rules.

   (a) If \( f \) involves only attributes in \( A_1 \ldots A_n \), then

   \[ \sigma(\Pi_{A_1...A_n}(r); f) = \Pi_{A_1...A_n}(\sigma(r; f)) \]

   **Proof:**

   \[ \Rightarrow \exists r' \in \sigma(\sigma(r; f)) \text{ such that } r'[A_1...A_n] = r \\ \Rightarrow \exists r' \in r \text{ such that } r'[A_1...A_n] = r \land f(r) \]

   \[ \Rightarrow \exists \tau' \in r \text{ such that } \tau'[A_1...A_n] = r \land f(\tau') \] (since \( f \) only involves attributes in \( A_1 \ldots A_n \)).

   \[ \Rightarrow \exists \tau' \in \sigma(r; f) \text{ such that } \tau'[A_1...A_n] = r \]

   \[ \Rightarrow \tau \in \Pi_{A_1...A_n}(\sigma(r; f)) \]

   Hence the result

   (b) If \( \mu \) involves only attributes in \( A_1 \ldots A_n \), then

   \[ \sigma(\Pi_{A_1...A_n}(r); i, \mu) = \Pi_{A_1...A_n}(\sigma(r; i, \mu)) \]

   **Proof:**

   Let \( t \) be an instant in \([0, \text{NOW}]\)

   \[ \tau \in \sigma(\Pi_{A_1...A_n}(r); i, \mu)(t) \]

   \[ \Rightarrow \tau \in (\Pi_{A_1...A_n}(r))(t) \land C_{\mu_t}(\tau) \]
\[
\exists \tau' \in r(t) \land C_{\mu,t}(\tau') \land \tau'[A_1 \ldots A_n] = \tau
\]

\[
\exists \tau' \in r(t) \land C_{\mu,t}(\tau') \land \tau'[A_1 \ldots A_n] = \tau \quad \text{(since } \mu \text{ only involves attributes in } A_1 \ldots A_n\text{)}.
\]

\[
\exists \tau' \in \sigma(r; ;\mu)(t) \land \tau'[A_1 \ldots A_n] = \tau
\]

\[
\tau \in (\Pi_{A_1 \ldots A_n}(\sigma(r; ;\mu)))(t)
\]

Hence \( \forall t \in [0, \infty] \), \( \sigma(\Pi_{A_1 \ldots A_n}(r; ;\mu))(t) = (\Pi_{A_1 \ldots A_n}(\sigma(r; ;\mu)))(t) \)

Also, the key of both sides is the same

Hence from I, II and Theorem 4.1, the result follows \( \square \)

The following identity subsumes both the above identities. However, the proof requires the use of (5a) and (5b) proved above.

(c) If \( f \) and \( \mu \) involve only attributes in \( A_1 \ldots A_n \), then

\[
\sigma(\Pi_{A_1 \ldots A_n}(r; f; \mu)) = \Pi_{A_1 \ldots A_n}(\sigma(r; f; \mu))
\]

PROOF:

\[
\sigma(\Pi_{A_1 \ldots A_n}(r; f; \mu)) = \sigma(\Pi_{A_1 \ldots A_n}(r; f; ); ;\mu) \quad \text{(by 4e)}
\]

\[
= \sigma(\Pi_{A_1 \ldots A_n}(r; f; ); ;\mu) \quad \text{(by 5a)}
\]

\[
= \Pi_{A_1 \ldots A_n}(\sigma(r; f; ); ;\mu) \quad \text{(by 5b)}
\]

\[
= \Pi_{A_1 \ldots A_n}(\sigma(r; f; ;\mu)) \quad \text{(by 4e)}
\]

Hence the result. \( \square \)

6. **Commutativity of selection and cross-product.** The cross-product is the most expensive operator. Reducing the size of the operands in the cross-product significantly reduces the cost of query execution. The following identities give rules for commuting a selection with a cross product.

(a) If all the attributes of \( \mu \) are attributes in \( R \) (the scheme of \( r \)) then

\[
\sigma(r \times s; ;\mu) = \sigma(r; ;\mu) \times s
\]

(b) Similarly, if all the attributes of \( \mu \) are attributes in \( S \) (the scheme of \( s \)) then

\[
\sigma(r \times s; ;\mu) = r \times \sigma(s; ;\mu)
\]

The proofs of (6a) and (6b) are left to the reader.

However, even if all the attributes in \( f \) are attributes in \( R \), the equality \( \sigma(r \times s; f; ) \times s \) does not necessarily hold. This is illustrated in Example 4.3.
Example 4.3. Let \( r(AB) \) consist of the single tuple \([0,5]a; [0,5]b\) and let \( s(CD) \) consist of the single tuple \([0,3]c; [0,3]d\). Let \( f \) be the formula \([0,5] \subseteq [B=b]\). Then, the expression \( \sigma(rxs;f;) \) evaluates to the empty relation while \( \sigma(r;f;)xs \) evaluates to the relation with the single tuple \([0,3]a; [0,3]b; [0,3]c; [0,3]d\). □

Example 4.3 also shows that the equality \( \sigma(rxs;f;/i) = \sigma(r;f;/i) x s \) does not hold. However, the operand \( r \) in the cross product may be made smaller by the following identity.

(c) If \( \mu \) has attribute only in \( R \) then
\[
\sigma(r \times s;f;\mu) = \sigma(\sigma(r;\mu \neq \emptyset;) \times s;f;\mu)
\]

**Proof:**

\[
\tau \in \sigma(r \times s;f;\mu)
\]
\[
\Rightarrow \exists \tau' \in r \times s [f(\tau') \wedge \tau' \upharpoonright \mu(\tau') = \tau \wedge \tau' \upharpoonright \mu(\tau') \text{ is not empty}]\\
\Rightarrow \exists \tau'_r \in r, \exists \tau'_s \in s [\text{for some } \tau'[f(\tau') \wedge \tau' \upharpoonright \mu(\tau') = \tau \wedge \tau' \upharpoonright \mu(\tau') \text{ is not empty } \wedge \tau' = \text{hom}(\tau'_r \circ \tau'_s)]\\
\Rightarrow \exists \tau'_r \in r, \exists \tau'_s \in s [\text{for some } \tau'[f(\tau') \wedge \tau' \upharpoonright \mu(\tau') = \tau \wedge \tau' \upharpoonright \mu(\tau') \neq \emptyset]\\
\Rightarrow \tau \in \sigma(\sigma(r;\mu \neq \emptyset;) \times s;f;\mu)
\]

Hence \( \sigma(r \times s;f;\mu) \subseteq \sigma(\sigma(r;\mu \neq \emptyset;) \times s;f;\mu) \)

\[
\tau \in \sigma(\sigma(r;\mu \neq \emptyset;) \times s;f;\mu)
\]
\[
\Rightarrow \exists \tau' \in \sigma(r;\mu \neq \emptyset;) \times s [f(\tau') \wedge \tau' \upharpoonright \mu(\tau') = \tau \wedge \tau' \upharpoonright \mu(\tau') \text{ is not empty}]\\
\Rightarrow \exists \tau'_r \in \sigma(r;\mu \neq \emptyset;) \times s, \exists \tau'_s \in s [\text{for some } \tau'[f(\tau') \wedge \tau' \upharpoonright \mu(\tau') = \tau \wedge \tau' \upharpoonright \mu(\tau') \text{ is not empty } \wedge \tau' = \text{hom}(\tau'_r \circ \tau'_s)]\\
\Rightarrow \exists \tau'_r \in r, \exists \tau'_s \in s [\text{for some } \tau'[f(\tau') \wedge \tau' \upharpoonright \mu(\tau') = \tau \wedge \tau' \upharpoonright \mu(\tau') \text{ is not empty}]\\
\Rightarrow \tau \in \sigma(r \times s;f;\mu)
\]

Hence \( \sigma(\sigma(r;\mu \neq \emptyset;) \times s;f;\mu) \subseteq \sigma(r \times s;f;\mu) \)

From I and II the result follows □
In fact, \( \sigma(r \times s; f; \mu) = \sigma(\sigma(r; \mu \cap [A] \neq \emptyset) \times s; f; \mu) \) where \( A \) is any attribute in \( R \), since we can easily show that \( \sigma(r \times s; f; \mu) = \sigma(r \times s; f; \mu \cap [A]) \).

7. **Commutativity of selection and union.** If \( r \) and \( s \) have the same scheme and same key and if \( r \cup s \) is defined then
\[
\sigma(r \cup s; f; \mu) = \sigma(r; f; \mu) \cup \sigma(s; f; \mu)
\]
However, the equality \( \sigma(r \cup s; f; \mu) = \sigma(r; f; \mu) \cup \sigma(s; f; \mu) \) does not hold in general. This is shown in Example 4.4.

Example 4.4. Let \( r(AB) \) and \( s(AB) \) have key \( A \). Let \( r(AB) \) consist of the single tuple \(([0,5]a; [0,5]b) \) and let \( s(AB) \) consist of the single tuple \(([6,10]a; [6,10]b) \). Let \( f \) be the formula \([0,5]c[B=b]\). Then \( \sigma(r \cup s; f; \mu) \) evaluates to the single tuple relation \(([0,10]a; [0,10]b) \) while \( \sigma(r; f; \mu) \cup \sigma(s; f; \mu) \) evaluates to the relation consisting of the single tuple \(([0,5]a; [0,5]b) \).

8. **Commutativity of selection and difference.** If \( r \) and \( s \) have the same scheme and same key then
\[
\sigma(r - s; f; \mu) = \sigma(r; f; \mu) - \sigma(s; f; \mu)
\]
However, in general, \( \sigma(r - s; f; \mu) \neq \sigma(r; f; \mu) - \sigma(s; f; \mu) \) as can be easily shown by an example.

9. **Commutativity of projection and cross-product.** If \( A_1 \ldots A_n \) is a list of attributes of which \( B_1 \ldots B_m \) are attributes of \( r \) and \( C_1 \ldots C_k \) are attributes of \( s \) then
\[
\Pi_{A_1 \ldots A_n}(r \times s) = \Pi_{B_1 \ldots B_m}(r) \times \Pi_{C_1 \ldots C_k}(s)
\]

10. **Commutativity of projection and union.** If \( r \cup s \) is defined then
\[
\Pi_{A_1 \ldots A_n}(r \cup s) = \Pi_{A_1 \ldots A_n}(r) \cup \Pi_{A_1 \ldots A_n}(s)
\]

We now examine the interaction of the restructuring operation with the other relational operations.

11. **Commutativity of restructuring and union.** If both sides of the equality are defined then
\[
I_K(r \cup s) = I_K(r) \cup I_K(s)
\]
Proof:
The key of both sides is \( K \)
(I_K(r \cup s))(t)
= (r \cup s)(t)
= r(t) \cup s(t)
= (I_K(r))(t) \cup (I_K(s))(t)

Hence, \((I_K(r \cup s))(t) = (I_K(r) \cup I_K(s))(t)\)

\[ \text{II} \]

From I, II, and Theorem 4.1, the result follows. \(\square\)

12. **Commutativity of restructuring and difference.** If \(K \rightarrow R\) (the scheme of \(r\)) and \(K \rightarrow S\) (the scheme of \(s\)) then
\[ I_K(r - s) = I_K(r) - I_K(s) \]

The proof of this is similar to the proof of rule 11.

13. **Commutativity of restructuring and cross-product.** If \(K_1 \rightarrow R\) in \(r\) and \(K_2 \rightarrow S\) in \(s\) and \(K = K_1K_2\) where \(K_1\) is the part of \(K\) in \(R\) and \(K_2\) is the part of \(K\) in \(S\) then
\[ I_K(r \times s) = I_K(r) \times I_K(s) \]

Note that the key of both sides is \(K_1K_2\).

14. **Commutativity of restructuring and projection.** The restructuring operation and the projection operation can be commuted using the following equality. We note however that for the left hand side of the equality to be well-defined we need \(A_1\ldots A_n \subset \text{key of } r\) and for the right hand side to be well-defined we need \(A_1\ldots A_n \subset K\) (see Section 3.6.3).
\[ I_K(\Pi_{A_1\ldots A_n}(r)) = \Pi_{A_1\ldots A_n}(I_K(r)) \]

\[ \text{PROOF:} \]

The key of both sides is \(K\)
\[ \tau \in (I_K(\Pi_{A_1\ldots A_n}(r))) \]
\[ \iff \tau \in (\Pi_{A_1\ldots A_n}(r)) \]
\[ \iff \exists \tau_1 \in r(t) \text{ such that } \tau_1[A_1\ldots A_n] = \tau \]
\[ \iff \exists \tau_1 \in (I_K(r))(t) \text{ such that } \tau_1[A_1\ldots A_n] = \tau \]
\[ \iff \tau \in (\Pi_{A_1\ldots A_n}(I_K(r)))(t) \]

Hence, \((I_K(\Pi_{A_1\ldots A_n}(r)))(t) = (\Pi_{A_1\ldots A_n}(I_K(r)))(t)\)

\[ \text{II} \]

From I, II and Theorem 4.1, the result follows \(\square\)

15. **Commutativity of restructuring and selection.** If \(K \rightarrow R\) (the scheme of \(r\)) holds in
If \( r \) then
\[
I_K(\sigma(r; ;\mu)) = \sigma(I_K(r); ;\mu)
\]

**Proof:**

The key of both sides is \( K \)

\[
\tau \in (I_K(\sigma(r; ;\mu)))(t)
\leftrightarrow \tau \in \sigma(\sigma(r; ;\mu))(t)
\leftrightarrow \tau \in (I_K(\mu)_{c,t}(\tau)
\leftrightarrow \tau \in (I_K(\mu)_{c,t}(\tau)
\leftrightarrow \tau \in \sigma(I_K(r); ;\mu)(t)
\]

Hence, \((I_K(\sigma(r; ;\mu)))(t) = \sigma(I_K(r); ;\mu)(t)\)

From I, II and Theorem 4.1, the result follows.

However, the equality \( I_K(\sigma(r; ;\mu)) = \sigma(I_K(r); ;\mu) \) does not necessarily hold. This is shown in Example 4.5.

**Example 4.5.** Let \( r(AB) \) be the relation with key \( A \) having the single tuple \(([0,10]a; [0,5]b, [6,10]b)\). Let \( f = [0,5] \subseteq [B=b] \). Then \( I_B(\sigma(r; ;f)) \) has the tuples \(([0,5]a; [0,5]b) \) and \(([6,10]a; [6,10]b) \). However, the result of \( \sigma(I_B(r); ;f) \) has only the tuple \(([0,5]a; [0,5]b) \).

16. **Cascade of restructuring operations.** If \( K_1 \rightarrow R \) and \( K_2 \rightarrow R \) hold in \( r \) then
\[
I_{K_1}(I_{K_2}(r)) = I_{K_1}(r)
\]

Note that the key of both sides is the same \( (K_1) \)

**4.2 Algorithm for Algebraic Optimization of Relational Queries**

In this section we sketch an algorithm that uses the identities from the previous section. The algorithm converts a given temporal query to a more optimal query. The algorithm uses certain heuristics to decide how a given expression is to be converted to another expression. Since the cross-product is the most expensive operator in temporal databases (as it is in classical snapshot databases), the algorithm tries to reduce the size of the operands in a cross-product. Similarly, the restructuring operator is a fairly expensive operator in temporal databases and the algorithm attempts to reduce the size of the operand to the restructuring operator. These and other ideas in the algorithm...
are listed below.

- **Perform selections as early as possible.** A selection reduces the size of the relation. Since the result of the selection may be an operand in a larger expression this could reduce the cost of execution of the query.

- **Perform projections as early as possible.** A projection also reduces the size of the relation by making the tuples shorter. However, note that in our algebra the number of tuples does not decrease since the key of the relation is always present in the projection.

- **Reduce size of operands in a cross-product.** As mentioned previously, the cross-product is the most expensive relational operator. Reducing the size of the operands in a cross-product drastically reduces the cost of the operation. The size of the operands is usually reduced by using the two heuristics above.

- **Reduce size of operands in the restructuring operation.** The restructuring operator is also a fairly expensive operator and the cost of executing the operation reduces quite drastically if the size of the operand is reduced. This can be done by using the first two heuristics mentioned above. The size of the operand can also be reduced by pushing the restructuring operation ahead of the cross-product to yield two restructuring operations but on much smaller relations (see rule 13 in Section 4.1). Note that on the average the cost of the cross-product stays the same.

- **Combine cascades of unary operations.** It may sometimes be possible to convert two selections into a single selection operator. Similarly, in a cascade of projection operators all but the last projection can be eliminated. Also, in a cascade of restructuring operations only the last one needs to be executed.

- **Remove redundant operations.** If $R$ is the scheme of an expression $E$, then $\Pi_R(E) = E$ and the redundant projection can be removed. Similarly, if $K$ is the key of an expression $E$, then $I_K(E) = E$ and the restructuring operation is redundant.

- **Combine operations that can be executed simultaneously.** Sometimes two or more operations can be executed simultaneously in a single step. For instance, in the expression $\Pi_X \sigma(r \times s; f; \mu)$ the selection and projection can be done while performing the cross-product. Similarly, in $\Pi_X (\sigma(I_K(r); f; \mu)$ the selection and projection can be performed while doing the restructuring.

We now sketch the algorithm that uses the equivalences from the previous section.
Algorithm 4.1. Apply the following sequence of steps in the given order to convert a given query to another equivalent but more efficient query.

1. Apply rules (4a) to (4e) to convert a single selection into a cascade of selections. We do this because it may be easier to push a smaller selection further down the expression tree rather than a larger selection. Also, note that the order of the cascade is important and various orders must be tried in a practical implementation. For instance, it may be better to convert $\sigma(e; \mu_1 \cap \mu_2)$ to $\sigma(\sigma(e; \mu_2); \mu_1)$ rather than to $\sigma(\sigma(e; \mu_1); \mu_2)$ since it may be possible to push the selection on $\mu_2$ (but not the selection on $\mu_1$) inside the expression $e$.

2. Apply rules 4, 5, 6 (including generalization of 6c), 7, 8 and 15 to move selections as far down the tree as possible.

3. Apply rules 3, 5, 9, and 14 to move projections as far down the expression tree as possible. Eliminate redundant projections where possible.

4. Apply rules 3, 4, 5 to convert a cascade of selections and projections into a single selection, a single projection or a selection followed by a projection.

5. Use rules 13 and 16 to move restructuring operations down the tree. Rule 13 moves restructuring operations before the cross-product. Rule 16 eliminates all but the last restructuring operation in a cascade of restructuring operations. Also, eliminate a restructuring operation if it restructures an expression on its own key i.e. eliminate redundant restructuring.

6. Identify sequences of operations that can be executed simultaneously. For instance, in $\Pi^X(\sigma(I_K(r); f; \mu))$ the selection and projection can be performed while doing the restructuring.

We illustrate the algorithm with two examples. In the examples the key attributes in a relation are underlined.

Example 4.6. Suppose $r(A B C)$ and $s(D E F G)$ are two relations. Consider the query

$$\Pi_{ABDF}(\sigma(r \times s; [0,5] \subseteq [A]; [NOW, NOW]));$$

Applying the generalized version of rule 6(c) twice, we get

$$\Pi_{ABDF}(\sigma(r; [A] \cap [NOW, NOW] \neq \emptyset); \times \sigma(s; [D] \cap [NOW, NOW] \neq \emptyset); [0,5] \subseteq [A]; [NOW, NOW]);$$
Now, using rule 5 followed by rule 9, we get
\[\sigma(\Pi_{AB}\sigma(r; [[A]] \cap [NOW,NOW] \neq \emptyset); \times \Pi_{DF}\sigma(s; [[D]] \cap [NOW,NOW] \neq \emptyset); [0,5] \subset [[A]]; [[NOW,NOW]])\]

This can be computed in the following steps:

**Step 1.** Compute \(s_1 = \Pi_{AB}\sigma(r; [[A]] \cap [NOW,NOW] \neq \emptyset); \) in a single step. The selection restricts \(r\) to those tuples that have information about the current instant.

**Step 2.** Compute \(s_2 = \Pi_{DF}\sigma(s; [[D]] \cap [NOW,NOW] \neq \emptyset); \) in a single step. The selection restricts \(s\) to those tuples that have information about the current instant.

**Step 3.** Compute \(\sigma(s_1 \times s_2; [0,5] \subset [[A]]; [[NOW,NOW]])\) in a single step. The cross-product is performed on relations smaller than in the original query, thus improving efficiency.

**Example 4.7.** Suppose \(r(A,B,C)\) and \(s(D,E,F,G)\) are relations and suppose we know that \(C \rightarrow ABC\) holds in \(r\). Consider the query \(\Pi_{CDE}(\sigma(I_{CD}(r \times s); [[B=5]] \cap [[E=F]])\). Using the algorithm this query can be shown to be equivalent to the query \(\Pi_{C}(I_{C}(\sigma(r; [[B=5]]))) \times \Pi_{DE}\sigma(s; [[E=F]])\). This query can be evaluated in the following steps:

**Step 1.** Compute \(s_1 = \sigma(r; [[B=5]])\).

**Step 2.** Compute \(s_2 = \Pi_{C}(I_{C}(s_1))\) in a single step.

**Step 3.** Compute \(s_4 = \Pi_{DE}\sigma(s; [[E=F]])\) in a single step.

**Step 4.** Compute \(s_2 \times s_4\).

### 4.3. Algorithm for Cross Product

In this section we discuss an algorithm to compute the cross product of two relations \(r\) and \(s\) when the tuples in each relation are sorted by the initial instant of the time domain of each tuple (if the relations are not so sorted the relations will first have to be sorted before the algorithm is applied). The algorithm yields a significant savings over the brute force algorithm especially when the relations store information spanning a long period of time relative to the length of the history of individual objects. The algorithm uses the fact that the result of the cross-product operation must be homogeneous. Hence, when considering a block \(b\) in relation \(r\) there is no need to read
blocks of relation \( s \) where we know there will be no tuples whose time domains intersect some tuple of \( b \).

We first introduce some notation. For a tuple \( \tau \) let \( \tau.\text{lower} \) denote the first instant at which the tuple is defined and let \( \tau.\text{upper} \) be the last instant at which the tuple is defined. For example, if \( \tau \) is John's tuple in Figure 3.1, \( \tau.\text{lower} = 8 \) and \( \tau.\text{upper} = 52 \). Also let \( \tau_{\text{max}}(b) \) be the tuple in block \( b \) with the largest upper limit. We now present the algorithm for computing \( r \times s \).

```plaintext
cursor1 ← 1; cursor2 ← 1; maxsofar ← 0;
for i = cursor1 to m do /* m is the number of blocks in \( r \) */
    begin
        read block(i) from \( r \) into \( b_r^{i} \);
        for j = cursor2 to n do /* n is the number of blocks in \( s \) */
            begin
                read block(j) from \( s \) into \( b_s^{j} \);
                concatenate tuples of \( b_r^{i}, b_s^{j} \) preserving homogeneous parts;
                /* Let \( \tau_1 \) be the last tuple in \( b_r^{i} \) and
                \( \tau_2 \) be the last tuple in \( b_s^{j} \) */
                maxsofar = max(maxsofar, \( \tau_{\text{max}}(b_s^{j}).\text{upper} \))
                if \( \tau_1.\text{lower} > \text{maxsofar} \) then
                    tempcursor2 ← j + 1; /* blocks of \( r \) to be read later
                    will not need blocks of \( s \) up to \( j \) */
                if \( \tau_{\text{max}}(b_r^{i}).\text{upper} < \tau_2.\text{lower} \) then
                    j ← n; /* temporal domain of tuples in the current block of \( r \)
                    will not intersect with any other blocks of \( s \) */
            endfor
        cursor2 ← tempcursor2
    endfor
```

□
5. MODEL FOR INCOMPLETE TEMPORAL INFORMATION

In this chapter, we define our model for temporal databases with incomplete information, called *partial temporal databases*, by generalizing notions of a temporal element, temporal assignment, tuples and relations to capture incomplete information.

5.1. Partial Temporal Elements

In the case of complete information, an expression like \([A=B]\) yields a temporal element which is the set of instants during which \(A=B\). When \(A\) and \(B\) have missing information we may not be able to compute this set exactly. Hence the knowledge of instants when we are sure \(A=B\) is \text{TRUE}, and instants when we are sure that \(A=B\) is \text{FALSE} is important. This leads to the notion of a *partial temporal element*, which is defined to be a pair \((\ell,u)\) where \(\ell \subseteq u\); \(\ell\) and \(u\) are called the lower and upper limit of the partial temporal element, respectively (see Figure 5.1(a)). Now \([A=B]\) yields a pair \((\ell,u)\), where \(\ell\) is a set of instants when \(A=B\) definitely holds, and \(u\) is a set of instants beyond which \(A=B\) could not hold.

Note that a temporal element \(\mu\) can be represented as the partial temporal element \((\mu,\mu)\). Thus partial temporal elements are a generalization of temporal elements.

The operations \(\cup, \cap, -\) and \(-\) are generalized as follows:

- **Union:** \(\ell_1 \cup u_1 \cup \ell_2 \cup u_2 \)
  
  e.g. \([0,5],[0,20] \cup [4,15],[0,15] = [0,15],[0,20]\).

- **Intersection:** \(\ell_1 \cap u_1 \cap \ell_2 \cap u_2 \)

- **Difference:** \(\ell_1 \cup u_1 - \ell_2 \cup u_2 \)
  
  e.g. \([0,5],[0,20] - [4,15],[0,15] = [0,3] \cup [16,20]\).
• Complementation: $\neg\langle \ell_1, u_1 \rangle = \langle u_1, \neg \ell_1 \rangle$
eq e.g. $\neg\langle [0,5] \cup [8,9], [0,20] \rangle = \langle [21, \text{NOW}], [6,7] \cup [10, \text{NOW}] \rangle$.  

The operators on the left hand side are operations on partial temporal elements while the operators on the right hand side are operations on temporal elements. The complementation operator can be defined in terms of the difference operator since $\langle \neg u_1, \neg \ell_1 \rangle = \langle [0, \text{NOW}], [0, \text{NOW}] \rangle - \langle \ell_1, u_1 \rangle$.  

The set of partial temporal elements is closed under the operations defined above. The following theorem is easily proved.

**Theorem 5.1.** The set of partial temporal elements together with $\cup$ and $\cap$ satisfy the following conditions and hence form a distributive lattice (for a definition of a lattice see [TM75]).

**Proof**

Suppose that $\mu_1$, $\mu_2$ and $\mu_3$ are partial temporal elements. Then it can be easily shown that the following identities hold.

**Commutativity**

$\mu_1 \cap \mu_2 = \mu_2 \cap \mu_1$

$\mu_1 \cup \mu_2 = \mu_2 \cup \mu_1$

**Associativity**

$(\mu_1 \cap \mu_2) \cap \mu_3 = \mu_1 \cap (\mu_2 \cap \mu_3)$

$(\mu_1 \cup \mu_2) \cup \mu_3 = \mu_1 \cup (\mu_2 \cup \mu_3)$

**Absorption**

$\mu_1 \cap (\mu_1 \cup \mu_2) = \mu_1$

$\mu_1 \cup (\mu_1 \cap \mu_2) = \mu_1$

**Distributivity**

$\mu_1 \cap (\mu_2 \cup \mu_3) = (\mu_1 \cap \mu_2) \cup (\mu_1 \cap \mu_3)$

$\mu_1 \cup (\mu_2 \cap \mu_3) = (\mu_1 \cup \mu_2) \cap (\mu_1 \cup \mu_3)$

\[\square\]

5.2. Attributes

In our model a *partial temporal assignment* to an attribute $A$ is a triple $\xi \ell u$, where $\xi$ is a temporal assignment (as defined in Chapter 3), $\ell$ and $u$ are temporal
elements such that \([\xi] \subseteq u\) and \(\ell \subseteq u\) (see Figure 5.1(b)). Note that we do not require that \([\xi] \subseteq \ell\). A snapshot of a tuple during \(\ell\) gives us a definite tuple of [Bi83] and a snapshot at instants in \(u - \ell\) gives us a maybe tuple of [Bi83] (see Chapter 2). Allowing \(\xi\) to be defined beyond \(\ell\) lets us capture maybe information. Allowing \(\xi\) to be defined beyond \(\ell\) also lets us reduce uncertainty as algebraic expressions are evaluated.

![Diagram](image)

(a) A partial temporal element
(b) Relationship between \([\xi]\), \(\ell\) and \(u\) in a partial temporal assignment \(\xi \ell u\)

Figure 5.1. Partial temporal elements and partial temporal assignments

The *restriction* of \(\xi \ell u\) to a partial temporal element \(\langle \ell', u' \rangle\), denoted \((\xi \ell u)'(\ell', u')\), is defined as \((\xi' u')(\ell' \cap \ell').(u' \cap u)\). The triple \(\xi \ell u\) when it is assigned to an attribute encodes the following information:

- During \(\ell\) we are sure that the object exists.
- Beyond \(u\) the object does not exist.
- During \(u - \ell\) we are uncertain about the existence of the object.
- During \(\ell \cap \xi\) we know that the object exists and the values it takes.
- During \(\ell - \xi\) the object exists but its values are unknown.
- During \(u - \xi\) the object may exist, but we do not know the values.
- During \(\xi - \ell\) if the object exists we know the values.

In the model for complete information an attribute is assigned a temporal assignment \(\xi\). This complete information can be represented in our model for
incomplete information by the partial temporal assignment \( \xi[[\xi]]\). In this sense, partial temporal assignments are a generalization of temporal assignments. The domain of an assignment \( \xi u \), denoted \( [[\xi u]] \), is defined to be the partial temporal element \( (\ell, u) \).

### 5.3. \( \theta \)-navigation

As we did in Chapter 3 for the complete temporal model, we want to introduce the construct \( [[A \theta B]] \) where \( A \) and \( B \) are attributes. In our model an assignment to an attribute is a partial temporal assignment of the type \( \xi u \) where \( \xi \) is a temporal assignment and \( \ell, u \) are temporal elements. Hence the constructs \( [[A \theta B]] \) may be introduced by defining \( [[(\xi_1 \ell_1 u_1) \theta (\xi_2 \ell_2 u_2)]] \).

The \( \theta \)-navigation expression \( [[(\xi_1 \ell_1 u_1) \theta (\xi_2 \ell_2 u_2)]] \) evaluates to the partial temporal element \( (\ell_1 \cap \ell_2, u_1 \cap u_2 - [[\xi_1 \theta \xi_2]]) \) where \( \theta' = \neg \theta \) (i.e. if \( \theta \) is \( \leq \) then \( \theta' \) is \( > \) etc.) The lower-limit, \( [[\xi_1 \theta \xi_2]] \cap \ell_2 \), is the time during which we are sure the \( \theta \)-relation holds. This lower limit cannot be greater than \( \ell_1 \cap \ell_2 \). The upper limit, \( u_1 \cap u_2 - [[\xi_1 \theta' \xi_2]] \), is the time beyond which the \( \theta \)-relation cannot exist.

A constant \( b \) can be identified with the assignment \( \xi u \) where \( \xi = [0, \text{NOW}]b \), \( \ell = [0, \text{NOW}] \) and \( u = [0, \text{NOW}] \). This allows us to introduce the construct \( [[A \theta b]] \) where \( A \) is an attribute and \( b \) is a constant.

**Example 5.1.** Let \( \xi_1 \ell_1 u_1 = ([0,5]a [6,9]b), \ell_1 = [0,10], u_1 = [0,20] \) and \( \xi_2 \ell_2 u_2 \) be \( ([0,4]a [5,8]c), \ell_2 = [0,9], u_2 = [0,15] \). Then \( [[(\xi_1 \ell_1 u_1) \theta (\xi_2 \ell_2 u_2)]] = ([0,4], [0,4]U [9,15]) \).

The definition of \( \theta \)-navigation as defined above is a generalization of the complete case. Thus, if we had complete information, we would get the same results as in the complete case. This can be shown as follows:

In the complete case, \( [[\xi_1]] = \ell_1 = u_1 \) and \( [[\xi_2]] = \ell_2 = u_2 \). Hence \( [[\xi_1 \theta \xi_2]] \cap \ell_2 = [[\xi_1 \theta \xi_2]] \). Similarly, \( u_1 \cap u_2 - [[\xi_1 \theta' \xi_2]] = [[\xi_1 \theta' \xi_2]] \cap \ell_2 \). Those instants in \( [[\xi_1 \cap \xi_2]] \) where \( \xi_1 \theta \xi_2 \) does not hold. This is the same as those instants in \( [[\xi_1 \ominus \xi_2]] \) where \( \xi_1 \theta \xi_2 \) holds. Hence \( u_1 \cap u_2 - [[\xi_1 \theta' \xi_2]] = [[\xi_1 \theta \xi_2]] \). Therefore, when we have
complete information, \[\langle [\xi_1 \ell_1 u_1] \ominus (\xi_2 \ell_2 u_2) \rangle\] evaluates to \[\langle [\xi_1 \ominus \xi_2] \rangle\]. This is equivalent to the temporal element \[\langle [\xi_1 \ominus \xi_2] \rangle\].

The following example shows that having \(\xi\) defined beyond \(\ell\) may help to reduce uncertainty in \([A \ominus B]\).

**Example 5.2.** In Example 5.1 \(\xi_1\) and \(\xi_2\) were not defined beyond \(\ell_1\) and \(\ell_2\) respectively. This example shows that having them defined beyond \(\ell_1\) and \(\ell_2\) may help to reduce uncertainty in \([A \ominus B]\).

Let \(\xi_1 = (a \in [0,5], b \in [11,15], \ell_1 = [0,10], u_1 = [0,20])\) and \(\xi_2 = (a \in [0,4], c \in [5,8], \ell_2 = [0,9], u_2 = [0,15])\).

Then \([\xi_1 \ominus \xi_2] = \langle [\xi_1 = \xi_2] \rangle\ominus (\ell_1 \cap \ell_2, u_1 \ominus u_2) = \langle [0,4] \cup [9,10] \rangle\). In Example 5.1 the result was \([0,4] \cup [9,15]\) which has greater uncertainty.

### 5.4. Tuples and Relations

A tuple \(\tau\) is a concatenation of partial temporal assignments whose \(\ell\) values are the same and \(u\) values are the same. Hence, in an actual implementation the common \(\ell\) value and \(u\) value could be stored at the tuple level rather than with each attribute. However, we will continue to have the temporal elements \(\ell\) and \(u\) associated with the attributes in order to simplify the formalism. The \(\ell\) values and \(u\) values have the following interpretation: During \(\ell\) we are sure the object represented by the tuple exists in the relation and beyond \(u\) the object cannot exist in the relation. The requirement that the \(\ell\) values of all attributes are equal and the \(u\) values of all attributes are equal makes the tuple *homogeneous*. This definition of homogeneity is analogous to the definition of homogeneity for the complete temporal case defined in Section 3.4. We denote the lower limit of the domains of the assignment in \(\tau\) by \(\tau.\ell\), and the upper limit by \(\tau.u\). The domain of a tuple \(\tau\), denoted \([\tau]\), is then defined as \(\langle \tau.\ell, \tau.u \rangle\). By \(\tau \mid \langle \ell,u \rangle\) is meant that each attribute in \(\tau\) is restricted to \(\langle \ell,u \rangle\) (the restriction of an attribute to a partial temporal element was defined in Section 5.2).

A relation \(r\) over a scheme \(R\) with key \(K \in CR\) is a set of tuples such that no key attribute values of a tuple change with time, for key attributes we have \([\xi] = u\), and no two tuples in \(r\) agree on all their key attributes.
Figure 5.2 shows a relation emp with NAME as the key. In each attribute $\xi \ell u$, the $\xi$ part is shown first, followed by the temporal element that represents the $\ell$ part and then the temporal element that represents the $u$ part. In the emp relation, we are sure that John was an employee at least during $[0,50]$ and that he was not an employee beyond $[0,100]$. However, we have missing information for his department during $[0,9]$. If he was present in the organization at any time during $[56,100]$, we have missing information on his department at that time also. If John was working for the organization during $[51,55]$, he was in the Shoes department. We also have some missing information for the SALARY attribute.

The domain of a relation $r$, denoted $\langle r \rangle$, is the union of the domains of its tuples. For example, $\langle \text{emp} \rangle = ([0,50],[0,100])$.

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,100]John</td>
<td>[10,40]30K</td>
<td>[10,30]Toys</td>
</tr>
<tr>
<td>[0,50]</td>
<td>[41,45]40K</td>
<td>[31,55]Shoes</td>
</tr>
<tr>
<td>[0,100]</td>
<td>[0,50]</td>
<td>[0,50]</td>
</tr>
<tr>
<td>[0,100]</td>
<td>[0,100]</td>
<td>[0,100]</td>
</tr>
<tr>
<td>[10,50]</td>
<td>[46,50]60K</td>
<td>[10,50]</td>
</tr>
<tr>
<td>[10,50]</td>
<td>[0,100]</td>
<td>[10,50]</td>
</tr>
</tbody>
</table>

Figure 5.2. The emp relation

5.5. Snapshots of Partial Temporal Relations

An instant $t$ can be represented as a partial temporal element $\langle [t,t],[t,t] \rangle$. A temporal snapshot of a relation at time $t$ is obtained by restricting each tuple in $r$ to $\langle [t,t],[t,t] \rangle$. The temporal snapshot may be represented as a static snapshot with nulls and an additional column called STATUS with domain = $\{d,m\}$ which denotes whether the tuple is definitely (d) in the relation or maybe (m) in the relation. Such a relation
corresponds to a classical relation with null values as in [Bi83]. The static snapshot at $t$ can be obtained from the temporal snapshot by placing a $d$ in the STATUS column for a tuple whose lower limits are $[t,t]$ or $m$ if the lower limits are $\emptyset$, by replacing empty assignments with null values and then deleting all timestamps. We denote a temporal or static snapshot by $r(t)$. Figure 5.3 shows the snapshot at $t = 50$ of the relation emp from Figure 5.2 as a temporal relation and as a static relation with nulls. Figure 5.4 shows the snapshot of emp at $t = 55$. 
<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>[50,50]John</td>
<td>[50,50]</td>
<td>[50,50]</td>
</tr>
<tr>
<td>[50,50]</td>
<td>[50,50]</td>
<td>[50,50]</td>
</tr>
<tr>
<td>[50,50]</td>
<td>[50,50]</td>
<td>[50,50]</td>
</tr>
<tr>
<td>[50,50]</td>
<td>[50,50]</td>
<td>[50,50]</td>
</tr>
<tr>
<td>[50,50]</td>
<td>[50,50]</td>
<td>[50,50]</td>
</tr>
</tbody>
</table>

Snapshot of emp at $t = 50$ as a temporal relation

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1</td>
<td>Shoes</td>
<td>d</td>
</tr>
<tr>
<td>Tom</td>
<td>60K</td>
<td>Toys</td>
<td>d</td>
</tr>
</tbody>
</table>

Snapshot of emp at $t = 50$ as a static relation with null values

Figure 5.3. Snapshot of emp relation at $t = 50$

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>¥</td>
<td>¥</td>
<td>¥</td>
</tr>
<tr>
<td>[55,55]</td>
<td>[55,55]</td>
<td>[55,55]</td>
</tr>
</tbody>
</table>

Snapshot of emp at $t = 55$ as a temporal relation

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>¥</td>
<td>Shoes</td>
<td>m</td>
</tr>
</tbody>
</table>

Snapshot of emp at $t = 55$ as a static relation with null values

Figure 5.4. Snapshot of emp relation at $t = 55$
6. ALGEBRA FOR PARTIAL TEMPORAL DATABASES

In this chapter we generalize our algebra of Chapter 2 to partial temporal relations. As before there are three kinds of algebraic expressions: partial temporal expressions, partial boolean expressions and partial relational expressions.

6.1. Partial Temporal Expressions

Partial temporal expressions are the syntactic counterparts of partial temporal elements and are formed from temporal elements, \([A\)], \([A\theta B]\), \([A\theta b]\), and \([E]\) where \(E\) is a relational expression to be defined in Section 6.3. More complex partial temporal expressions are formed using \(U\), \(\cap\) and \(\neg\).

The construct \([E]\) makes the algebra highly recursive since partial temporal expressions may contain relational expressions and as we shall see in Section 6.3.5 relational expressions can include partial temporal expressions in a very natural way. \([E]\) evaluates to the union of the temporal domains of the tuples in the relation for \(E\), i.e. \([E]\) = \(\cup_{\tau \in E}^{\tau_1, \ell, \tau_1 \in E^{\tau_1, u}}\).

Partial temporal expressions are syntactically the same as temporal expressions. However, the evaluation of these expressions on a tuple \(\tau\) yield a partial temporal element as follows:

- If \(\mu\) is a temporal element then \(\mu(\tau) = \langle \mu, \mu \rangle\).
- \([A](\tau) = \langle \ell, u \rangle\) where \(\tau(A) = \xi \ell u\) and \(A\) is an attribute.
- \([A\theta B](\tau) = \langle (\xi_1 \ell_1 u_1), (\xi_2 \ell_2 u_2) \rangle\) where \(\tau(A) = \xi_1 \ell_1 u_1\), \(\tau(B) = \xi_2 \ell_2 u_2\) and \(A\) and \(B\) are attributes (for our model \(\ell_1 = \ell_2, u_1 = u_2\) by homogeneity).
- \([A \theta b](\tau) = \langle (\xi_1 \ell_1 u_1), (\xi_2 \ell_2 u_2) \rangle\) where \(\tau(A) = \xi_1 \ell_1 u_1\), \(\xi_2 = [0, \text{NOW}]\), \(\ell_2 = [0, \text{NOW}]\) and \(u_2 = [0, \text{NOW}]\).
- \([E](\tau) = [E]\) where \(E\) is a relational expression.
\[ (t_1 \cap t_2)(\tau) = (t_1)(\tau) \cap (t_2)(\tau) \] where \( t_1 \) and \( t_2 \) are temporal expressions.

\[ (t_1 \cup t_2)(\tau) = (t_1)(\tau) \cup (t_2)(\tau) \] where \( t_1 \) and \( t_2 \) are temporal expressions.

\[ (t_1 - t_2)(\tau) = (t_1)(\tau) - (t_2)(\tau) \] where \( t_1 \) and \( t_2 \) are temporal expressions.

**Example 6.1.** Consider the `emp` relation in Figure 5.2. If John's tuple is denoted by \( \tau \) then \([\text{NAME}] (\tau) = ([0,50],[0,100])\). Similarly, \([\text{SALARY} = 30K](\tau) = ([10,40],[0,40] \cup [46,100])\) and \([\text{DEPT} = \text{Shoes}](\tau) = ([31,50],[0,9] \cup [31,100])\). Also, \( ([\text{SALARY} = 30K] \cup [\text{DEPT} = \text{Toys}]) (\tau) = ([10,40],[0,40] \cup [46,100])\).

### 6.2. Partial Boolean Expressions

Like the complete temporal case, atomic partial boolean expressions are formed using \( \text{TRUE}, \text{FALSE} \) and \( \mu \cap \nu \), where \( \mu \) and \( \nu \) are partial temporal expressions. More complex boolean expressions are formed using \( \lor, \land \) and \( \neg \).

Since we have incomplete information, we may not always be able to determine if a particular formula applied to a tuple yields \( \text{TRUE} \) or \( \text{FALSE} \). For instance, consider John's tuple in Figure 5.2. The atomic formula \([46,48] \subseteq [\text{SALARY} = 60K] \) may be \( \text{TRUE} \) or \( \text{FALSE} \) depending on the `SALARY` values during \([46,48]\). Hence we need to introduce the truth value \( \text{UNDEFINED} \). Then \([46,48] \subseteq [\text{SALARY} = 60K] \) yields the truth value \( \text{UNDEFINED} \) for John's tuple. The three-valued truth tables for \( \land, \lor \) and \( \neg \) are shown in Figure 6.1. We make an observation that \( \text{TRUE}, \text{FALSE}, \text{UNDEFINED}, \neg, \lor, \land \) are isomorphic to \( \langle I, I \rangle, \langle \emptyset, \emptyset \rangle, \langle \emptyset, I \rangle, \neg, \lor, \land \) where \( I = [0,\infty] \). This is clear from Figure 6.1 and Figure 6.2.

![Truth Tables](image)

Figure 6.1. Three-valued truth tables for \( \land, \lor, \neg \).

\( T = \text{TRUE}, F = \text{FALSE}, U = \text{UNDEFINED} \)
The evaluation of $\mu \land \nu$ for a given tuple $\tau$ is performed by first computing $\mu(\tau)$ and $\nu(\tau)$. Since $\mu$ and $\nu$ are partial temporal expressions, $\mu(\tau)$ and $\nu(\tau)$ yield partial temporal elements. Hence, we need to decide the truth value for one partial temporal element being a subset of another partial temporal element. In other words, we need to define the truth value of $\langle \ell_1, u_1 \rangle \subseteq \langle \ell_2, u_2 \rangle$. The partial temporal element $\langle \ell_1, u_1 \rangle$ means "at least $\ell_1$ and at most $u_1$." Similarly, the partial temporal element $\langle \ell_2, u_2 \rangle$ means "at least $\ell_2$ and at most $u_2$." Hence, $\langle \ell_1, u_1 \rangle \subseteq \langle \ell_2, u_2 \rangle$ is TRUE for sure if $u_1 \subseteq \ell_2$. Similarly, $\langle \ell_1, u_1 \rangle \subseteq \langle \ell_2, u_2 \rangle$ is FALSE for sure if $\ell_1 \nsubseteq u_2$. Otherwise, we say that $\langle \ell_1, u_1 \rangle \subseteq \langle \ell_2, u_2 \rangle$ is UNDEFINED. To further see the motivation for the definition consider the following example. Suppose from the given information in the database we can conclude that a certain condition holds at least during $\ell_1$ and at most during $u_1$. Suppose that we can also conclude that a second condition holds at least during $\ell_2$ and at most during $u_2$. We now want to ask if the second condition holds during the time the first condition holds. This would be TRUE for sure if $u_1 \subseteq \ell_2$ i.e. if the maximum possible time for the first condition is a subset of the minimum possible time for the second condition. Hence $\langle \ell_1, u_1 \rangle \subseteq \langle \ell_2, u_2 \rangle$ should be TRUE if $u_1 \subseteq \ell_2$. Similarly, $\langle \ell_1, u_1 \rangle \subseteq \langle \ell_2, u_2 \rangle$ should be FALSE if $\ell_1 \nsubseteq u_2$. Hence we get the following definition.

$$\langle \ell_1, u_1 \rangle \subseteq \langle \ell_2, u_2 \rangle = \begin{cases} \text{TRUE} & \text{if } u_1 \subseteq \ell_2 \\ \text{FALSE} & \text{if } \ell_1 \nsubseteq u_2 \\ \text{UNDEFINED} & \text{otherwise} \end{cases}$$

Now we introduce a function called eval, which takes a partial boolean expression and a tuple $\tau$, and returns one of \{ $\langle I, I \rangle$, $\langle \emptyset, \emptyset \rangle$, $\langle \emptyset, I \rangle$ \}. eval(f($\tau$)) is defined in such a way that eval(f($\tau$)) = $\langle I, I \rangle$ if and only if $f(\tau) = \text{TRUE}$, eval(f($\tau$)) = $\langle \emptyset, \emptyset \rangle$ if and only if $f(\tau) = \text{FALSE}$, and eval(f($\tau$)) = $\langle \emptyset, I \rangle$ if and only if $f(\tau) = \text{UNDEFINED}$. The main use of

<table>
<thead>
<tr>
<th>$\cap$</th>
<th>$\langle I, I \rangle$</th>
<th>$\emptyset$, $\emptyset$</th>
<th>$\emptyset$, $I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle I, I \rangle$</td>
<td>$\langle I, I \rangle$</td>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $I$</td>
</tr>
<tr>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $I$</td>
</tr>
<tr>
<td>$\emptyset$, $I$</td>
<td>$\emptyset$, $I$</td>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $I$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\cup$</th>
<th>$\langle I, I \rangle$</th>
<th>$\emptyset$, $\emptyset$</th>
<th>$\emptyset$, $I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle I, I \rangle$</td>
<td>$\langle I, I \rangle$</td>
<td>$\langle I, I \rangle$</td>
<td>$\langle I, I \rangle$</td>
</tr>
<tr>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $I$</td>
</tr>
<tr>
<td>$\emptyset$, $I$</td>
<td>$\emptyset$, $I$</td>
<td>$\emptyset$, $\emptyset$</td>
<td>$\emptyset$, $I$</td>
</tr>
</tbody>
</table>

Figure 6.2. Tables for $\cap$, $\cup$, $\neg$ of $\langle I, I \rangle$, $\emptyset$, $\emptyset$, $\emptyset$, $I$ where $I = [0, \text{NOW}]$.
eval is to simplify the definition of the selection operator and to simplify the statements of some of our results. The function eval allows the evaluation of a partial boolean expression using the operations $\lor$, $\land$ and $\neg$ for partial temporal elements. Formally, $\text{eval}(f)(\tau)$ is defined as follows:

- $\text{eval}(\text{TRUE})(\tau) = (I, I)$
- $\text{eval}(\text{FALSE})(\tau) = (\emptyset, \emptyset)$
- If $\mu$ and $\nu$ are partial temporal elements then
  $\text{eval}(\mu \& \nu)(\tau) = (I, I)$ if $\mu \& \nu$ is TRUE
  $ (\emptyset, \emptyset)$ if $\mu \& \nu$ is FALSE
  $ (\emptyset, I)$ if $\mu \& \nu$ is UNDEFINED.
- If $\mu$ and $\nu$ are more complex partial temporal expressions then
  $\text{eval}(\mu \& \nu)(\tau) = \text{eval}(\mu(\tau) \& \nu(\tau))(\tau)$.
- $\text{eval}(f_1 \lor f_2)(\tau) = \text{eval}(f_1)(\tau) \lor \text{eval}(f_2)(\tau)$
- $\text{eval}(f_1 \land f_2)(\tau) = \text{eval}(f_1)(\tau) \land \text{eval}(f_2)(\tau)$
- $\text{eval}(\neg f)(\tau) = \neg(\text{eval}(f)(\tau))$.

Example 6.2. If $\tau$ is John's tuple in Figure 5.2 then let us calculate $\text{eval}([46,48] \subseteq [\text{[SALARY}=60\text{K}]])(\tau)$. For John's tuple, $[\text{[SALARY}=60\text{K}]] = (\emptyset, [0,9]) \cup [46,100])$. Since $<[46,48],[46,48]> \subseteq (\emptyset, [0,9]) \cup [46,100])$ is UNDEFINED, $\text{eval}([46,48] \subseteq [\text{[SALARY}=60\text{K}]])(\tau) = (\emptyset, I)$ for John's tuple.

6.3. Relational Expressions

Relational expressions are the syntactic counterparts of partial temporal relations and are defined as follows.

6.3.1. Restructuring

Suppose $r$ is a relation over $R$ with key $K$. The snapshot of $r$ at $t$ was defined in Section 5.5. Two relations are said to be weakly equal if they have the same snapshot at each instant. If $K' \subseteq R$ such that $K' \rightarrow R$ in each snapshot, then $I_{K'_t}(r)$ is the
relation weakly equal to \( r \) but having \( K' \) as the key. We require that there is no missing value in the attributes in \( K' \) in the relation \( r \) otherwise it is not possible to do the restructuring (No missing value in \( \xi \) means \( [\xi] = u \)).

6.3.2. Union

Suppose \( r \) and \( s \) are relations with the same scheme and key. Then \( r \cup s \) also has the same scheme and key. To arrive at \( r \cup s \) we first compute the union of \( r \) and \( s \) treating them as sets, and then collapse each pair of tuples that agree on all key attributes into a single tuple. Hence the union is an objectwise union with the object being identified by the key values. Note that the collapsing could give an error if the two tuples being collapsed have different values at some non-key attributes at the same instant of time. Hence we need to assume that the union is being performed between compatible relations.

To formally define the union operation we first define \( \xi_1 \cup \xi_2 \cup g_1 \cup g_2 \), the union of two partial temporal assignments, to be \( \xi_1 \cup \xi_2 \cup g_1 \cup g_2 \). Suppose we are given partial temporal relations \( r \) and \( s \) with the same scheme \( R \), and the same key \( K \). Tuples \( r_1 \in r \) and \( r_2 \in s \) are said to be key-equivalent if they agree on all their key attributes. For key-equivalent tuples \( r_1 \) and \( r_2 \) their union \( r_1 \cup r_2 \) is defined attributewise. Thus, \( (r_1 \cup r_2)(A) = r_1(A) \cup r_2(A) \) for each attribute \( A \) in \( R \). We can now define \( r \cup s \) as follows:

\[
r \cup s = \{ \tau: \tau \in r \text{ and } \tau \text{ is not key-equivalent to any tuple in } s \} \cup \\
\{ \tau: \tau \in s \text{ and } \tau \text{ is not key-equivalent to any tuple in } r \} \cup \\
\{ \tau_1 \cup \tau_2: \tau_1 \in r \text{ and } \tau_2 \in s \text{ and } \tau_1, \tau_2 \text{ are key-equivalent} \}
\]

6.3.3. Difference

Suppose \( r \) and \( s \) are relations with the same scheme \( R \) and the same key \( K \). Then \( r - s \) has the same scheme and key. The relation \( r - s \) is computed as follows. We start with \( r \). For each tuple \( \tau_1 \) of \( r \) we check to see if there is a key-equivalent tuple in \( s \). If there is no such tuple in \( s \), then \( \tau_1 \) does not change. If \( s \) has such a tuple \( \tau_2 \), then let the lower limit of the assignments in \( \tau_1 \) and \( \tau_2 \) be \( l_1 \) and \( l_2 \) respectively, and the upper limits be \( u_1 \) and \( u_2 \) respectively. Now, at any instant in \( l_2 \), if for \( \tau_1 \) and \( \tau_2 \) the \( \xi \)'s are defined and agree on all attributes, then that instant is removed from the domain of \( \xi \) part of the assignments in \( \tau_1 \). That instant is also removed from \( l_1 \) and
u_1, the lower and upper limits of assignments in \( \tau_1 \). Also, those instants at which the \( \xi \)'s in \( \tau_1 \) and \( \tau_2 \) agree on all attributes or may have agreed on all attributes, if all the temporal assignments were completely defined, must be removed from \( \ell_1 \).

Consider the relation emp from Figure 5.2 and the relation emp' shown below in Figure 6.3. The result of emp - emp' is shown in Figure 6.4.

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>[41,120]John</td>
<td>[41,50]40K</td>
<td>[41,70]Shoes</td>
</tr>
<tr>
<td>[41,70]</td>
<td>[41,70]</td>
<td>[41,70]</td>
</tr>
<tr>
<td>[41,120]</td>
<td>[41,120]</td>
<td>[41,120]</td>
</tr>
</tbody>
</table>

Figure 6.3. The relation emp'

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,40]</td>
<td>[0,40]</td>
<td>[0,40]</td>
</tr>
<tr>
<td>[0,40]U[46,100]</td>
<td>[0,40]U[46,100]</td>
<td>[0,40]U[46,100]</td>
</tr>
<tr>
<td>[10,50]</td>
<td>[10,50]</td>
<td>[10,50]</td>
</tr>
<tr>
<td>[10,50]</td>
<td>[10,50]</td>
<td>[10,50]</td>
</tr>
</tbody>
</table>

Figure 6.4. Result of emp - emp'

6.3.4. Projection

The projection operation allows the user to choose certain columns of a relation. However, we require that all the attributes in the key of a relation must be projected. Thus, if \( r \) is a relation with scheme \( R \) and key \( K \) and if \( K \subseteq X \subseteq R \) then

\[
\Pi_X(r) = \{ \tau(X) : \tau \in r \}
\]
6.3.5. Selection

The selection operator is the most powerful operator in temporal databases. The selection operator has the form $\sigma(r; f; \mu)$ where $r$ is a relation, $f$ is a partial boolean expression and $\mu$ is a partial temporal expression. As in Section 3.6.3, the computation of the result of a selection operation cannot always be done by merely considering snapshots of relations at each instant of time. The operator uses the function eval defined in Section 6.2. In the incomplete information case, the expression $f$ when applied to a tuple $\tau$ yields TRUE, FALSE or UNDEFINED i.e. $\text{eval}(f)(\tau)$ yields $(1,1)$, $(0,0)$, or $(0,1)$. If $f(\tau)$ is FALSE (i.e. $\text{eval}(f)(\tau) = (0,0)$) we want to reject the tuple. If $f(\tau)$ is TRUE (i.e. $\text{eval}(f)(\tau) = (1,1)$) then we accept the tuple, but restrict it to $\mu(\tau)$ in the result because of the parameter $\mu$. If $f(\tau)$ is UNDEFINED (i.e. $\text{eval}(f)(\tau) = (0,1)$) we are not sure if the tuple $\tau$ should be in the result or not and so the lower limits of the assignment in the tuple must be set to $\emptyset$. This lower limit says that we are sure that the object must be present in the result only during $\emptyset$. In other words, it is possible that the object should never be there in the result relation. The tuple must be further restricted to $\mu(\tau)$. All of these requirements are captured by the following definition of $\sigma(r; f; \mu)$.

Let $r$ be a relation, $f$ be a partial boolean expression and $\mu$ be a temporal expression. Then $\sigma(r; f; \mu)$ is defined to be $\{\tau \in r : \text{eval}(f)(\tau) \cap \mu(\tau) \neq \emptyset\}$. The tuple $\tau \in r : \text{eval}(f)(\tau) \cap \mu(\tau) \neq \emptyset$ if the upper limit of the tuple is not $\emptyset$. In $\sigma(r; f; \mu)$, $f$ and $\mu$, when omitted, default to TRUE and $[0,$NOW$]$ respectively.

Example 6.3. Consider the query give all details of employees if they had a salary of 60K during $[46,48]$ applied to the.emp relation in Figure 5.2. This query can be expressed as $\sigma(\text{emp}; [46,48] \subseteq [\text{SALARY}=60K]; [0,\text{NOW}])$. For John's tuple $\text{eval}([46,48] \subseteq [\text{SALARY}=60K]) = (\emptyset,1)$, which is what we expect since we cannot determine if the condition is TRUE or FALSE with the given information. The result of the relational expression is shown in Figure 6.3. For John's tuple the $l$-values are $\emptyset$. This means that we are sure that John's tuple belongs in the relation only during $\emptyset$ i.e. John's tuple may not belong in the relation at all. However, we do carry the $\xi$ values forward in the result since John's tuple may belong in the relation.
Example 6.4. Consider the relation emp in Figure 5.2. Suppose we want to answer the following question: give details of employees if they had a salary of 60K during [46,48] but restrict the information to the time they were in the Toys department. This can be expressed as

$$
\sigma_{\text{emp; } [46,48][\text{SALARY}=60K]; \{\text{DEPT}=\text{Toys}\}}.
$$

For John's tuple, eval([46,48][\text{SALARY}=60K]) = (\emptyset, 1) as before, and [\text{DEPT}=\text{Toys}] = ([0,100],[0,100][56,100]). For Tom's tuple eval([46,48][\text{SALARY}=60K]) = (1,1) and
Example 6.5. Consider the partial temporal expressions in Example 6.1, but applied to the relation in Figure 6.5. If John's tuple is denoted by \( r \), then \( [[\text{name}]](r) = \langle \emptyset, [0,100] \rangle \). Similarly, \( [[\text{salary}=30\text{k}}]](r) = \langle \emptyset, [0,40] \cup [46,100] \rangle \) and \( [[\text{dept}=\text{shoes}}]](r) = \langle \emptyset, [0,9] \cup [31,100] \rangle \). Also, \( \langle [[\text{salary}=30\text{k}}] \cup [[\text{dept}=\text{toys}}] \rangle(r) = \langle \emptyset, [0,30] \cup [46,100] \rangle \). □

Example 6.6. Call the result obtained in Example 6.3 emp1. From that relation choose those employees that have worked in Toys during \([41,50]\). This query can be expressed as

\[
\sigma(\text{emp1}; [41,50] \cap [[\text{dept}=\text{toys}}])
\]

The result of the above expression is shown in Figure 6.7. Note that John's tuple does not appear in the result. This is what we expect because the condition \([41,50] \cap [[\text{dept}=\text{toys}}] is clearly FALSE even in \text{emp}. Since \text{emp1} has been obtained from \text{emp} as a result of a selection with an \( f \) parameter but no \( \mu \) parameter, we expect the condition to evaluate to \text{FALSE} in \text{emp1} as well. We would not have been able to get this if we had not carried forward the assignments \( \xi \)s from \text{emp} to \text{emp1}. □

6.3.6. Cross product

To compute \( r \times s \) when a tuple of \( r \) is concatenated with a tuple of \( s \), to preserve the homogeneity in the resultant tuple, we reduce its \( \ell \) value (\( u \) value resp.) to the intersection of the \( \ell \) values (\( u \) values resp.) of the tuples being concatenated. We also restrict the \( \xi \)'s to the new \( u \) value. The key of \( r \times s \) is the union of the keys of \( r \) and \( s \).
More formally, let $\tau_1$ be a tuple with lower limit $\ell_1$ and upper limit $u_1$ for each attribute. Let $\tau_2$ be a tuple with lower limit $\ell_2$ and upper limit $u_2$ for each attribute. Then we define $\text{hom}(\tau_1 \circ \tau_2)$ as follows. If $A = \xi \ell_1 u_1$ is an attribute of $\tau_1$ then $\text{hom}(\tau_1 \circ \tau_2)(A) = (\xi \ell_2 u_2)(\ell_1 \cap \ell_2)(u_1 \cap u_2)$. If $A = \xi \ell_2 u_2$ is an attribute of $\tau_2$ then $\text{hom}(\tau_1 \circ \tau_2)(A) = (\xi \ell_1 u_1)(\ell_1 \cap \ell_2)(u_1 \cap u_2)$. Then $r \times s = \{\text{hom}(\tau_1 \circ \tau_2)/\tau_1 \in r$ and $\tau_2 \in s\}.$

Note: The algebraic identities in Chapter 4 hold for the model for temporal databases with complete information. As shown in Example 6.7, these identities do not necessarily continue to hold for the model for temporal databases with incomplete information. However, if we our database had complete information, the identities would continue to hold even in we used the incomplete information model.

**Example 6.7.** Let $r(ABC)$ be a relation with key $A$ having the following single tuple: 
\[(0,5,10][0,10]; [0,5]; [0,10]; [0,5,10][0,10]).\]
Let $\mu_1$ be the temporal expression $[B=b]$ and $\mu_2$ be the temporal expressions $[A=a] - [C=c]$. Then $\sigma(r; \mu_1 \cap \mu_2)$ is the relation with the following single tuple: 
\[(6,10][a,\emptyset]; [6,10]; \emptyset; [0,10]).\]
However, $\sigma(\sigma(r; \mu_1); \mu_2)$ yields the relation with the following single tuple: 
\[(0,10][a,\emptyset]; [0,10]; [0,5][c,\emptyset; [0,10]).\]
Hence, the identity $\sigma(r; \mu_1 \cap \mu_2) = \sigma(\sigma(r; \mu_1); \mu_2)$ does not necessarily hold.

### 6.4. Generalization of the Complete Information Model

In this section, we state a result which says that our model for incomplete information is a generalization of the model for complete temporal information. A relation $r$ in the model for complete temporal information can be converted to an equivalent relation in the format of the incomplete information model by changing each temporal assignment $\xi$ to the partial assignment $\xi[[\xi]][\xi]$. The following theorem states that if all the relations had no missing information then a relational expression evaluated according to our model would give the same results as the complete information model described in Chapter 3.
THEOREM 6.1. Suppose \( \delta \) is a database in the complete information model, and \( E \) is a relational expression. If \( \delta' \) is the database in the incomplete information model obtained from \( \delta \) by replacing every attribute value \( \xi \) with \( \xi^{[\xi]} \), then \( E(\delta') \) can be obtained from \( E(\delta) \) by making a similar replacement.

PROOF:

As we showed in Section 5.1 a temporal element \( \mu \) can be represented as the partial temporal element \( \langle \mu, \mu \rangle \). With this identification the union, intersection, difference and complementation operations on partial temporal elements are a generalization of the corresponding operations temporal elements. In Section 5.3 we saw that that the \( \theta \)-navigation in the model for incomplete information is a generalization of the \( \theta \)-navigation in the model for complete information. Further, from Section 6.1 and Section 6.2 the evaluation of partial temporal expressions and partial boolean expressions is a generalization of the evaluation of temporal expressions and boolean expressions respectively. Hence, from the definitions in Section 6.3 it can be easily show that the evaluation of relation expressions in our model is a generalization of the evaluation of relation expressions in the complete model. \( \square \)
7. RELIABILITY OF ALGEBRAIC EXPRESSIONS

In Chapter 6 we stated a theorem that says that our model for incomplete temporal information is a generalization of the model for complete temporal information. In this chapter we prove results which show that our algebraic expressions give reliable results even when we have incomplete information. To do this we first introduce the notion of completions.

7.1. Preliminary Definitions and Notation

As in Section 5.4 we denote the lower and upper limits of a tuple τ by τ.l and τ.u respectively. τ.ξ(A) denotes the temporal assignment part of the assignment to A in τ. If μ is a partial temporal expression, μ(τ).l and μ(τ).u denote lower and upper limits of the resulting partial temporal element. If τ and τ' are tuples with the same scheme R and key K such that τ and τ' agree on all attributes in K, then τ and τ' are said to be key-equivalent.

7.2. Completions

A relation r in our model has correct but incomplete information about the objects it describes. Informally, a relation r' is a completion of a relation r if r' has complete information but is consistent with r. Thus, if we had complete information about the objects in r it is possible that we would have r' as our relation. Clearly, there could be many possible completions for a relation r. In fact, the more incomplete the information in r, the larger is the set of possible completions for r. On the other hand, the more complete the information in r, the smaller is the set of possible completions of r. If r had complete information then there would be just one completion of r, namely r itself.
7.2.1. Completions of partial temporal assignments and tuples

A partial temporal assignment describes an object which should be present in the relation at least during \( t \) and at most during \( u \). In "reality", the object would be present during some intermediate period \( u' \) such that \( l \subseteq u' \subseteq u \). Also, time instants during \( u' \) at which we do not have values for an attribute would actually have some value. This motivates the following definitions. An assignment \( \xi' u' u' \) is a completion of an assignment \( \xi t u \) if (i) \( l \subseteq u' \subseteq u \), (ii) \( \xi' \) agrees with \( \xi \) everywhere that both are defined, and (iii) \( [[\xi']] = u' \).

Example 7.1. Consider the assignment to NAME for John's tuple in Figure 5.2. In that assignment we had \( \xi = [0,100] \) John, \( l = [0,50] \) and \( u = [0,100] \). Thus John must be in the relation at least during \([0,50]\) and at most during \([0,100]\). An assignment \( \xi' u' u' \) such that \( \xi' = [0,60] \) John, \( u' = [0,60] \) is a completion of the assignment \( \xi t u \).

Now consider the assignment to SALARY for John's tuple in Figure 5.2. In that assignment we had \( \xi = ([0,40] 30K, [41,45] 40K) \), \( l = [0,50] \) and \( u = [0,100] \). An assignment \( \xi' u' u' \) such that \( \xi' = ([0,40] 30K, [41,50] 40K, [51,60] 50K) \), \( u' = [0,60] \) is a completion of the assignment \( \xi t u \). •

We define a tuple \( \tau' \) over \( R \) to be a completion of a tuple \( \tau \) over \( R \) if for all attributes \( A \in R \), \( \tau'(A) \) is a completion of \( \tau(A) \). Thus, if \( \tau \) and \( \tau' \) are tuples over \( R \), then \( \tau' \) is a completion of \( \tau \) if (i) \( \tau. l \subseteq \tau'. l = \tau'. u \subseteq \tau. u \), (ii) for all \( A \in R \), \( \tau'. \xi(A) \) and \( \tau. \xi(A) \) agree everywhere both of them are defined, and (iii) for all \( A \in R \), \( \tau'. \xi(A) \) is defined everywhere along \( \tau'. u \).

7.2.2 Completions of relations and databases

A relation \( r' \) is a completion of a relation \( r \) if (i) given a \( \tau \in r \) with \( \tau. l \neq \emptyset \) there is a \( \tau' \in r' \) such that \( \tau' \) is a completion of \( \tau \), and (ii) given a \( \tau' \in r' \) there is a \( \tau \in r \) such that \( \tau' \) is a completion of \( \tau \). A completion of a database is a natural extension.

In the definition, \( \tau \) and \( \tau' \) must be key-equivalent since \( \tau' \) is a completion of \( \tau \). Because a relation has at most one tuple with a given key, the phrase "there is a" may be replaced by "there is exactly one" in conditions (i) and (ii) of the definition of completions of relations.

Example 7.2. The relation shown in Figure 7.1 is one of the possible completions of the relation emp shown in Figure 5.2. •
7.2.3 Completions in terms of snapshots

We now describe what a completion of a relation means in terms of snapshots. Consider a tuple \( r \in r(t) \), the snapshot of \( r \) at \( t \). We say that \( r' \) is a snapshot completion of \( r \) if (i) \( \tau'.\ell = \tau'.u = [t,t] \), (ii) for each attribute \( A \), if \( \tau.\xi(A) \) is not empty then \( \tau'.\xi(A) = \tau.\xi(A) \), and (iii) for no attribute \( A \) is \( \tau'.\xi(A) \) empty. A snapshot relation \( r'(t) \) is a snapshot completion of \( r(t) \) if (i) given \( r(t) \) such that \( r.\ell = [t,t] \) there is a tuple \( r' \in r'(t) \) such that \( r' \) is a snapshot completion of \( r \) and (ii) given a \( r' \in r'(t) \) there is a tuple \( r \in r(t) \) such that \( r' \) is a snapshot completion of \( r \).

We are now in a position to define completions in terms of snapshots. A relation \( r' \) is a completion of \( r \) if \( r'(t) \) is a snapshot completion of \( r(t) \) for all \( t \in [0,\text{NOW}] \).

7.3. Results when Construct \([E]\) is Omitted

In this section we state some results for the model. We first remove the construct \([E]\) from consideration. In Section 7.4 we will reintroduce the construct. The results with the construct included are very similar to those without the construct. However, we still state the results separately for two reasons. First, when \([E]\) is included there is no clear separation between the lemmas stated below and Proposition 7.1. The fact that the lemmas can be proved independent of the proposition when \([E]\) is omitted but
not when \([\mathbb{E}]\) is included is interesting, and is made clear by our presentation. Second, there is an additional notational complexity when \([\mathbb{E}]\) is included. These points will be made clearer in Section 7.4 where we reintroduce \([\mathbb{E}]\) into our analysis.

**Lemma 7.1.** Let \(\tau, \tau'\) be tuples over \(\mathbb{R}\) such that \(\tau'\) is a completion of \(\tau\). Let \(\mu\) be a partial temporal expression not involving the construct \([\mathbb{E}]\). Then \(\mu(\tau).l \subseteq \mu(\tau').l\) and \(\mu(\tau).u \supseteq \mu(\tau').u\).

**Proof:**

The proof is by induction on the complexity of \(\mu\).

- Let \(\tau\) have lower limit \(l\) and upper limit \(u\).
- Let \(\tau'\) have lower and upper limit \(u'\).

Hence, since \(\tau'\) is a completion of \(\tau\) we have

1. \(l \subseteq u' \subseteq u\)
2. For all \(A \in \mathbb{R}\), \(\tau', \xi(A)\) is defined everywhere along \(u'\)
3. For all \(A \in \mathbb{R}\), \(\tau', \xi(A)\) and \(\tau, \xi(A)\) agree everywhere both of them are defined.

**Basis:**

- **Case A.** \(\mu\) is a temporal element.
  
  Then \(\mu(\tau).l = \mu(\tau').l = \mu\). Thus, \(\mu(\tau).l \subseteq \mu(\tau').l\) (in fact, equal).
  
  Similarly, \(\mu(\tau).u = \mu(\tau').u = \mu\). Thus \(\mu(\tau).u \supseteq \mu(\tau').u\) (in fact, equal)

- **Case B.** \(\mu = [\mathbb{A}]\), where \(A\) is an attribute.
  
  \(\mu(\tau).l = [\mathbb{A}](\tau).l = l\)
  
  \(\mu(\tau').l = [\mathbb{A}](\tau').l = u'\)
  
  From condition (i), \(l \subseteq u'\)
  
  Hence \(\mu(\tau).l \subseteq \mu(\tau').l\)
  
  Similarly,
  
  \(\mu(\tau).u = [\mathbb{A}](\tau).u = u\)
  
  \(\mu(\tau').u = [\mathbb{A}](\tau').u = u'\)
  
  From condition (i), \(u \supseteq u'\)
  
  Hence \(\mu(\tau).u \supseteq \mu(\tau').u\)

- **Case C.** \(\mu = [\mathbb{A} \theta \mathbb{B}]\), where \(A, B\) are attributes.
  
  \([\mathbb{A} \theta \mathbb{B}](\tau).l = [\tau, \xi(A) \theta \tau, \xi(B)]\mathbb{A} \cap \mathbb{B}\)
Consider a time instant \( t \) in \( \ell \). If at instant \( t \) we have \( \tau \cdot \xi(A)(t) \cup \tau \cdot \xi(B)(t) \), then we also have \( \tau' \cdot \xi(A)(t) \cup \tau' \cdot \xi(B)(t) \) (conditions i, ii and iii)

Hence, \([A \cap B](\tau) \cdot \ell \subseteq [A \cap B](\tau') \cdot \ell\)

Also,

\([A \cap B](\tau) \cdot u = u \cap u' - [\tau \cdot \xi(A) \cup \tau \cdot \xi(B)]\), where \( \theta' \) is a complement of \( \theta \).

Thus \([A \cap B](\tau) \cdot u \subseteq u' \cap u' - [\tau \cdot \xi(A) \cup \tau \cdot \xi(B)]\) (\( u \supseteq u' \) (1)

Now \([A \cap B](\tau') \cdot u = u' \cap u' - [\tau' \cdot \xi(A) \cup \tau' \cdot \xi(B)]\) (2)

Consider time instants \( t \) in \( u' \). If at instant \( t \) we have \( \tau \cdot \xi(A)(t) \cup \tau \cdot \xi(B)(t) \), then we also have \( \tau' \cdot \xi(A)(t) \cup \tau' \cdot \xi(B)(t) \)

Hence, from (1) and (2), \([A \cap B](\tau) \cdot u \subseteq [A \cap B](\tau') \cdot u\)

Case D. \( \mu = [A \cap b] \), where \( A \) is an attribute and \( b \) is a constant.

The proof in this case is similar to the proof of case C.

Induction Step.

Case A. \( \mu = \mu_1 \cap \mu_2 \)

\( \mu(\tau) \cdot \ell = \mu_1(\tau) \cdot \ell \cap \mu_2(\tau) \cdot \ell \)

\( \subseteq \mu_1(\tau') \cdot \ell \cap \mu_2(\tau') \cdot \ell \) (Using Induction Hypothesis)

\( = \mu(\tau') \cdot \ell \)

Hence \( \mu(\tau) \cdot \ell \subseteq \mu(\tau') \cdot \ell \)

Similarly,

\( \mu(\tau) \cdot u = \mu_1(\tau) \cdot u \cap \mu_2(\tau) \cdot u \)

\( \supseteq \mu_1(\tau') \cdot u \cap \mu_2(\tau') \cdot u \) (Using Induction Hypothesis)

\( = \mu(\tau') \cdot u \)

Hence \( \mu(\tau) \cdot u \supseteq \mu(\tau') \cdot u \)

Case B. \( \mu = \mu_1 \cup \mu_2 \)

\( \mu(\tau) \cdot \ell = \mu_1(\tau) \cdot \ell \cup \mu_2(\tau) \cdot \ell \)

\( \subseteq \mu_1(\tau') \cdot \ell \cup \mu_2(\tau') \cdot \ell \) (Using Induction Hypothesis)

\( = \mu(\tau') \cdot \ell \)

Hence \( \mu(\tau) \cdot \ell \subseteq \mu(\tau') \cdot \ell \)

Similarly, we can show that \( \mu(\tau) \cdot u \supseteq \mu(\tau') \cdot u \)
Case C. \( \mu = \mu_1 - \mu_2 \)
\[
\mu(\tau).l = \mu_1(\tau).l - \mu_2(\tau).u
\]
\( \xi \mu_1(\tau').l - \mu_2(\tau').u \) (By Induction Hyp, \( \mu_1(\tau).l \xi \mu_1(\tau').l \) and \( \mu_2(\tau).u \notin \mu_2(\tau').u \)
\[
= \mu(\tau').l
\]
Hence \( \mu(\tau).l \xi \mu(\tau').l \)

Similarly,
\[
\mu(\tau).u = \mu_1(\tau).u - \mu_2(\tau).l
\]
\( \notin \mu_1(\tau').u - \mu_2(\tau').l \) (By Induction Hyp, \( \mu_2(\tau).l \xi \mu_2(\tau').l \) and \( \mu_1(\tau).u \notin \mu_1(\tau').u \)
\[
= \mu(\tau').u
\]
Hence \( \mu(\tau).u \notin \mu(\tau').u \)

Hence, the result.

\[\square\]

**Lemma 7.2.** Let \( \tau, \tau' \) be tuples over \( R \) such that \( \tau' \) is a completion of \( \tau \). Let \( f \) be a partial boolean expression not involving the construct \([E]\). Then

1. if eval\((f)(\tau') = \langle 1,1 \rangle \) then eval\((f)(\tau) = \langle 1,1 \rangle \) or \( \langle 0,1 \rangle \)
2. if eval\((f)(\tau') = \langle 0,0 \rangle \) then eval\((f)(\tau) = \langle 0,0 \rangle \) or \( \langle 0,1 \rangle \)
3. if eval\((f)(\tau) = \langle 1,1 \rangle \) then eval\((f)(\tau') = \langle 1,1 \rangle \)
4. if eval\((f)(\tau) = \langle 0,0 \rangle \) then eval\((f)(\tau') = \langle 0,0 \rangle \)

**Proof:**

We first prove both 1 and 2 hand in hand.

The proof is by induction on complexity of \( f \).

**Basis.** \( f \) is \( \mu_1 \xi \mu_2 \), where \( \mu_1, \mu_2 \) are temporal expressions.

1. Let eval\((f)(\tau') = \langle 1,1 \rangle \)

   By definition of eval\((f)(\tau')\),
   \[
   \mu_1(\tau').u \xi \mu_2(\tau').l
   \]
   Assume eval\((f)(\tau) = \langle 0,0 \rangle \)

   Then \( \mu_1(\tau).l \notin \mu_2(\tau).u \) (By defn. of eval)

   Thus \( \mu_1(\tau').l \notin \mu_2(\tau').u \) (Using Lemma 7.1, making LHS larger and RHS smaller)
Thus $\mu_1(r').u \not\subseteq \mu_2(r').\ell$ (making LHS larger, RHS smaller)
But this contradicts statement A.
Thus, the assumption that eval(f)(\tau) = \langle \emptyset, \emptyset \rangle does not hold
Hence, eval(\mu_1 \subseteq \mu_2)(\tau) = \langle 1, 1 \rangle \text{ or } \langle 0, 1 \rangle

2. Let eval(f)(\tau') = \langle \emptyset, \emptyset \rangle
By definition of eval(f)(\tau'),
\[ \mu_1(\tau').\ell \notin \mu_2(\tau').u \quad \text{(B)} \]
Assume eval(f)(\tau) = \langle 1, 1 \rangle
Then $\mu_1(\tau).u \subseteq \mu_2(\tau).\ell$ (By defn. of eval)
Thus $\mu_1(\tau').u \subseteq \mu_2(\tau').\ell$ (Using Lemma 7.1, making RHS larger and LHS smaller)
Thus $\mu_1(\tau').\ell \not\subseteq \mu_2(\tau').u$ (making RHS larger, LHS smaller)
But this contradicts statement B.
Thus, the assumption that eval(f)(\tau) = \langle 1, 1 \rangle does not hold
Hence, eval(\mu_1 \subseteq \mu_2)(\tau) = \langle 0, 0 \rangle \text{ or } \langle 0, 1 \rangle

Induction Step
Assume 1 and 2 hold for formulas smaller than f
Case A. $f = f_1 \land f_2$

1. eval(f)(\tau') = \langle 1, 1 \rangle
   Then eval(f_1)(\tau') = \langle 1, 1 \rangle \text{ and eval(f_2)(\tau') = \langle 1, 1 \rangle}
   By induction hypothesis,
   eval(f_1)(\tau) = \langle 1, 1 \rangle \text{ or } \langle 0, 1 \rangle, \text{ and }
   eval(f_2)(\tau) = \langle 1, 1 \rangle \text{ or } \langle 0, 1 \rangle
   Thus eval(f_1)(\tau) \text{ or eval(f_2)(\tau) = \langle 1, 1 \rangle or } \langle 0, 1 \rangle
   Hence eval(f_1 \land f_2)(\tau) = \langle 1, 1 \rangle \text{ or } \langle 0, 1 \rangle
   Hence eval(f)(\tau) = \langle 1, 1 \rangle \text{ or } \langle 0, 1 \rangle

2. eval(f)(\tau') = \langle 0, 0 \rangle
   Then eval(f_1)(\tau') = \langle 0, 0 \rangle \text{ or eval(f_2)(\tau') = \langle 0, 0 \rangle}
   Without loss of generality,
let eval(f_1)(τ') = ⟨∅,∅⟩
Then by induction hypothesis,
eval(f_1)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩
Thus eval(f_1)(τ) ∨ eval(f_2)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩
Hence eval(f_1 ∧ f_2)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩
Hence eval(f)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩

Case B. f = f_1 ∨ f_2
1. eval(f)(τ') = ⟨I,I⟩
   Then eval(f_1)(τ') = ⟨I,I⟩ or eval(f_2)(τ') = ⟨I,I⟩
   Without loss of generality,
   let eval(f_1)(τ') = ⟨I,I⟩
   By induction hypothesis,
   eval(f_1)(τ) = ⟨I,I⟩ or ⟨∅,I⟩
   Thus eval(f_1)(τ) ∨ eval(f_2)(τ) = ⟨I,I⟩ or ⟨∅,I⟩
   Hence eval(f_1 ∨ f_2)(τ) = ⟨I,I⟩ or ⟨∅,I⟩
   Hence eval(f)(τ) = ⟨I,I⟩ or ⟨∅,I⟩

2. eval(f)(τ') = ⟨∅,∅⟩
   Then eval(f_1)(τ') = ⟨∅,∅⟩ and eval(f_2)(τ') = ⟨∅,∅⟩
   By induction hypothesis,
   eval(f_1)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩, and
   eval(f_2)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩
   Thus eval(f_1)(τ) ∨ eval(f_2)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩
   Hence eval(f_1 ∨ f_2)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩
   Hence eval(f)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩

Case C. f = ¬f_1
1. eval(f)(τ') = ⟨I,I⟩
   Then eval(f_1)(τ') = ⟨∅,∅⟩
   By induction hypothesis,
   eval(f_1)(τ) = ⟨∅,∅⟩ or ⟨∅,I⟩
Hence \( \neg \text{eval}(f)(\tau) = \langle I,I \rangle \) or \( \langle \emptyset,I \rangle \)
Hence \( \text{eval}(f)(\tau) = \langle I,I \rangle \) or \( \langle \emptyset,I \rangle \)

2. \( \text{eval}(f)(\tau') = \langle \emptyset,\emptyset \rangle \)
Then \( \text{eval}(f_1)(\tau') = \langle I,I \rangle \)
By induction hypothesis,
\[ \text{eval}(f_1)(\tau) = \langle I,I \rangle \) or \( \langle \emptyset,I \rangle \)
Hence \( \neg \text{eval}(f_1)(\tau) = \langle \emptyset,\emptyset \rangle \) or \( \langle \emptyset,I \rangle \)
Hence \( \text{eval}(f)(\tau) = \langle \emptyset,\emptyset \rangle \) or \( \langle \emptyset,I \rangle \)
This proves statements 1 and 2.

3. We are given that \( \text{eval}(f)(\tau) = \langle I,I \rangle \)
Since \( \tau' \) is a complete tuple, \( \text{eval}(f)(\tau') = \langle I,I \rangle \) or \( \langle \emptyset,\emptyset \rangle \)
Assume \( \text{eval}(f)(\tau') = \langle \emptyset,\emptyset \rangle \)
Then by statement 2 of the lemma, \( \text{eval}(f)(\tau) = \langle \emptyset,\emptyset \rangle \) or \( \langle \emptyset,I \rangle \)
But this contradicts the condition that \( \text{eval}(f)(\tau) = \langle I,I \rangle \)
The assumption that \( \text{eval}(f)(\tau') = \langle \emptyset,\emptyset \rangle \) is not valid.
Hence, \( \text{eval}(f)(\tau') = \langle I, I \rangle \)

4. We are given that \( \text{eval}(f)(\tau) = \langle \emptyset,\emptyset \rangle \)
Since \( \tau' \) is a complete tuple, \( \text{eval}(f)(\tau') = \langle I,I \rangle \) or \( \langle \emptyset,\emptyset \rangle \)
Assume \( \text{eval}(f)(\tau') = \langle I,I \rangle \)
Then by statement 1 of the lemma, \( \text{eval}(f)(\tau) = \langle I,I \rangle \) or \( \langle \emptyset,I \rangle \)
But this contradicts the condition that \( \text{eval}(f)(\tau) = \langle \emptyset,\emptyset \rangle \)
The assumption that \( \text{eval}(f)(\tau') = \langle I,I \rangle \) is not valid.
Hence, \( \text{eval}(f)(\tau') = \langle \emptyset,\emptyset \rangle \)

Part 1 of Lemma 7.2 shows that if \( \tau' \) is a completion of \( \tau \) (and hence \( \tau' \) is consistent with \( \tau \) but has complete information) and if \( f(\tau') \) evaluates to \( \text{TRUE} \) then in our model \( f(\tau) \) will not evaluate to \( \text{FALSE} \) in spite of \( \tau \) having incomplete information. Similarly, part 2 of the above Lemma states that if \( f(\tau') \) evaluates to \( \text{FALSE} \) then we will not evaluate \( f(\tau) \) to be \( \text{TRUE} \) in spite of incomplete information. Part 3 of the Lemma states that if we evaluate \( f(\tau) \) to be \( \text{TRUE} \) then \( f(\tau') \) is \( \text{TRUE} \) for every tuple \( \tau' \).
which is a completion of \( \tau \). Similarly, according to part 4 of the Lemma, if we evaluate \( f(\tau) \) to be \text{FALSE} then \( f(\tau') \) is False for every tuple \( \tau' \) which is a completion of \( \tau \).

**Lemma 7.3.** Let \( \tau, \tau' \) be tuples over \( R \) such that \( \tau' \) is a completion of \( \tau \). Let \( f \) be a boolean expression not involving the construct \([E]\). Then

1. \( \text{eval}(f)(\tau).l \subseteq \text{eval}(f)(\tau').l \)
2. \( \text{eval}(f)(\tau).u \supseteq \text{eval}(f)(\tau').u \)

**Proof:** Immediate from statements 1 and 2 of Lemma 7.2 and the fact that \( \text{eval}(f)(\tau') \) is either \( \{0,0\} \) or \( \{1,1\} \). □

We now state a theorem which shows that the results of our algebraic expressions are reliable. Suppose \( \delta \) is the state of a database having incomplete information. Then a completion of \( \delta \) (say \( \delta' \)) has complete information and is consistent with the information in \( \delta \). Hence, \( \delta' \) is a possibility for \( \delta \) in the "real" world. Since our model generalizes the complete information model we know that when there is complete information our model gives correct information. Hence, \( E(\delta') \) should be a possibility for the result when \( E \) is applied to \( \delta \). In other words if \( E(\delta') \) is a completion of \( E(\delta) \) we may say our result \( E(\delta) \) is reliable because then \( E(\delta) \) does allow \( E(\delta') \) as a possibility with complete information.

**Proposition 7.1.** Let \( \delta \) be the state of a database and let \( E \) be an algebraic expression not involving a construct such as \([E']\). Then for any completion \( \delta' \) of the database \( \delta \), \( E(\delta') \) is a completion of \( E(\delta) \).

**Proof**

The proof is by induction on complexity of \( E \). However, to simplify the proof, in the induction step we consider expressions like \( r \cup s \) etc., instead of \( E_1 \cup E_2 \) etc.

**Basis:**

If \( E \) has zero operators then clearly, \( E(\delta') (=\delta') \) is a completion of \( E(\delta)(=\delta) \).

**Induction Step:**

**Case A.** \( E = r \cup s \)

Let \( r', s' \) be completions of \( r, s \) respectively.

Let \( \tau_1 \in r \) and \( \tau_2 \in s \) be key-equivalent tuples in \( r \) and \( s \) respectively.

(The case where \( \tau_1 \in r \) and there is no \( \tau_2 \in s \) which is key equivalent to \( \tau_1 \) is obvious)
Let \( r' \in r' \) and \( r'' \in s' \) be completions of \( r_1, r_2 \) respectively.

Let \( A \) be any arbitrary attribute in R (or S).

Let \( \tau_1(A) = \xi_{11} \ell_{11} u_1 \) and \( \tau_2(A) = \xi_{12} \ell_{12} u_2 \)

Then \( \tau_1(A) = \xi_{11}' u_1' u_1' \) and \( \tau_2(A) = \xi_{22}' u_2' u_2' \) for some \( u_1', u_2' \) such that

i) \( \ell_{11} \subseteq u_1' \subseteq u_1 \) and \( \ell_{12} \subseteq u_2' \subseteq u_2 \).

ii) \( [\xi_{11}'] = u_1' \) and \( [\xi_{12}'] = u_2' \), and

iii) \( \xi_{11}', \xi_{12}' \) agree everywhere both are defined and

\( \xi_{21}', \xi_{22}' \) agree everywhere both are defined.

Now these produce key-equivalent tuples \( \tau, \tau' \) in \( rUs, r'Us' \) respectively such that

\[
\tau(A) = \tau_1(A) \cup \tau_2(A) = \xi_{11}' \cup \xi_{12}' \cup \ell_{11} \cup u_1 \cup \ell_{12} \cup u_2
\]

\[
\tau'(A) = \tau_1'(A) \cup \tau_2'(A) = \xi_{11}' \cup \xi_{12}' \cup u_1' \cup u_2'
\]

Now, from (i) we have

\[
\ell_{11} \cup \ell_{12} \subseteq u_1' \cup u_2' \subseteq u_1 \cup u_2
\] (1)

From (ii) we have

\[
[\xi_{11}' \cup \xi_{12}'] = u_1' \cup u_2' \] (2)

From (iii), \( \xi_{11}' \cup \xi_{12}' \) and \( \xi_{11}' \cup \xi_{12} ' \) agree everywhere both are defined.

Hence, from 1, 2 and 3, \( \tau'(A) \) is a completion of \( \tau(A) \)

Since \( A \) is an arbitrary attribute,

\( \tau' \) is a completion of \( \tau \).

Hence, \( r' \cup s' \) is a completion of \( r \cup s \).

Case B. \( E = r - s \)

Let \( r', s' \) be completions of \( r, s \) respectively.

Let \( \tau_1 \in r \) and \( \tau_2 \in s \) be key equivalent tuples in \( r \) and \( s \) respectively.

(The case where \( \tau_1 \in r \) and there is no \( \tau_2 \in s \) which is key-equivalent to \( \tau_1 \) is obvious)

Let \( \tau_1' \in r' \) and \( \tau_2' \in s' \) be completions of \( r_1, r_2 \) respectively.

Let \( A \) be any arbitrary attribute in R (or S).

Let \( \tau_1(A) = \xi_{11} \ell_{11} u_1 \) and \( \tau_2(A) = \xi_{12} \ell_{12} u_2 \)

Then \( \tau_1(A) = \xi_{11}' u_1' u_1' \) and \( \tau_2(A) = \xi_{22}' u_2' u_2' \) for some \( u_1', u_2' \) such that

i) \( \ell_{11} \subseteq u_1' \subseteq u_1 \) and \( \ell_{12} \subseteq u_2' \subseteq u_2 \).
ii) \([\xi_1] = u_1^*\) and \([\xi_2] = u_2^*\), and
iii) \(\xi_1, \xi_1^*\) agree everywhere both are defined and 
\(\xi_2, \xi_2^*\) agree everywhere both are defined.

Let \(\tau, \tau'\) be the resultant tuples in \(r-s, r'-s'\) respectively.

Suppose \(\tau_1, \tau_2\) agree at \(\mu_1\) and could agree upto \(\mu_2\) if they were complete and that \(\tau_1', \tau_2'\) agree at \(\mu'\) (and therefore could agree upto \(\mu'\)).

Clearly, \(\mu_1 \subseteq \mu' \subseteq \mu_2\)

The lower limit of \(\tau\) i.e. \(\tau.l = \tau_1.l - \mu_2\) and upper limit \(\tau.u = \tau_1.u - \mu_1\).

Also, the lower and upper limit of \(\tau', \tau'.l = \tau'.u = \tau_1.u - \mu'\)

Now since \(\tau_1.l \subseteq \tau_1'.l\) and \(\mu' \subseteq \mu_2\),
\[\tau_1.l - \mu_2 \subseteq \tau_1'.l - \mu'\]
Hence \(\tau.l \subseteq \tau'.l\) (A)

Also since \(\tau_1.u \supseteq \tau_1'.u\) and \(\mu' \supseteq \mu_1\),
\[\tau_1.u - \mu_1 \supseteq \tau_1'.u - \mu'\]
Hence \(\tau.u \supseteq \tau'.u\) (B)

From A and B
\[\tau.l \subseteq \tau'.l = \tau_1'.l \subseteq \tau.u\] (1)

Now, \(\tau_1.\xi(A)\) and \(\tau_1'.\xi(A)\) agree everywhere both are defined
Hence, \((\tau_1.\xi(A))\tau(\tau.u)\) and \((\tau_1'.\xi(A))\tau(\tau'.u)\) agree everywhere both are defined
i.e. \(\tau.\xi(A)\) and \(\tau'.\xi(A)\) agree everywhere both are defined. (2)

Now, \(\tau_1'.\xi(A)\) is defined everywhere along \(\tau_1'.u\). Every instant removed from \(\tau_1'.\xi(A)\) to get \(\tau'.\xi(A)\) is also removed from \(\tau_1'.u\) to get \(\tau'.u\).
Hence, \(\tau'.\xi(A)\) is defined everywhere along \(\tau'.u\) (3)

From 1, 2 and 3
\(\tau'(A)\) is a completion of \(\tau(A)\).
Hence \(\tau'\) is a completion of \(\tau\).
i.e. \(r' - s'\) is a completion of \(r - s\)

Case C. \(E = \pi_X(r)\)

Let \(r'\) be a completion of \(r\).

Let \(\tau' \in r'\) and \(\tau \in r\) be such that \(\tau'\) is a completion of \(\tau\).

Then, clearly \(\tau'(X)\) is a completion of \(\tau(X)\).
Hence, \(\pi_X(r')\) is a completion of \(\pi_X(r)\).
Case D. \( E = r \times s \)

Let \( r' \), \( s' \) be completions of \( r, s \) respectively.

Let \( r_1' \in r' \) and \( r_2' \in s' \) be completions of \( r_1 \in r, r_2 \in s \) respectively.

Let \( A \) be any arbitrary attribute in \( R \).

Then from the definition of cross-product in Section 6.3.6,

\[
\text{hom}(r_1 \circ r_2)(A) = ((r_1 \cdot A) \cap (r_1 \cdot u)) \cap (r_2 \cdot u),
\]

and \( \text{hom}(r_1' \circ r_2')(A) = ((r_1' \cdot A) \cap (r_1' \cdot u)) \cap (r_2' \cdot u). \)

Now, since \( r_1', r_2' \) are completions of \( r_1, r_2 \) respectively,

\[
(r_1 \cdot A) \cap (r_1 \cdot u) \subseteq (r_1' \cdot A) \cap (r_1' \cdot u) \subseteq (r_2' \cdot u) \cap (r_2' \cdot u).
\]

Hence,

\[
(t_1 \cdot A) \cap (t_2 \cdot u) \subseteq (t_1' \cdot A) \cap (t_1' \cdot u) \subseteq (t_2' \cdot u) \cap (t_2' \cdot u). \tag{1}
\]

Also, since \( t_1' \) is a completion of \( t_1 \),

\( t_1', t_2' \) agree everywhere both are defined.

Hence

\[
(t_1' \cdot A) \cap (t_2' \cdot u) \text{ agree everywhere both are defined.} \tag{2}
\]

Further, \( (t_1' \cdot A) \cap (t_2' \cdot u) \) is defined everywhere along \( (t_1' \cdot u) \cap (t_2' \cdot u) \) \tag{3}

From 1, 2 and 3,

\( \text{hom}(r_1' \circ r_2')(A) \) is a completion of \( \text{hom}(r_1 \circ r_2)(A) \).

This would also be the case if \( A \in S \).

Hence, \( \text{hom}(r_1' \circ r_2') \) is a completion of \( \text{hom}(r_1 \circ r_2) \).

Thus, \( r' \times s' \) is a completion of \( r \times s \).

Case E. \( E = \sigma(r; f; \mu) \)

Let \( r' \) be a completion of \( r \).

Let \( r' \in r' \) be a completion of \( r \in r \).

\( \tau \) changes to \( \tau \cdot (\text{eval}(f) \cap \mu)(\tau) \) in \( \sigma(r; f; \mu) \) and \( \tau' \) changes to \( \tau' \cdot (\text{eval}(f) \cap \mu)(\tau') \) in \( \sigma(r'; f; \mu) \).

Let the lower and upper limit of assignments in \( \tau \cdot (\text{eval}(f) \cap \mu)(\tau) \) be \( \ell \) and \( u \) respectively and let the lower limit = upper limit = \( u' \) for the assignments in \( \tau' \cdot (\text{eval}(f) \cap \mu)(\tau') \).
We first show that \( \ell \subseteq u' \subseteq u \).

\[
\ell = (r.\ell) \cap eval(f)(r) \cap \mu(\tau).\ell \tag{A}
\]

Now since \( \tau' \) is a completion of \( \tau \), we have \( \tau.\ell \subseteq \tau'.\ell \)

Also, \( \mu(\tau).\ell \subseteq \mu(\tau').\ell \) (By Lemma 5.1)

\[ eval(f)(\tau).\ell \subseteq eval(f)(\tau').\ell \] (by Lemma 5.3)

Using these facts we get from A

\[ \ell \subseteq (\tau'.\ell) \cap eval(f)(\tau').\ell \cap \mu(\tau').\ell \]

Hence \( \ell \subseteq u' \)

Similarly,

\[ u = (r.u) \cap eval(f)(r) \cap \mu(\tau).u \tag{B} \]

Now since \( \tau' \) is a completion of \( \tau \), we have \( \tau.u \supseteq \tau'.u \)

Also, \( \mu(\tau).u \supseteq \mu(\tau').u \) (By Lemma 5.1)

\[ eval(f)(\tau).u \supseteq eval(f)(\tau').u \] (by Lemma 5.3)

Using these facts we get from B

\[ u \supseteq (\tau'.u) \cap eval(f)(\tau').u \cap \mu(\tau').u \]

u \supseteq u'

Hence, \( \ell \subseteq u' \subseteq u \) \tag{1}

For a given attribute \( A \),

\( \tau.\xi(A) \) and \( \tau'.\xi(A) \) agree everywhere both are defined

Hence, clearly \( \tau.\xi(A) \mid (eval(f)(\tau).u \cap \mu(\tau).u) \) and \( \tau'.\xi(A) \mid (eval(f)(\tau').u \cap \mu(\tau').u) \)

agree everywhere both are defined \tag{2}

Also since \( \tau'.\xi(A) \) is defined everywhere along \( \tau'.u \), hence

\( \tau'.\xi(A) \mid (eval(f)(\tau').u \cap \mu(\tau').u) \) is defined everywhere along

\( \tau'.\xi(A) \mid (eval(f)(\tau').u \cap \mu(\tau').u) \) \tag{3}

From 1,2,3, \( \tau'.(eval(f)(\tau').u \cap \mu(\tau')) \) is a completion of \( \tau \mid (eval(f)(\tau).u \cap \mu(\tau)) \)

Hence, \( \sigma(r;f;\mu) \) is a completion of \( \sigma(r;f;\mu) \).

Case F. \( E = \Pi_\mathcal{K}(r) \)

Let \( r' \) be a completion of \( r \)

Hence, by our definition of completions in terms of snapshots, for all \( t \in [0,\text{NOW}] \),

\( r'(t) \) is a snapshot completion of \( r(t) \).

Hence, for all \( t \in [0,\text{NOW}] \)

(i) given a \( \tau \in r(t) \) if \( \tau.\ell = [t,t] \) there is a tuple \( \tau' \in r'(t) \) such that \( \tau' \) is a
snapshot completion of $\tau$ and
(ii) given a $\tau' \in r'(t)$ there is a tuple $\tau \in r(t)$ such that $\tau'$ is a snapshot completion of $\tau$.

But, by definition of restructuring, we have $(I_K(r))(t) = r(t)$ and $(I_K(r'))(t) = r'(t)$.

Hence, for all $t \in [0, NC^*]$
(i) given a $\tau \in (I_K(r))(t)$ if $\tau.\ell = [t, t]$ there is a tuple $\tau' \in (I_K(r'))(t)$ such that $\tau'$ is a snapshot completion of $\tau$ and
(ii) given a $\tau' \in (I_K(r'))(t)$ there is a tuple $\tau \in (I_K(r))(t)$ such that $\tau'$ is a snapshot completion of $\tau$.

Hence, for all $t \in [0, NC^*] (I_K(r'))(t)$ is a snapshot completion of $(I_K(r))(t)$.

Hence, $I_K(r')$ is a completion of $I_K(r)$.

Hence the result. $\Box$

The above proposition is analogous to the notion of adequacy of operators as defined in [Bi83]. However, a transition to our model is not trivial. This is because (i) in [Bi83] $A \theta B$ and $A \theta b$ are defined only when $\theta$ is the equality comparison, and (ii) our selection operation is of the form $\sigma(r; f; \mu)$ and it cannot be evaluated using snapshots of relations at instants of time (see Section 3.6.3).

7.4. Results when Construct $[[E]]$ is Included

We now reintroduce the construct $[[E]]$ into the model to study the effect of the addition of the construct on our results in Section 7.3. We find that results similar to those in Section 7.3 continue to hold. However, we postponed the discussion of this construct for two reasons.

First, with the construct $[[E]]$ included there is no clear separation between the lemmas and Proposition 7.1. Because of the recursive nature of the model we would need the result of Proposition 7.1 to prove the lemmas and we would need the lemmas to prove the Proposition 7.1. Hence we need to break this cycle which requires us to state three auxiliary results (Lemma 7.4, Lemma 7.5, Lemma 7.6) which are then used to prove a theorem which is similar to Proposition 7.1. We then use the results in the Theorem to prove results analogous to the lemmas of Section 7.3.
The second reason is a matter of notation. When we have a construct like \([E]\), the recursive evaluation of a temporal expression \(\mu\) now also depends on the state of the database. This was implicit in Section 6.1. Since we want to compare \(E(\delta')\) with \(E(\delta)\) we now need to make this explicit in our notation. We thus need to define the evaluation of \(\mu(\delta, \tau)\) for a partial temporal expression \(\mu\), where \(\delta\) is a database and \(\tau\) is a tuple. However these definitions are completely analogous to the definitions in Section 6.1. For instance, \([E](\delta, \tau) = (\cup_{\tau_1 \in E(\delta)} \tau_1 \cdot \ell, \cup_{\tau_1 \in E(\delta)} \tau_1 \cdot \ell').\)

Since the evaluation of \(\mu\) requires the parameter \(\delta\) in the definition, the function \(\text{eval}\) will also require the parameter \(\delta\) in the definition. Again, the definitions are completely analogous to the definitions in Section 6.2.

Since temporal expressions may now contain relational expressions, we find that we cannot prove results analogous to Lemma 7.1 through Lemma 7.3 without a result analogous to Proposition 7.1. On the other hand, proving a result like Proposition 7.1 requires the lemmas. To break this cycle we prove the lemmas with an if statement, then prove the theorem and finally remove the if statement in the lemmas.

**Lemma 7.4.** Let \(\delta'\) be a completion of \(\delta\). Let \(\tau, \tau'\) be tuples over \(R\) such that \(\tau'\) is a completion of \(\tau\). Let \(\mu\) be a partial temporal expression. If for every relational expression \(E\) in \(\mu\) we have \(E(\delta')\) to be a completion of \(E(\delta)\) then \(\mu(\delta, \tau) \cdot \ell \subseteq \mu(\delta', \tau') \cdot \ell\) and \(\mu(\delta, \tau) \cdot u \supseteq \mu(\delta', \tau') \cdot u\).

**Proof:** By induction on complexity of \(\mu\).

If \(\mu\) is a temporal element or of the form \([A], [A \land B], [A \lor B]\), the proof is similar to the proof of Lemma 7.1. Suppose \(\mu\) is of the form \([E]\) where \(E\) is a relational expression. Then \(\mu(\delta, \tau) \cdot \ell = [E](\delta, \tau) \cdot \ell = \cup_{\tau_1 \in E(\delta)} \tau_1 \cdot \ell = \cup_{\tau_1 \in E(\delta)} (\tau_1 \cdot \ell) \cup \tau_1 \cdot \ell \neq \emptyset\cdot \ell\).

Since by hypothesis \(E(\delta')\) is a completion of \(E(\delta)\), for every such tuple \(\tau_1\) there is a tuple \(\tau'_1 \in E(\delta')\) such that \(\tau'_1\) is a completion of \(\tau_1\) (and so \(\tau_1 \cdot \ell \subseteq \tau'_1 \cdot \ell\)). Hence, \(\mu(\delta, \tau) \cdot \ell \subseteq \cup_{\tau'_1 \in E(\delta')} \tau'_1 \cdot \ell\). Hence \(\mu(\delta, \tau) \cdot \ell \subseteq [E](\delta', \tau') \cdot \ell\). i.e. \(\mu(\delta, \tau) \cdot \ell \subseteq \mu(\delta', \tau') \cdot \ell\).

Similarly, \(\mu(\delta', \tau') \cdot u \subseteq [E](\delta', \tau') \cdot u = \cup_{\tau'_1 \in E(\delta')} \tau'_1 \cdot u\). Since by hypothesis \(E(\delta')\) is a completion of \(E(\delta)\), for every such tuple \(\tau'_1\) there is a tuple \(\tau_1 \in E(\delta)\) such that \(\tau'_1\) is a completion of \(\tau_1\) (and so \(\tau'_1 \cdot u \subseteq \tau_1 \cdot u\)). Hence, \(\mu(\delta', \tau') \cdot u \subseteq \cup_{\tau_1 \in E(\delta)} \tau_1 \cdot u\). Hence \(\mu(\delta', \tau') \cdot u \subseteq [E](\delta, \tau) \cdot u\). i.e. \(\mu(\delta, \tau) \cdot u \supseteq \mu(\delta', \tau') \cdot u\).

For the inductive step, i.e. for \(\cup, \cap\) and \(-\) the proof is similar to Lemma 7.1. □
**Lemma 7.5.** Let $\delta'$ be a completion of $\delta$. Let $\tau, \tau'$ be tuples over $R$ such that $\tau'$ is a completion of $\tau$. Let $f$ be a partial boolean expression. If for every relational expression $E$ in $f$ we have $E(\delta')$ to be a completion of $E(\delta)$ then

1. if $\text{eval}(f)(\delta', \tau) = (1, 1)$ then $\text{eval}(f)(\delta, \tau) = (1, 1)$ or $(\emptyset, 1)$
2. if $\text{eval}(f)(\delta', \tau') = (\emptyset, \emptyset)$ then $\text{eval}(f)(\delta, \tau) = (\emptyset, \emptyset)$ or $(\emptyset, 1)$
3. if $\text{eval}(f)(\delta, \tau) = (1, 1)$ then $\text{eval}(f)(\delta', \tau') = (1, 1)$
4. if $\text{eval}(f)(\delta, \tau) = (\emptyset, \emptyset)$ then $\text{eval}(f)(\delta', \tau') = (\emptyset, \emptyset)$

**Proof:** Similar to Lemma 7.2. □

**Lemma 7.6.** Let $\delta'$ be a completion of $\delta$. Let $\tau, \tau'$ be tuples over $R$ such that $\tau'$ is a completion of $\tau$. Let $f$ be a partial boolean expression. If for every relational expression $E$ in $f$ we have $E(\delta')$ to be a completion of $E(\delta)$ then

1. $\text{eval}(f)(\delta, \tau).l \subseteq \text{eval}(f)(\delta', \tau').l$
2. $\text{eval}(f)(\delta, \tau).u \supseteq \text{eval}(f)(\delta', \tau').u$

**Proof:** Immediate from statements 1 and 2 of Lemma 7.5 and the fact that $\text{eval}(f)(\delta', \tau')$ is either $(\emptyset, \emptyset)$ or $(1, 1)$. □

The following theorem shows that the results of our algebraic expressions are reliable even when constructs such as $[[E]]$ are allowed.

**Theorem 7.1.** Let $\delta$ be the state of a database and let $E$ be a relational expression applied to the database. Then for any completion $\delta'$ of the database $\delta$, $E(\delta')$ is a completion of $E(\delta)$.

**Proof:** We sketch below the proof for the selection operator. The proof for the other operators remain the same as the proof of Proposition 7.1.

Let $\tau'$ be a completion of $\tau$ in the completion $\delta'$ of $\delta$. Let $\tau' \in \tau$ be a completion of $\tau \in \tau$. $\tau$ changes to $\tau' \uparrow (\text{eval}(f) \cap \mu)(\delta, \tau)$ in $\sigma(\tau'; \mu)$ and $\tau'$ changes to $\tau' \uparrow (\text{eval}(f) \cap \mu)(\delta', \tau')$ in $\sigma(\tau'; \mu)$. Let the lower and upper limit of assignments in $\tau \uparrow (\text{eval}(f) \cap \mu)(\delta, \tau)$ be $l$ and $u$ respectively and let the lower limit = upper limit = $u'$ for the assignments in $\tau' \uparrow (\text{eval}(f) \cap \mu)(\delta', \tau')$.

We first show that $l \subseteq u' \subseteq u$.

\[ l = (\tau \cdot l) \cap \text{eval}(f)(\delta, \tau) \cdot l \cap \mu(\delta, \tau) \cdot l, \text{ by definition of } \sigma \] (A)
Now since \( \tau' \) is a completion of \( \tau \), we have \( \tau \cdot \xi \subseteq \tau' \cdot \xi \). Now by induction hypothesis for every relational expression \( E \) in \( \mu \) we have \( E(\delta') \) is a completion of \( E(\delta) \).

Hence, \( \mu(\delta,\tau) \cdot \xi \subseteq \mu(\delta',\tau') \cdot \xi \), by Lemma 7.4. Similarly, \( \text{eval}(f)(\delta,\tau) \cdot \xi \subseteq \text{eval}(f)(\delta',\tau') \cdot \xi \), by Lemma 7.6. Using these facts we get from (A), \( \xi \subseteq (\tau' \cdot \xi) \cap \text{eval}(f)(\delta',\tau') \cdot \xi \cap \mu(\delta',\tau') \cdot \xi = \mu' \) by definition of \( \sigma \). Hence \( \xi \subseteq \mu' \). The proof of \( \mu' \subseteq \mu \) is similar.

Hence, \( \xi \subseteq \mu' \subseteq \mu \) (1)

For a given attribute \( A \), \( \tau \cdot \xi(A) \) and \( \tau' \cdot \xi(A) \) agree everywhere both are defined. Hence, \( \tau \cdot \xi(A) \perp (\text{eval}(f)(\delta,\tau) \cup \mu(\delta,\tau) \cdot \xi) \) and \( \tau' \cdot \xi(A) \perp (\text{eval}(f)(\delta',\tau') \cup \mu(\delta',\tau') \cdot \xi) \) agree everywhere both are defined. (2)

Also since \( \tau' \cdot \xi(A) \) is defined everywhere along \( \tau' \cdot \xi \), hence \( \tau' \cdot \xi(A) \perp (\text{eval}(f)(\delta',\tau') \cup \mu(\delta',\tau') \cdot \xi) \) is defined everywhere along \( \tau' \cdot \xi \). (3)

Then using (1), (2) and (3), from the definition of completion, we see that \( \tau' \cdot \xi \cup \text{eval}(f)(\delta',\tau') \cdot \xi \) is a completion of \( \tau \cdot \xi(\delta,\tau) \). Hence, \( \sigma(\tau;f;\mu) \) is a completion of \( \sigma(\tau;f;\mu) \).

The following propositions which are analogous to Lemma 7.1 and Lemma 7.2 now easily follow from Theorem 7.1, Lemma 7.4 and Lemma 7.5.

**Proposition 7.2.** Let \( \delta' \) be a completion of \( \delta \). Let \( \tau,\tau' \) be tuples over \( R \) such that \( \tau' \) is a completion of \( \tau \). Let \( \mu \) be a partial temporal expression. Then \( \mu(\delta,\tau) \cdot \xi \subseteq \mu(\delta',\tau') \cdot \xi \).

**Proof:** By Theorem 7.1, for every expression \( E \), \( E(\delta') \) is a completion of \( E(\delta) \). Hence by Lemma 7.4 the result follows.

**Proposition 7.3.** Let \( \delta' \) be a completion of \( \delta \). Let \( \tau,\tau' \) be tuples over \( R \) such that \( \tau' \) is a completion of \( \tau \). Let \( f \) be a partial boolean expression. Then

1. if \( \text{eval}(f)(\delta',\tau') = (1,1) \) then \( \text{eval}(f)(\delta,\tau) = (1,1) \) or \( (0,1) \)
2. if \( \text{eval}(f)(\delta',\tau') = (0,0) \) then \( \text{eval}(f)(\delta,\tau) = (0,0) \) or \( (0,1) \)
3. if \( \text{eval}(f)(\delta,\tau) = (1,1) \) then \( \text{eval}(f)(\delta',\tau') = (1,1) \)
4. if \( \text{eval}(f)(\delta,\tau) = (0,0) \) then \( \text{eval}(f)(\delta',\tau') = (0,0) \)
**Proof:** By Theorem 7.2 for every expression $E$ we have $E(\delta')$ is a completion of $E(\delta)$. Hence by Lemma 7.6 the results follow. \hfill \Box

Part 1 of Proposition 7.3 shows that if $\tau'$ is a completion of $\tau$ and $\delta'$ is a completion of $\delta$ (and hence $\delta',\tau'$ are consistent with $\delta,\tau$ resp. but have complete information) and if $f(\delta',\tau')$ evaluates to TRUE then in our model $f(\delta,\tau)$ will not evaluate to FALSE in spite of $\delta$ and $\tau$ having incomplete information. Similarly, part 2 of the above proposition states that if $f(\delta',\tau')$ evaluates to FALSE then we will not evaluate $f(\delta,\tau)$ to be TRUE in spite of incomplete information. Part 3 of the proposition states that if we evaluate $f(\delta,\tau)$ to be TRUE then $f(\delta',\tau')$ is TRUE for every tuple $\tau'$ which is a completion of $\tau$ and database $\delta'$ which is a completion of $\delta$. Similarly, according to part 4 of the proposition, if we evaluate $f(\delta,\tau)$ to be FALSE then $f(\delta',\tau')$ is FALSE for every tuple $\tau'$ which is a completion of $\tau$ and database $\delta'$ which is a completion of $\delta$. 
8. MAXIMALITY OF OPERATORS

In Chapter 7 we showed that the results of algebraic expressions are reliable in the sense that they never produce incorrect information. However, we want to produce results that have as much information as possible. Thus we want the results of our operations to be maximal in some sense. We examine the maximality of our operations through the notions of information content of relations and extensions of relations.

8.1. Information Content of a Relation

As we mentioned in Section 7.1, the more incomplete the information in \( r \), the larger is the set of possible completions for \( r \). On the other hand, the more complete the information in \( r \), the smaller the set of possible completions of \( r \). This motivates the following definition for more informative relations. The set of completions of a relation \( r \), denoted \( C_r \), is given by \( C_r = \{r': r' \text{ is a completion of } r\} \). A relation \( r_2 \) is more informative than a relation \( r_1 \) if \( C_{r_2} \subset C_{r_1} \). A relation \( r_2 \) is as informative as \( r_1 \) if \( C_{r_2} = C_{r_1} \). A relation \( r_2 \) is strictly more informative than a relation \( r_1 \) if \( C_{r_2} \subset C_{r_1} \).

To find if \( r_2 \) is more informative than \( r_1 \) based on the above definition one has to compute the set of completions for the relations \( r_1 \) and \( r_2 \). However, a relation has a potentially infinite set of completions. Hence, a more syntactic notion is needed by which one can determine if a relation \( r_2 \) is more informative than a relation \( r_1 \). Hence, we introduce the notion of extensions. The relationship between extensions and information content of relations is then captured in Theorem 8.1.

Let \( \tau \) and \( \tau' \) be tuples over a scheme \( R \). Then \( \tau' \) is an extension of \( \tau \) if the following conditions hold (i) \( \tau.\ell \subseteq \tau'.\ell \subseteq \tau'.u \subseteq \tau.u \), (ii) for all \( A \in R \), \( \tau'.\xi(A) \) is defined everywhere in \( \tau'.u \) where \( \tau.\xi(A) \) is defined, and (iii) for all \( A \in R \), \( \tau'.\xi(A) \) and \( \tau.\xi(A) \) agree everywhere both of them are defined. A tuple \( \tau' \) is a proper extension of \( \tau \) if (i) \( \tau' \) is an extension of \( \tau \), and (ii) \( \tau \) is not an extension of \( \tau' \).

These definition can be extended to relations. A relation \( r' \) is an extension of \( r \) if
(i) given a tuple \( \tau \in r \) with \( \tau.l \neq \emptyset \), there is a \( \tau' \in r' \) such that \( \tau' \) is an extension of \( \tau \),
(ii) given a tuple \( \tau' \in r' \) there is a \( \tau \in r \) such that \( \tau' \) is an extension of \( \tau \). A relation \( r' \)

is a proper extension of \( r \) if (i) \( r' \) is an extension of \( r \), and (ii) \( r \) is not an extension of \( r' \).

As in the case of completions, the phrase "there is" may be replaced by "there is exactly one" in the definition for proper extension and extension of relations. The following theorem shows the relation between extensions and more informative relations.

**Theorem 8.1.** Let \( r_1 \) and \( r_2 \) be two relations. Then

(i) \( r_2 \) is more informative than \( r_1 \) iff \( r_2 \) is an extension of \( r_1 \)

(ii) \( r_2 \) is strictly more informative than \( r_1 \) iff \( r_2 \) is a proper extension of \( r_1 \)

**Proof:**

Proof of (i)

(if part)

\( r_2 \) is an extension of \( r_1 \)

Suppose \( r' \in C_{r_2} \)

Let \( \tau_1 \in r_1 \) such that \( \tau_1.l \neq \emptyset \)

Then there is a \( \tau_2 \in r_2 \) such that \( \tau_2 \) is an extension of \( \tau_1 \) (since \( r_2 \) is an extension of \( r_1 \)).

Hence \( \tau_2.l \neq \emptyset \) (since \( \tau_1.l \subseteq \tau_2.l \) by definition of extensions)

Hence, there is a \( \tau' \in r' \) such that \( \tau' \) is a completion of \( \tau_2 \) (since \( r' \) is a completion of \( r_2 \)).

Hence, there is a \( \tau' \in r' \) such that \( \tau' \) is a completion of \( \tau_1 \)

Hence, for any \( \tau_1 \in r_1 \) with \( \tau_1.l \neq \emptyset \) there is a \( \tau' \in r' \) such that \( \tau' \) is a completion of \( \tau_1 \). (1)

Let \( \tau' \in r' \)

Then there is a \( \tau_2 \in r_2 \) such that \( \tau' \) is a completion of \( \tau_2 \) (since \( r' \) is a completion of \( r_2 \)).

But for \( \tau_2 \in r_2 \) there is a \( \tau_1 \in r_1 \) such that \( \tau_2 \) is an extension of \( \tau_1 \) (since \( r_2 \) is an extension of \( r_1 \)).

Hence, there is a \( \tau_1 \in r_1 \) such that \( \tau' \) is a completion of \( \tau_1 \)

Hence, for any \( \tau' \in r' \) there is a \( \tau_1 \in r_1 \) such that \( \tau' \) is a completion of \( \tau_1 \). (2)
By (1), (2) and definition of completion, \( r' \) is a completion of \( r_1 \).

Hence, \( r' \in C_{r_1} \)

Hence \( C_{r_2} \nsubseteq C_{r_1} \)

Hence, \( r_2 \) is more informative than \( r_1 \) (only if part)

\( r_2 \) is more informative than \( r_1 \)

Hence \( C_{r_2} \nsubseteq C_{r_1} \)

Suppose \( r_2 \) is not an extension of \( r_1 \)

Then at least one of the following conditions must not hold

(A) Given a tuple \( \tau_1 \in r_1 \) with \( \tau_1.l \neq \emptyset \) there is a \( \tau_2 \in r_2 \) such that \( \tau_2 \) is an extension of \( \tau_1 \).

(B) Given a tuple \( \tau_2 \in r_2 \) there is a \( \tau_1 \in r_1 \) such that \( \tau_2 \) is an extension of \( \tau_1 \).

Case A Suppose (A) does not hold

Then for some tuple \( \tau_1 \in r_1 \) such that \( \tau_1.l \neq \emptyset \) at least one of the following is true

(a) there is no key-equivalent tuple \( \tau_2 \in r_2 \)

(b) In the key-equivalent tuple \( \tau_2 \in r_2, \tau_1.l \nsubseteq \tau_2.l \)

(c) In the key-equivalent tuple \( \tau_2 \in r_2, \tau_2.u \nsubseteq \tau_1.u \)

(d) In the key-equivalent tuple \( \tau_2 \in r_2 \), for some \( A \in R_{r_2} \xi(A) \) is not defined somewhere in \( \tau_2.u \) where \( \tau_1.\xi(A) \) is defined

(e) In the key-equivalent tuple \( \tau_2 \in r_2 \), for some \( A \in R_{r_2} \xi(A)(t) \neq \tau_1.\xi(A)(t) \) for some instant \( t \), although both of them are defined at \( t \).

Case (a) There is no key-equivalent tuple \( \tau_2 \in r_2 \).

Then in a completion \( r' \) of \( r_2 \) there is no tuple key-equivalent to \( \tau_1 \).

In completion \( r' \) of \( r_2 \) there is no tuple key-equivalent to \( \tau_1 \in r_1 \)

Hence, since \( \tau_1.l \neq \emptyset, r' \) is not a completion of \( r_1 \)

This contradicts \( C_{r_2} \nsubseteq C_{r_1} \)

Case (b) In the key-equivalent tuple \( \tau_2 \in r_2, \tau_1.l \nsubseteq \tau_2.l \)

Construct a completion \( r' \) of \( r_2 \) that has a tuple \( \tau' \in r' \) key-equivalent to \( \tau_2 \) such that

\[ \tau'.l = \tau'.u = \tau_2.l \]

Then \( \tau' \in r' \) is key-equivalent to \( \tau_1 \in r_1 \) but \( \tau_1.l \nsubseteq \tau'.l \)

Hence \( \tau' \) is not a completion of \( r_1 \)
Hence \( r' \) is not a completion of \( r_1 \)
This contradicts \( C_{r_2} \subseteq C_{r_1} \)

Case (c) In the key-equivalent tuple \( \tau_2 \in r_2, \tau_2.u \notin \tau_1.u \)
Construct a completion \( r' \) of \( r_2 \) that has a tuple \( \tau' \in r' \) key-equivalent to \( \tau_2 \) such that
\[
\tau'.l = \tau'.u = \tau_2.u
\]
Then \( \tau' \in r' \) is key-equivalent to \( \tau_1 \in r_1 \) but \( \tau'.u \notin \tau_1.u \)
Hence \( \tau' \) is not a completion of \( \tau_1 \)
Hence \( r' \) is not a completion of \( r_1 \)
This contradicts \( C_{r_2} \subseteq C_{r_1} \)

Case (d) In the key-equivalent tuple \( \tau_2 \in r_2 \), for some \( A \in R \), \( \tau_2.\xi(A) \) is not defined somewhere in \( \tau_2.u \) where \( \tau_1.\xi(A) \) is defined.
Hence, at some time \( t \), \( \tau_1.\xi(A)(t) = a \) (for some \( a \)) but \( \tau_2.\xi(A)(t) \) is not defined.
Construct a completion \( r' \) of \( r_2 \) that has a tuple \( \tau' \in r' \) key-equivalent to \( \tau_2 \) such that
\[
\tau'.l = \tau'.u = \tau_2.u \text{ and } \tau'.\xi(A)(t) = b \ (\neq a)
\]
Then \( \tau' \in r' \) is key-equivalent to \( \tau_1 \in r_1 \) but \( \tau' \) and \( \tau_1 \) are both defined at \( t \) but do not agree at \( t \).
Hence \( \tau' \) is not a completion of \( \tau_1 \)
Hence \( r' \) is not a completion of \( r_1 \)
This contradicts \( C_{r_2} \subseteq C_{r_1} \)

Case (e) In the key-equivalent tuple \( \tau_2 \in r_2 \), for some \( A \in R \), \( \tau_2.\xi(A) \neq \tau_1.\xi(A) \) for some instant \( t \), although both of them are defined at \( t \).
The proof that this contradicts \( C_{r_2} \subseteq C_{r_1} \) is similar to Case (d)
Hence (A) must hold.

Case B Suppose (B) does not hold
Then for some tuple \( \tau_2 \in r_2 \) at least one of the following holds
(a) there is no key-equivalent tuple \( \tau_1 \in r_1 \)
(b) In the key-equivalent tuple \( \tau_1 \in r_1, \tau_1.l \notin \tau_2.l \)
(c) In the key-equivalent tuple \( \tau_1 \in r_1, \tau_2.u \notin \tau_1.u \)
(d) In the key-equivalent tuple \( \tau_1 \in r_1 \), for some \( A \in R \), \( \tau_2.\xi(A) \) is not defined
somewhere in r^u where r_1.ξ(A) is defined
(e) In the key-equivalent tuple τ_1 ∈ r_1, for some A ∈ R, τ_2.ξ(A)(t) ≠ τ_1.ξ(A)(t) for some instant t, although both of them are defined at t.

Case (a) There is no tuple key-equivalent to τ_1 ∈ r_1
    Construct a completion r' of r_2 that has a tuple τ' ∈ r' which is key-equivalent to τ_2 such that
    τ'.L = τ'.u = τ_2.u
    τ' has no key-equivalent counterpart in r_1
    Hence r' is not a completion of r_1
    This contradicts C_{r_2} ⊂ C_{r_1}
Case (b),(c),(d),(e) are similar to Case (A).
    Hence both A and B must hold
    Hence our supposition that r_2 is not an extension of r_1 is false.
    Thus r_2 is an extension of r_1.
    This proves part (i) of the theorem.

Proof of (ii)
(if part)
    r_2 is a proper extension of r_1
    Thus r_2 is an extension of r_1
    Hence by part (i) of the theorem,
    C_{r_2} ⊂ C_{r_1}
    Since r_2 is a proper extension of r_1, hence r_1 is not an extension of r_2.
    Hence C_{r_1} ⊄ C_{r_2}
    Hence C_{r_2} ⊄ C_{r_1}
    From 1 and 2, C_{r_2} ⊂ C_{r_1}
    Hence r_2 is strictly more informative than r_1

(only if part)
    r_2 is strictly more informative than r_1
    Hence r_2 is more informative than r_1
    Hence r_2 is an extension of r_1 (by part (i) of the theorem)
    Also, r_1 is not an extension of r_2 else r_1 is more informative than r_2 and so C_{r_1} ⊂
C_{r_2} which contradicts C_{r_2} \subset C_{r_1}

Hence \( r_1 \) is not an extension of \( r_2 \)

From 1 and 2, \( r_2 \) is a proper extension of \( r_1 \).

This proves part (ii) of the theorem. \( \square \)

### 8.2. The Maximality Theorem

We now examine how far our definition of the operations produce as much information as is possible. Our result is analogous to the restrictedness property of operators for classical databases with null values as developed in [Bi83]. Again, however, our result cannot be obtained as a straightforward transition from the snapshot case to the temporal case.

Theorem 8.2. Let \( r_1, r_2 \) be two distinct relations and let \( r_3 \) be a proper extension of \( r_1 \odot r_2 \) where \( \odot \) is a relational operator. Then there are completions \( r'_1, r'_2 \) of \( r_1, r_2 \) respectively such that \( r'_1 \odot r'_2 \) is not a completion of \( r_3 \) (This shows that the result of \( r_1 \odot r_2 \) cannot be strengthened to \( r_3 \)). For unary operators we have a similar statement.

The above statement holds for the following operators

- \( \times, U, -, \Pi \), restructuring
- \( \sigma \) of the form \( \sigma (r; ; \mu), \sigma (r; ; [A]), \sigma (r; ; [A \theta b]), \sigma (r; ; [A \theta B]), \sigma (r; \mu \subset [A]); \), \( \sigma (r; \mu \subset [A \theta b]); \), \( \sigma (r; \mu \subset [A \theta B]); \) where \( r \) is a relation, \( \mu \) is a temporal element and \( \theta \) is one of \( \{<, \leq, >, \geq, \neq\} \). (Note that it does not hold if \( \theta \) is \( = \), see Example 8.1 for a counterexample).

Proof:

Cross-Product.

Since \( r_3 \) is a proper extension of \( r_1 \times r_2 \) at least one of the following holds for some pair of key-equivalent tuples \( \tau \in r_1 \times r_2 \) and \( \tau_3 \in r_3 \)

i) \( \tau.l \subset \tau_3.l \)

ii) \( \tau.u \supset \tau_3.u \)

iii) For some \( A \in R_1R_2 \), \( \tau_3.\xi(A) \) is defined somewhere in \( \tau_3.u \) where \( \tau.\xi(A) \) is not defined.
Case (i): \( \tau \cdot l \subset \tau_3 \cdot l \)

There is a tuple \( \tau_1 \in r_1, \tau_2 \in r_2 \) such that \( \tau = \text{hom}(\tau_1 \circ \tau_2) \) (since \( \tau \in r_1 \times r_2 \))

Hence, \( \tau \cdot l = \tau_1 \cdot l \cap \tau_2 \cdot l \) (by definition of \( \times \))

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has

\[
\tau'_1 \cdot l = \tau'_1 \cdot u = \tau_1 \cdot l
\]

and choose a completion \( r'_2 \) of \( r_2 \) where the tuple \( \tau'_2 \in r'_2 \) which is key equivalent to \( \tau_2 \in r_2 \) has

\[
\tau'_2 \cdot l = \tau'_2 \cdot u = \tau_2 \cdot l
\]

Then in \( r'_1 \times r'_2 \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) (and so key-equivalent to \( \tau_3 \)) such that

\[
\tau' \cdot l = \tau'_1 \cdot l \cap \tau'_2 \cdot l
\]

\[
= \tau_1 \cdot l \cap \tau_2 \cdot l
\]

\[
= \tau \cdot l
\]

\[
\subset \tau_3 \cdot l
\]

Hence, \( \tau' \in r'_1 \times r'_2 \) cannot be a completion of \( \tau_3 \).

Thus \( r'_1 \times r'_2 \) cannot be a completion \( r_3 \).

Case (ii): \( \tau \cdot u \subset \tau_3 \cdot u \)

There is a tuple \( \tau_1 \in r_1, \tau_2 \in r_2 \) such that \( \tau = \text{hom}(\tau_1 \circ \tau_2) \) (since \( \tau \in r_1 \times r_2 \))

Hence, \( \tau \cdot u = \tau_1 \cdot u \cap \tau_2 \cdot u \) (by definition of \( \times \))

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has

\[
\tau'_1 \cdot l = \tau'_1 \cdot u = \tau_1 \cdot u
\]

and choose a completion \( r'_2 \) of \( r_2 \) where the tuple \( \tau'_2 \in r'_2 \) which is key equivalent to \( \tau_2 \in r_2 \) has

\[
\tau'_2 \cdot l = \tau'_2 \cdot u = \tau_2 \cdot u
\]

Then in \( r'_1 \times r'_2 \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[
\tau' \cdot u = \tau'_1 \cdot u \cap \tau'_2 \cdot u
\]

\[
= \tau_1 \cdot u \cap \tau_2 \cdot u
\]

\[
= \tau \cdot u
\]

\[
\subset \tau_3 \cdot u
\]

Hence, \( \tau' \in r'_1 \times r'_2 \) cannot be a completion of \( \tau_3 \).
Thus \( r'_1 \times r'_2 \) cannot be a completion \( r_3 \).

Case (iii): For some \( A \in R_1 \times R_2 \), \( \tau_3 \cdot \xi(A) \) is defined at some instant \( t \) in \( \tau_3 \cdot u \) but in the key-equivalent tuple \( \tau \in r_1 \times r_2 \), \( \tau \cdot \xi(A) \) is not defined at \( t \).

(Without loss of generality, let \( A \) be an attribute in \( R_1 \)).

Hence \( \tau_3 \cdot \xi(A)(t) = a \) (for some \( a \)) and \( \tau \cdot \xi(A) \) is not defined at \( t \)

The instant \( t \in \tau_3 \cdot u \)

Hence, \( t \in \tau \cdot u \) (since \( \tau_3 \) is an extension of \( \tau \))

Now, there is a tuple \( \tau_1 \in r_1 \), \( \tau_2 \in r_2 \) such that \( \tau = \text{hom}(\tau_1 \circ \tau_2) \) (since \( \tau \in r_1 \times r_2 \))

Hence, \( \tau \cdot \xi(A) = \tau_1 \cdot \xi(A) \mid \tau \cdot u \) (by definition of \( \times \))

Hence, since \( t \in \tau \cdot u \), \( \tau_1 \cdot \xi(A) \) is not defined at \( t \) (else \( \tau \cdot \xi(A) \) would be defined at \( t \)).

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) is as follows

(i) \( \tau'_1 \cdot L = \tau'_1 \cdot u = \tau_1 \cdot u \)

(ii) At instant \( t \) (where \( \tau_1 \cdot \xi(A) \) is not defined in \( r_1 \)),

\[ \tau'_1 \cdot \xi(A) = b \ (	ext{where } b \neq a). \]

and choose a completion \( r'_2 \) of \( r_2 \) where the tuple \( \tau'_2 \in r'_2 \) which is key equivalent to \( \tau_2 \in r_2 \) has

\[ \tau'_2 \cdot L = \tau'_2 \cdot u = \tau_2 \cdot u \]

Then in \( r'_1 \times r'_2 \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[ \tau' \cdot \xi(A)(t) = b \] (note that \( \tau' \cdot \xi(A) \) is defined at \( t \))

Hence, \( \tau' \) is not a completion of \( \tau_3 \) since they do not agree everywhere both are defined (specifically they do not agree at \( t \))

Hence, \( \tau' \in r'_1 \times r'_2 \) cannot be a completion of \( \tau_3 \).

Hence, \( r'_1 \times r'_2 \) cannot be a completion of \( r_3 \).

**Union.**

Since \( r_3 \) is a proper extension of \( r_1 \cup r_2 \) at least one of the following holds for some pair of key-equivalent tuples \( \tau \in r_1 \cup r_2 \) and \( \tau_3 \in r_3 

i) \( \tau \cdot L \subset \tau_3 \cdot L \)

ii) \( \tau \cdot u \supset \tau_3 \cdot u \)

iii) For some \( A \in R \), \( \tau_3 \cdot \xi(A) \) is defined somewhere in \( \tau_3 \cdot u \) where \( \tau \cdot \xi(A) \) is not defined.
Case (i): \( r.l \cap r'.l \)

There are key-equivalent tuples \( \tau_1 \in r_1, \tau_2 \in r_2 \) such that \( \tau = \tau_1 \cup \tau_2 \) (since \( \tau \in r_1 \cup r_2 \))

(the case where there is only a key-equivalent tuple in \( r_1 \) and the case where there is only a key-equivalent tuple in \( r_2 \) can be proved similarly)

Hence, \( \tau.l = \tau_1.l \cup \tau_2.l \) (by definition of \( \cup \))

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \) \( \in r_1 \) has

\( \tau'_1.l = \tau'_1.u = \tau_1.l \)

and choose a compatible completion \( r'_2 \) of \( r_2 \) where the tuple \( \tau'_2 \in r'_2 \) which is key equivalent to \( \tau_2 \in r_2 \) has

\( \tau'_2.l = \tau'_2.u = \tau_2.l \)

Then in \( r'_1 \cup r'_2 \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[
\tau'.l = \tau'_1.l \cup \tau'_2.l \\
= \tau_1.l \cup \tau_2.l \\
= \tau.l \\
\subseteq \tau_3.l
\]

Hence, \( \tau' \in r'_1 \cup r'_2 \) cannot be a completion of \( \tau_3 \).

Thus \( r'_1 \cup r'_2 \) cannot be a completion of \( r_3 \).

Case (ii): \( r.u \cup r'.u \)

There are key-equivalent tuples \( \tau_1 \in r_1, \tau_2 \in r_2 \) such that \( \tau = \tau_1 \cup \tau_2 \) (since \( \tau \in r_1 \cup r_2 \))

(the case where there is only a key-equivalent tuple in \( r_1 \) and the case where there is only a key-equivalent tuple in \( r_2 \) can be proved similarly)

Hence, \( \tau.u = \tau_1.u \cup \tau_2.u \) (by definition of \( \cup \))

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \) \( \in r_1 \) has

\( \tau'_1.l = \tau'_1.u = \tau_1.u \)

and choose a compatible completion \( r'_2 \) of \( r_2 \) where the tuple \( \tau'_2 \in r'_2 \) which is key equivalent to \( \tau_2 \in r_2 \) has

\( \tau'_2.l = \tau'_2.u = \tau_2.u \)
Then in $r'_1 \cup r'_2$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

\[
\tau'.u = \tau'_1.u \cup \tau'_2.u = \tau_1.u \cup \tau_2.u = \tau.u
\]

Hence, $\tau' \not\in r'_1 \cup r'_2$ cannot be a completion of $\tau_3$.

Thus $r'_1 \cup r'_2$ cannot be a completion $r'_3$.

Case (iii): For some $A \in R$, $\tau_3.\xi(A)$ is defined at some instant $t$ in $\tau_3.u$ but in the key-equivalent tuple $\tau \in r'_1 \cup r'_2$, $\tau.\xi(A)$ is not defined at $t$.

Hence $\tau_3.\xi(A)(t) = a$ (for some $a$) and $\tau.\xi(A)$ is not defined at $t$.

The instant $t \in \tau_3.u$

Hence, $t \in \tau.u$ (since $\tau_3$ is an extension of $\tau$)

There are key-equivalent tuples $\tau_1 \in r'_1$, $\tau_2 \in r'_2$ such that $\tau = \tau_1 \cup \tau_2$ (since $\tau \in r'_1 \cup r'_2$)

(The case where there is only a key equivalent tuple in $r'_1$ and the case where there is only a key-equivalent tuple in $r'_2$ can be proved similarly)

Hence, $\tau.\xi(A) = \tau_1.\xi(A) \cup \tau_2.\xi(A)$ (by definition of $\cup$)

Hence, since $\tau.\xi(A)$ is not defined at $t$, neither is $\tau_1.\xi(A)$ or $\tau_2.\xi(A)$, although $t$ is an element of at least one of $\tau_1.u$, $\tau_2.u$.

Choose a completion $r'_1$ of $r'_1$ where the tuple $\tau'_1 \in r'_1$ which is key equivalent to $\tau_1 \in r'_1$ is as follows

(i) $\tau'_1.\ell = \tau'_1.u = \tau_1.u$

(ii) If $t \in \tau'_1.u$ (where $\tau_1.\xi(A)$ is not defined in $r'_1$),

\[
\tau'_1.\xi(A) = b \quad (\text{where } b \neq a).
\]

and choose a compatible completion $r'_2$ of $r'_2$ where the tuple $\tau'_2 \in r'_2$ which is key equivalent to $\tau_2 \in r'_2$ has

(i) $\tau'_2.\ell = \tau'_2.u = \tau_2.u$

(ii) If $t \in \tau'_2.u$ (where $\tau_2.\xi(A)$ is not defined in $r'_2$),

\[
\tau'_2.\xi(A) = b \quad (\text{same } b \text{ as above}).
\]

Then in $r'_1 \cup r'_2$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

\[
\tau'.\xi(A)(t) = b \quad (\text{note that } \tau'.\xi(A) \text{ is defined at } t)
\]

Hence, $\tau'$ is not a completion of $\tau_3$ since they do not agree everywhere both are
defined (specifically they do not agree at t)

Hence, \( r' \in r'_1 \cup r'_2 \) cannot be a completion of \( \tau_3 \).

Hence, \( r'_1 \cup r'_2 \) cannot be a completion of \( \tau_3 \).

Projection.

Since \( \tau_3 \) is a proper extension of \( \Pi_X(\tau_1) \) at least one of the following holds for some pair of key-equivalent tuples \( \tau \in \Pi_X(\tau_1) \) and \( \tau_3 \in \tau_3 \):

i) \( \tau.l \subseteq \tau_3.l \)

ii) \( \tau.u \cap \tau_3.u \)

iii) For some \( A \in X \), \( \tau_3.\xi(A) \) is defined somewhere in \( \tau_3.u \) where \( \tau.\xi(A) \) is not defined.

Case (i): \( \tau.l \subseteq \tau_3.l \)

There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \in \Pi_X(\tau_1) \) such that \( \tau = \tau_1(X) \) (since \( \tau \in \Pi_X(\tau_1) \))

Hence, \( \tau.l = \tau_1.l \) (by definition of \( \Pi \))

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau_1' \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has

\[ \tau'_1.l = \tau_1.l = \tau_1.u \]

Then in \( \Pi_X(\tau'_1) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[ \tau'.l = \tau'_1.l \] (by defn of \( \Pi \))

\[ = \tau_1.l \]

\[ = \tau.l \]

\[ \subseteq \tau_3.l \]

Hence, \( \tau' \in \Pi_X(\tau'_1) \) cannot be a completion of \( \tau_3 \).

Thus \( \Pi_X(\tau'_1) \) cannot be a completion \( \tau_3 \).

Case (ii): \( \tau.u \cap \tau_3.u \)

There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \in \Pi_X(\tau_1) \) such that \( \tau = \tau_1(X) \) (since \( \tau \in \Pi_X(\tau_1) \))

Hence, \( \tau.u = \tau_1.u \) (by definition of \( \Pi \))

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau_1' \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has

\[ \tau'_1.l = \tau_1.l = \tau_1.u \]
Then in \( \Pi_X(\tau'_1) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[
\tau'.u = \tau'_1.u \quad \text{(by defn. of } \Pi) \\
= \tau_1.u \\
= \tau.u \\
\subseteq \tau_3.u
\]

Hence, \( \tau' \in \Pi_X(\tau'_1) \) cannot be a completion of \( \tau_3 \).

Thus, \( \Pi_X(\tau'_1) \) cannot be a completion \( r_3 \).

Case (iii): For some \( A \in X \), \( \tau_3.\xi(A) \) is defined at some instant \( t \) in \( \tau_3.u \) but in the key-equivalent tuple \( \tau \in \Pi_X(r_1) \), \( \tau.\xi(A) \) is not defined at \( t \).

Hence, \( \tau_3.\xi(A)(t) = a \) (for some a) and \( \tau.\xi(A) \) is not defined at \( t \)

The instant \( t \in \tau_3.u \)

Hence, \( t \in \tau.u \) (since \( \tau_3 \) is an extension of \( \tau \))

There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \) such that \( \tau(A) = \tau_1(A) \)

Hence, since \( \tau.\xi(A) \) is not defined at \( t \), neither is \( \tau_1.\xi(A) \).

Also note that \( t \in \tau_1.u \) (since \( t \in \tau.u \) and \( \tau.u = \tau_1.u \))

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) is as follows

(i) \( \tau'_1.\xi = \tau'_1.u = \tau_1.u \)

(ii) At instant \( t \) (where \( \tau_1.\xi(A) \) is not defined in \( r_1 \) but which is in \( \tau_1.u = \tau'_1.u \)),

\( \tau'_1.\xi(A) = b \) (where \( b \neq a \)).

Then in \( \Pi_X(\tau'_1) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\( \tau'.\xi(A)(t) = b \) (note that \( \tau'.\xi(A) \) is defined at \( t \))

Hence, \( \tau' \) is not a completion of \( \tau_3 \) since they do not agree everywhere both are defined (specifically they do not agree at \( t \))

Hence, \( \tau' \in \Pi_X(\tau'_1) \) cannot be a completion of \( \tau_3 \).

Hence, \( \Pi_X(\tau'_1) \) cannot be a completion of \( r_3 \).

**Minus.**

Since \( r_3 \) is a proper extension of \( r_1 - r_2 \) at least one of the following holds for some pair of key-equivalent tuples \( \tau \in r_1 - r_2 \) and \( \tau_3 \in r_3 

i) \( \tau.\xi \subseteq \tau_3.\xi \)

ii) \( \tau.u \subseteq \tau_3.u \)
iii) For some $A \in R$, $\tau_3 \xi(A)$ is defined somewhere in $\tau_3 . u$ where $\tau . \xi(A)$ is not defined.

Case (i): $\tau . l \subseteq \tau_3 . l$

There are key-equivalent tuples $\tau_1 \in r_1$, $\tau_2 \in r_2$ from which $\tau$ is computed

Let $y = \{\text{those instants at which all the } \xi\text{s agree or could agree if they were defined to } \tau_1 . u, \tau_2 . u\}$.  

Hence, $\tau . l = \tau_1 . l - y$ (by definition of $-$)

Choose a completion $r'_1$ of $r_1$ where the tuple $\tau'_1 \in r'_1$ which is key-equivalent to $\tau_1 \in r_1$ has

(i) $t'^{r'_1} = t'^{r_1} = t_1 . l$  

(ii) For each attribute $A$, if $\tau_1 . \xi(A)$ is not defined at $t$ for $t \in \tau'_1 . l$ then

if $\tau_2 . \xi(A)$ is defined at $t$ then

$\tau'_1 . \xi(A)(t) = \tau_2 . \xi(A)(t)$

else

$\tau'_1 . \xi(A)(t) = a$ (for some prechosen $a \in \text{dom}(A)$)

Now choose a completion $r'_2$ of $r_2$ where the tuple $\tau'_2 \in r'_2$ which is key-equivalent to $\tau_2 \in r_2$ has

(i) $t'^{r'_2} = t'^{r_2} = t_2 . u$  

(ii) For each attribute $A$, if $\tau_2 . \xi(A)$ is not defined at $t$ for $t \in \tau'_2 . u$ then

if $\tau_1 . \xi(A)$ is defined at $t$ then

$\tau'_2 . \xi(A)(t) = \tau_1 . \xi(A)(t)$

else

$\tau'_2 . \xi(A)(t) = a$ (for the prechosen $a \in \text{dom}(A)$)

Then in $r'_1 - r'_2$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$\tau' . l = \tau_1 . l - \{\text{those instants in } \tau'_1 . l \text{ where the } \xi\text{s in } \tau'_1, \tau'_2 \text{ agree}\}$

$= \tau'_1 . l - (\tau'_1 . l \cap y)$ (by construction of completions)

$= \tau . l$

$\subseteq \tau_3 . l$

Hence, $\tau' \not\in r'_1 - r'_2$ cannot be a completion of $\tau_3$.  

Thus $r'_1 - r'_2$ cannot be a completion $r_3$. 


Case (ii): $\tau.u \cup \tau_3.u$

There are key-equivalent tuples $\tau_1 \in \tau_1$, $\tau_2 \in \tau_2$ from which $\tau$ is computed

Let $x =$ those instants in $\tau_2^* \cdot \ell$ where all the $\xi$s in $\tau_1$ and $\tau_2$ agree.

Hence, $\tau.u = \tau_1.u - x$ (by definition of $-\cdot$)

$$= \tau_1.u - (x \cap \tau_1.u)$$

Choose a completion $\tau_1'$ of $\tau_1$ where the tuple $\tau_1' \in \tau_1'$ which is key equivalent to $\tau_1 \in \tau_1$ has

(i) $\tau_1'.\ell = \tau_1'.u = \tau_1.u$

(ii) For each attribute $A$, if $\tau_1.\xi(A)$ is not defined at $t$ for $t \in \tau_1'.u$ then

if $\tau_2.\xi(A)$ is defined at $t$ then

$$\tau_1'.\xi(A)(t) = \text{some value not equal to } \tau_2.\xi(A)(t)$$

else

$$\tau_1'.\xi(A)(t) = a_1 \text{ (for some prechosen } a_1 \in \text{dom}(A))$$

Now choose a completion $\tau_2'$ of $\tau_2$ where the tuple $\tau_2' \in \tau_2'$ which is key equivalent to $\tau_2 \in \tau_2$ has

(i) $\tau_2'.\ell = \tau_2'.u = \tau_2.\ell$

(ii) For each attribute $A$, if $\tau_2.\xi(A)$ is not defined at $t$ for $t \in \tau_2'.\ell$ then

if $\tau_1.\xi(A)$ is defined at $t$ then

$$\tau_2'.\xi(A)(t) = \text{some value not equal to } \tau_1.\xi(A)(t)$$

else

$$\tau_2'.\xi(A)(t) = a_2 \text{ (for some } a_2 \neq a_1)$$

Then in $\tau_1' - \tau_2'$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$$\tau'.u = \tau_1'.u - \text{those instants where the } \xi$s in $\tau_1'$, $\tau_2'$ agree

$$= \tau_1'.\ell - (\tau_1'.\ell \cap x) \text{ (by construction of completions)}$$

$$= \tau.u$$

$$\cup \tau_3.u$$

Hence, $\tau' \in \tau_1' - \tau_2'$ cannot be a completion of $\tau_3$.

Thus $\tau_1' - \tau_2'$ cannot be a completion $\tau_3$.

Case (iii): For some $A \in \mathbf{R}$, $\tau_3.\xi(A)$ is defined at some instant $t$ in $\tau_3.u$ but in the key-equivalent tuple $\tau \in \tau_1 - \tau_2$, $\tau.\xi(A)$ is not defined at $t$.

Hence $\tau_3.\xi(A)(t) = a$ (for some $a$) and $\tau.\xi(A)$ is not defined at $t$

The instant $t \in \tau_3.u$
Hence, \( t \in \tau.u \) (since \( \tau_3 \) is an extension of \( \tau \))

There are key-equivalent tuples \( \tau_1 \in r_1, \tau_2 \in r_2 \) from which \( \tau \) is computed since \( \tau \in r_1 - r_2 \)

(The case where there is only a key equivalent tuple in \( r_1 \) and none in \( r_2 \) can be proved similarly)

Let \( x = \) those instants in \( r_2 \) at which all the \( \xi \)'s in \( \tau_1, \tau_2 \) agree

Then \( \tau.u = \tau_1.u - x \) (by defn of -)

Hence, \( t \in \tau_1.u - x \)

Hence \( t \in \tau_1.u \) and \( t \notin x \)

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) is as follows

(i) \( \tau'_1.l = \tau_1.u = \tau_1.l \)

(ii) For each attribute \( B \), if \( \tau_1,\xi(B) \) is not defined at \( t \) for \( t \in \tau_1.u \) then

- if \( \tau_2,\xi(\xi(B)) \) is defined at \( t \) then

  \[ \tau'_1,\xi(B)(t) = b \text{ (for some } b \neq \tau_2,\xi(B)(t) \text{ and } b \neq a) \]

- else

  \[ \tau'_1,\xi(B)(t) = b \text{ (for some } b \neq a) \]

Now choose a completion \( r'_2 \) of \( r_2 \) where the tuple \( \tau'_2 \in r'_2 \) which is key equivalent to \( \tau_2 \in r_2 \) has

(i) \( \tau'_2.l = \tau'_2.u = \tau_2.l \)

(ii) For each attribute \( B \), if \( \tau_2,\xi(\xi(B)) \) is not defined at \( t \) for \( t \in \tau_2.l \) then

- if \( \tau_1,\xi(\xi(B)) \) is defined at \( t \) then

  \[ \tau'_2,\xi(B)(t) = \text{ some value not equal to } \tau_1,\xi(B)(t) \]

- else

  \[ \tau'_2,\xi(\xi(B)(t)) = c \text{ (for some } c \neq b) \]

Then in \( r'_1 - r'_2 \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[ \tau'.\xi(A)(t) = b \text{ (note that } \tau'.\xi(A) \text{ is defined at } t) \]

Hence, \( \tau' \) is not a completion of \( \tau_3 \) since they do not agree everywhere both are defined (specifically they do not agree at \( t \))

Hence, \( \tau' \in r'_1 - r'_2 \) cannot be a completion of \( \tau_3 \).

Hence, \( r'_1 - r'_2 \) cannot be a completion of \( r_3 \).
Selection of the form $\sigma(r_1; \mu)$.

Since $r_3$ is a proper extension of $\sigma(r_1; \mu)$ at least one of the following holds for some pair of key-equivalent tuples $\tau \in \sigma(r_1; \mu)$ and $\tau_3 \in r_3$

i) $\tau.l \subset \tau_3.l$

ii) $\tau.u \cap \tau_3.u$

iii) For some $A \in R$, $\tau_3.\xi(A)$ is defined somewhere in $\tau_3.u$ where $\tau.\xi(A)$ is not defined.

Case (i): $\tau.l \subset \tau_3.l$

There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1; \mu)$ such that $\tau = \tau_1^1 \langle \mu, \mu \rangle$

Hence, $\tau.l = \tau_1.l \cap \mu$ (by definition of $\sigma$)

Choose a completion $r'_1$ of $r_1$ where the tuple $\tau'_1 \in r'_1$ which is key equivalent to $\tau_1 \in r_1$ has

$\tau'_1.l = \tau'_1.u = \tau_1.l$

Then in $\sigma(r'_1; \mu)$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$\tau'.l = \tau'_1.l \cap \mu$ (by defn of $\sigma$)

$= \tau_1.l \cap \mu$

$= \tau.l$

$\subset \tau_3.l$

Hence, $\tau' \in \sigma(r'_1; \mu)$ cannot be a completion of $\tau_3$. Thus $\sigma(r'_1; \mu)$ cannot be a completion $r_3$.

Case (ii): $\tau.u \cap \tau_3.u$

There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1; \mu)$ such that $\tau = \tau_1^1 \langle \mu, \mu \rangle$

Hence, $\tau.u = \tau_1.u \cap \mu$ (by definition of $\sigma$)

Choose a completion $r'_1$ of $r_1$ where the tuple $\tau'_1 \in r'_1$ which is key equivalent to $\tau_1 \in r_1$ has

$\tau'_1.l = \tau'_1.u = \tau_1.u$

Then in $\sigma(r'_1; \mu)$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$\tau'.u = \tau'_1.u \cap \mu$ (by defn of $\sigma$)

$= \tau_1.u \cap \mu$

$= \tau.u$

$\subset \tau_3.u$

Hence, $\tau' \in \sigma(r'_1; \mu)$ cannot be a completion of $\tau_3$.
Thus \( \sigma(r_1'; \mu) \) cannot be a completion \( r_3 \).

Case (iii): For some \( A \in R \), \( \tau_3, \xi(A) \) is defined at some instant \( t \) in \( \tau_3.u \) but in the key-equivalent tuple \( \tau \in \sigma(r_1'; \mu) \), \( \tau, \xi(A) \) is not defined at \( t \).

Hence \( \tau_3, \xi(A)(t) = a \) (for some \( a \)) and \( \tau, \xi(A) \) is not defined at \( t \)

The instant \( t \in \tau_3.u \)

Hence, \( t \in \tau.u \) (since \( \tau_3 \) is an extension of \( \tau \))

There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \in \sigma(r_1'; \mu) \) such that \( \tau = \tau_1, \langle \mu, \mu \rangle \)

Hence, if \( \tau_1, \xi(A) \) were defined at \( t \), then \( \tau, \xi(A) = (\tau_1, \xi(A))^{\tau.u} \) would be defined at \( t \) (because \( t \in \tau.u \))

Hence, \( \tau_1, \xi(A) \) is not defined at \( t \)

Choose a completion \( r_1' \) of \( r_1 \) where the tuple \( \tau_1' \in r_1' \) which is key equivalent to \( \tau_1 \in r_1 \) is as follows

(i) \( \tau_1', \ell = \tau_1.u = \tau_1.u \)

(ii) At instant \( t \) (where \( \tau_1, \xi(A) \) is not defined in \( r_1 \) but which is in \( \tau_1.u = \tau_1', u \)), \( \tau_1', \xi(A) = b \) (where \( b \neq a \)).

Then in \( \sigma(r_1'; \mu) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that \( \tau', \xi(A)(t) = b \)

(note that \( t \in \tau'.u \))

Hence, \( \tau' \) is not a completion of \( \tau_3 \) since they do not agree everywhere both are defined (specifically they do not agree at \( t \))

Hence, \( \tau' \in \sigma(r_1'; \mu) \) cannot be a completion of \( \tau_3 \).

Hence, \( \sigma(r_1'; \mu) \) cannot be a completion of \( r_3 \).

Selection of the form \( \sigma(r_1 ; [A]) \).

Since \( r_3 \) is a proper extension of \( \sigma(r_1 ; [A]) \) at least one of the following holds for some pair of key-equivalent tuples \( \tau \in \sigma(r_1 ; [A]) \) and \( r_3 \in r_3 

i) \( \tau.\ell \subset \tau_3.\ell \)

ii) \( \tau.u \supset \tau_3.u \)

iii) For some \( B \in R \) (\( B \) could be \( A \)), \( \tau_3, \xi(B) \) is defined somewhere in \( \tau_3.u \) where \( \tau, \xi(B) \) is not defined.

Case (i): \( \tau.\ell \subset \tau_3.\ell \)

There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \in \sigma(r_1 ; [A]) \) such that \( \tau = \tau_1^{\tau.\ell}, \langle r_1.\ell, \tau_1.u \rangle = \tau_1 \)

Hence, \( \tau.\ell = \tau_1.\ell \) (by definition of \( \sigma \))
Choose a completion $r'_1$ of $r_1$ where the tuple $r'_1 \in r'_1$ which is key equivalent to $r_1 \in r_1$ has
$$r'_1.l = r'_1.u = r_1.l$$

Then in $\sigma(r'_1; []; [A])$ there is a tuple $r'$ key-equivalent to $r$ such that
$$r'.l = r'_1.l \cap r'_1.l \quad \text{(by defn of } \sigma)$$

$$= r'_1.l$$
$$= r_1.l$$
$$= r.l$$
$$\subseteq \tau_3.l$$

Hence, $r' \in \sigma(r'_1; []; [A])$ cannot be a completion of $\tau_3$.

Thus $\sigma(r'_1; []; [A])$ cannot be a completion of $r_3$.

Case (ii): $\tau.u \cap \tau_3.u$

There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1; []; [A])$ such that $\tau = \tau_1.l(\tau_1.l, \tau_1.u) = \tau_1$

Hence, $\tau.u = \tau_1.u$ (by definition of $\sigma$)

Choose a completion $r'_1$ of $r_1$ where the tuple $r'_1 \in r'_1$ which is key equivalent to $r_1 \in r_1$ has
$$r'_1.l = r'_1.u = \tau_1.u$$

Then in $\sigma(r'_1; []; [A])$ there is a tuple $r'$ key-equivalent to $r$ such that
$$r'.u = r'_1.u \cap r'_1.u \quad \text{(by defn of } \sigma)$$

$$= r'_1.u$$
$$= \tau_1.u$$
$$= \tau.u$$
$$\subseteq \tau_3.u$$

Hence, $r' \in \sigma(r'_1; []; [A])$ cannot be a completion of $\tau_3$.

Thus $\sigma(r'_1; []; [A])$ cannot be a completion of $r_3$.

Case (iii): For some $B \in R$, $\tau_3.\xi(B)$ is defined at some instant $t$ in $r_3.u$ but in the key-equivalent tuple $\tau \in \sigma(r_1; []; [A])$, $\tau.\xi(B)$ is not defined at $t$.

Hence $\tau_3.\xi(B)(t) = b$ (for some $b$) and $\tau.\xi(B)$ is not defined at $t$

The instant $t \in r_3.u$

Hence, $t \in \tau.u$ (since $\tau_3$ is an extension of $\tau$)
There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1;[[A]])$ such that $\tau = \tau_1 \langle \tau_1.l, \tau_1.u \rangle$

i.e. $\tau = \tau_1$

Since $\tau.\xi(B)$ is not defined at $t$, hence $\tau_1.\xi(B)$ is defined at $t$

Choose a completion $r'_1$ of $r_1$ where the tuple $\tau'_1 \in r'_1$ which is key equivalent to $\tau_1 \in r_1$ is as follows

(i) $\tau'_1.l = \tau'_1.u = \tau_1.u$

(ii) At instant $t$ (where $\tau_1.\xi(B)$ is not defined in $r_1$ but which is in $\tau_1.u = \tau'_1.u$),

$$\tau'_1.\xi(B) = b_1 \text{ (where } b \neq b_1).$$

Then in $\sigma(r'_1;[[A]])$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$$\tau'.\xi(B)(t) = b_1 \text{ (note that } t \in \tau'.u)$$

Hence, $\tau'$ is not a completion of $\tau_3$ since they do not agree everywhere both are defined (specifically they do not agree at $t$)

Hence, $\tau' \in \sigma(r'_1;[[A]])$ cannot be a completion of $\tau_3$.

Hence, $\sigma(r'_1;[[A]])$ cannot be a completion of $r^1$.

Selection of the form $\sigma(r_1;[[A \theta b]])$.

We prove it for the case where $\theta$ is $\neq$ (the cases where $\theta$ is $<$, $>$, $\geq$, $\leq$ can be proved similarly).

Since $r_3$ is a proper extension of $\sigma(r_1;[[A \neq b]])$ at least one of the following holds for some pair of key-equivalent tuples $\tau \in \sigma(r_1;[[A \neq b]])$ and $\tau_3 \in r_3$

i) $\tau.l \subseteq \tau_3.l$

ii) $\tau.u \supseteq \tau_3.u$

iii) For some $C \in R$ ($C$ could be $A$), $\tau_3.\xi(C)$ is defined somewhere in $\tau_3.u$ where $\tau.\xi(C)$ is not defined.

Case (i): $\tau.l \subseteq \tau_3.l$

There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1;[[A \neq b]])$ such that

$$\tau.l = \tau_1.l \cap [[[A \neq b]](\tau_1).l] \text{ (by defn. of } \sigma)$$

$$= \tau_1.l \cap [[[\tau_1.\xi(A) \neq b]] \cap \tau_1.l \cap [0, \text{NOW}]) \text{ (by defn. of } [[[A \theta b]])$$

Choose a completion $r'_1$ of $r_1$ where the tuple $\tau'_1 \in r'_1$ which is key equivalent to $\tau_1 \in r_1$ has
(i) \( \tau_1'.l = \tau_1'.u = \tau_1.l \)

(ii) For every instant \( t \) in \( \tau_1'.l \) where \( \tau_1'.\xi(A) \) is undefined, \( \tau_1'.\xi(A)(t) = b \)

Then in \( \sigma(r_1'; [[A=b]]) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[
\tau'.l = \tau_1'.l \cap ([A\neq b]|[\tau_1'.\xi(A) = b])
\]

\[
= \tau_1'.l \cap ([\tau_1'.\xi(A) = b] \cap [\tau_1'.\xi(A) \neq b])
\]

\[
= \tau_1'.l \cap \tau_1.l \cap [\tau_1'.\xi(A) \neq b] \quad (\text{by construction})
\]

\[
= \tau_1.l \cap \tau_1.l \cap [\tau_1'.\xi(A) \neq b] \quad (\text{since } \tau_1'.l = \tau_1.l)
\]

\[
= \tau.l
\]

\( \in \tau_3.l \).

Hence, \( \tau' \in \sigma(r_1'; [[A\neq b]]) \) cannot be a completion of \( \tau_3 \).

Thus \( \sigma(r_1'; [[A\neq b]]) \) cannot be a completion \( r_3 \).

Case (ii): \( \tau.u \not\subseteq \tau_3.u \)

There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \in \sigma(r_1'; [[A\neq b]]) \) such that

\[
\tau.u = \tau_1.u \cap ([A\neq b]|(\tau_1.u)) \quad \text{(by defn. of } \sigma)\]

\[
= \tau_1.u \cap ((\tau_1.u \cap [0,\text{NOW}]) - [[\tau_1'.\xi(A) = b]]) \quad \text{(defn of } [A\neq b])\]

Choose a completion \( r_1' \) of \( r_1 \) where the tuple \( r_1' \in r_1' \) which is key equivalent to \( \tau_1 \in r_1 \) has

(i) \( \tau_1'.l = \tau_1'.u = \tau_1.u \)

(ii) For every instant \( t \) in \( \tau_1'.l \) (= \( \tau_1'.u \)) where \( \tau_1'.\xi(A) \) is undefined, \( \tau_1'.\xi(A)(t) = c \) (where \( c \neq b \))

Then in \( \sigma(r_1'; [[A\neq b]]) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[
\tau'.u = \tau_1'.u \cap ((\tau_1'.u \cap [0,\text{NOW}]) - [[\tau_1'.\xi(A) = b]])
\]

\[
= \tau_1'.u - ([\tau_1'.\xi(A) = b] \cap \tau_1'.u)
\]

\[
= \tau_1'.u - ([\tau_1'.\xi(A) = b] \cap \tau_1'.u) \quad (\text{by construction})
\]

\[
= \tau_1'.u - ([\tau_1'.\xi(A) = b] \cap \tau_1'.u)
\]

\[
= \tau.u
\]

\( \in \tau_3.u \)

Hence, \( \tau' \in \sigma(r_1'; [[A\neq b]]) \) cannot be a completion of \( \tau_3 \).

Thus \( \sigma(r_1'; [[A\neq b]]) \) cannot be a completion \( r_3 \).
Case (iii): For some $C \in R$, $\zeta(C)$ is defined at some instant $t$ in $\tau_3.u$ but in the key-equivalent tuple $\tau \in \sigma(\tau_1; [[A#b]])$, $\zeta(C)$ is not defined at $t$.

Hence $\tau_3.\zeta(C)(t) = c$ (for some $c$) and $\tau.\zeta(C)$ is not defined at $t$

There is a tuple $\tau_1 \in \tau_1$ key-equivalent to $\tau \in \sigma(\tau_1; [[A#b]])$ such that

$$\tau.\zeta(A) = \tau_1.\zeta(A) \upharpoonright \tau.u$$

$$= (\tau_1.\zeta(A))\upharpoonright (\tau_1.u - (\tau_1.u \cap ([\tau_1.\zeta(A) = b]$$

Now $t \in \tau_3.u$. Hence $t \in \tau.u$. Hence $t \in \tau_1.u$

Since $\tau.\zeta(C)$ is not defined at $t$, hence $\tau_1.\zeta(C)$ is not defined at $t$

Choose a completion $r'_1$ of $r_1$ where the tuple $r'_1 \in r'_1$ which is key equivalent to $r_1 \in r_1$ is as follows

(i) $\tau'_1.c = \tau'_1.u = \tau_1.u$

(ii) For every instant $t'$ in $\tau'_1.\ell (= \tau'_1.u)$ where $\tau_1.\zeta(C)$ is not defined, $\tau'_1.\zeta(C)(t') = c_1(\#c)$.

(iii) For every instant $t'$ in $\tau'_1.\ell (= \tau'_1.u)$ where $\tau_1.\zeta(A)$ is undefined, $\tau'_1.\zeta(A)(t') = b_1(\#b)$

Then in $\sigma(\tau'_1; [[A#b]])$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$$\tau'.\zeta(C) = \tau'_1.\zeta(C) \upharpoonright (\tau'.u) \text{ (note that } \tau'.\zeta(A) \text{ is defined at } t)$$

$$= \tau'_1.\zeta(C)(t)$$

Hence $\tau'.\zeta(C)(t) = \tau'_1.\zeta(C)(t)$

and since $t \in \tau.u$

$$\tau'.\zeta(C)(t) = \tau'_1.\zeta(C)(t)$$

$$= c_1(\#c)$$

Hence, $\tau'$ is not a completion of $\tau_3$ since they do not agree everywhere both are defined (specifically they do not agree at $t$)

Hence, $\tau' \in \sigma(\tau'_1; [[A#b]])$ cannot be a completion of $\tau_3$.

Hence, $\sigma(\tau'_1; [[A#b]])$ cannot be a completion of $r_3$.

The proof is similar for cases where $\theta$ is $<, >, \leq, \geq$.

Selection of the form $\sigma(\tau_1; [[A#B]])$.

The proof is similar to $\sigma(\tau_1; [[A#b]])$

Selection of the form $\sigma(\tau_1; u \subseteq [[A]])$.

Since $r_3$ is a proper extension of $\sigma(\tau_1; u \subseteq [[A]])$ at least one of the following holds for some pair of key-equivalent tuples $\tau \in \sigma(\tau_1; u \subseteq [[A]])$ and $\tau_3 \in r_3$
i) \( \tau_e \subseteq \tau_u \)

ii) \( \tau_u \cup \tau_u \)

iii) For some \( B \in R \) (\( B \) could be \( A \)), \( \tau^3 \xi(B) \) is defined somewhere in \( \tau_u \) where \( \tau \xi(B) \) is not defined.

Case (i): \( \tau_e \subseteq \tau_u \)

There are two sub-cases

(a) There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \in \sigma(\tau_1; \mu \subseteq [A]) \) such that \( \text{eval}(\mu \subseteq [A])(\tau_1) = \langle 1, 1 \rangle \) i.e. \( \mu \subseteq \tau_1 \).

(b) There is a tuple \( \tau_1 \in r_1 \) key-equivalent to \( \tau \in \sigma(\tau_1; \mu \subseteq [A]) \) such that \( \text{eval}(\mu \subseteq [A])(\tau_1) = \langle 0, 1 \rangle \) i.e. \( \mu \notin \tau_1 \).

Subcase (a) In this case \( \tau = \tau_1 \)

Hence \( \tau_e = \tau_1 \).

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has

\[ \tau'_1 \subseteq \tau'_1 = \tau_1. \]

Then \( \text{eval}(\mu \subseteq [A])(\tau'_1) = \langle 1, 1 \rangle \) since \( \mu \subseteq \tau_1 \).

Then in \( \sigma(\tau'_1; \mu \subseteq [A]) \), there is a tuple \( \tau' \) key-equivalent to \( \tau' \) such that

\[ \tau'_1 \subseteq \tau'_1 = \tau_1. \]

Hence, \( \tau' \in \sigma(\tau'_1; \mu \subseteq [A]) \) cannot be a completion of \( \tau_1 \).

Thus \( \sigma(\tau'_1; \mu \subseteq [A]) \) cannot be a completion of \( \tau_3 \).

Subcase (b) In this case \( \mu \notin \tau_1 \).

Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has

\[ \tau'_1 \subseteq \tau'_1 = \tau_1. \]

Then \( \text{eval}(\mu \subseteq [A])(\tau'_1) = \langle 0, 0 \rangle \) since \( \mu \notin \tau_1 \).

Then in \( \sigma(\tau'_1; \mu \subseteq [A]) \), there is no tuple \( \tau' \) key-equivalent to \( \tau_3 \) since \( \tau_3 \subseteq \tau'_1 \).

Hence, \( \tau' \notin \sigma(\tau'_1; \mu \subseteq [A]) \) cannot be a completion of \( \tau_3 \).
Case (ii): $\tau_4 \cup \tau_3 \cup u$

There are two sub-cases

(a) There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1; \mu \subseteq [A])$ such that $\text{eval}(\mu \subseteq [A])(\tau_1) = \langle I, I \rangle$ i.e. $\mu \subseteq \tau_1 \cdot \ell$

(b) There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1; \mu \subseteq [A])$ such that $\text{eval}(\mu \subseteq [A])(\tau_1) = \langle \emptyset, I \rangle$ i.e. $\mu \nsubseteq \tau_1 \cdot \ell$ but $\mu \nsubseteq \tau_1 \cdot u$

Subcase (a) In this case $\tau = \tau_{1}$

Hence $\tau \cdot u = \tau_{1} \cdot u$

Choose a completion $r_{1}'$ of $r_1$ where the tuple $\tau_{1}' \in r_{1}'$ which is key equivalent to $\tau_{1} \in r_1$ has

$$\tau_{1}' \cdot \ell = \tau_{1}' \cdot u = \tau_{1} \cdot u$$

Then $\text{eval}(\mu \subseteq [A])(\tau_{1}') = \langle I, I \rangle$ since $\mu \subseteq \tau_{1} \cdot \ell$ and so $\mu \subseteq \tau_{1}' \cdot \ell$

Then in $\sigma(r_1; \mu \subseteq [A];)$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$$\tau' \cdot u = \tau_{1}' \cdot u$$

(by defn of $\sigma$)

$$= \tau_{1} \cdot u$$

$$= \tau_{1} \cdot u$$

$$= \tau \cdot u$$

$$= \tau \cdot u$$

Hence, $\tau' \in \sigma(r_1; \mu \subseteq [A];)$ cannot be a completion of $\tau_{1}' \ni r_{1}'$.

Thus $\sigma(r_1; \mu \subseteq [A];)$ cannot be a completion $r_{3}' \in r_3$.

Subcase (b) Thus $\mu \subseteq \tau_{1} \cdot u$ and $\tau \cdot u = \tau_{1} \cdot u$

Choose a completion $r_{1}'$ of $r_1$ where the tuple $\tau_{1}' \in r_{1}'$ which is key equivalent to $\tau_{1} \in r_1$ has

$$\tau_{1}' \cdot \ell = \tau_{1}' \cdot u = \tau_{1} \cdot u$$

Then $\text{eval}(\mu \subseteq [A])(\tau_{1}') = \langle I, I \rangle$ since $\mu \subseteq \tau_{1} \cdot u$ and so $\mu \subseteq \tau_{1}' \cdot \ell$

Then in $\sigma(r_1; \mu \subseteq [A];)$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that

$$\tau' \cdot u = \tau_{1} \cdot u$$

(by defn of $\sigma$)

$$= \tau_{1} \cdot u$$

$$= \tau_{1} \cdot u$$

$$= \tau \cdot u$$

$$= \tau \cdot u$$

Hence, $\tau' \in \sigma(r_1; \mu \subseteq [A];)$ cannot be a completion of $\tau_{1}' \ni r_{1}'$.

Thus $\sigma(r_1; \mu \subseteq [A];)$ cannot be a completion $r_{3}' \in r_3$.

Case (iii): For some $B \in R$, $\tau_{3} \cdot \xi(B)$ is defined at some instant $t$ in $\tau_{3} \cdot u$ but in the
key-equivalent tuple $\tau \in \sigma(r_1;\mu \subseteq [A]_b)$, $\tau.\xi(B)$ is not defined at $t$.
Hence $\tau.\xi(B)(t) = b$ (for some $b$) and $\tau.\xi(B)$ is not defined at $t$
The instant $t \in \tau.u$
Hence, $t \in \tau.u$ (since $\tau$ is an extension of $\tau$)
There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1;\mu \subseteq [A]_b)$ such that $\tau_1.\xi(A) = \tau.\xi(A)$
Hence, since $\tau.\xi(B)$ is not defined at $t$, so $\tau_1.\xi(B)$ is not defined at $t$
Choose a completion $r_1'$ of $r_1$ where the tuple $\xi \in r_1$ which is key equivalent to $\xi \in r_1$ is as follows
(i) $\tau_1'.l = \tau_1.u = \tau_1.u$
(ii) At instant $t$ (where $\tau_1.\xi(B)$ is not defined in $r_1$ but which is in $\tau_1.u = \tau_1'.u$),
$\tau_1'.\xi(B) = b_1$ (where $b \neq b_1$).
Then in $\sigma(r_1;\mu \subseteq [A]_b)$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that
$\tau'.\xi(B)(t) = b_1$ (note that $t \in \tau'.u$)
Hence, $\tau'$ is not a completion of $\tau$ since they do not agree everywhere both are defined (specifically they do not agree at $t$)
Hence, $\sigma(r_1;\mu \subseteq [A]_b)$ cannot be a completion of $\tau$.

Selection of the form $\sigma(r_1;\mu \subseteq [A,\theta b])$.
We prove the case where $\theta$ is $\neq$. The cases where $\theta$ is $<,\leq,>,\geq$ can be proved similarly.
Since $\tau$ is a proper extension of $\sigma(r_1;\mu \subseteq [A,\theta b])$ at least one of the following holds for
some pair of key-equivalent tuples $\tau \in \sigma(r_1;\mu \subseteq [A,\theta b])$ and $\tau_3 \in r_3$

i) $\tau.l \subseteq \tau_3.l$

ii) $\tau.u \supset \tau_3.u$

iii) For some $C \in R$ ($C$ could be $A$), $\tau_3.\xi(C)$ is defined somewhere in $\tau_3.u$ where $\tau.\xi(C)$ is not defined.

Case (i): $\tau.l \subseteq \tau_3.l$
There are two sub-cases
(a) There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1;\mu \subseteq [A,\theta b])$ such that
eval($\mu \subseteq [A,\theta b])($($\tau_1.l = \{I, I\}$ i.e. $\mu \subseteq ([A,\theta b])($($\tau_1.l = \{I, I\}$
(b) There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1;\mu \subseteq [A,\theta b])$ such that
eval($\mu \subseteq [A,\theta b])($($\tau_1.l = \{\emptyset, I\}$ i.e. $\mu \subseteq ([A,\theta b])($($\tau_1.l = \{\emptyset, I\}$ but $\mu \subseteq ([A,\theta b])($($\tau_1.l = \{\emptyset, I\}$ u
Subcase (a) In this case \( \tau = \tau_1 \)

Choose a completion \( \tau'_1 \) of \( \tau_1 \) where the tuple \( \tau'_1 \in \tau'_1 \) which is key equivalent to \( \tau_1 \in \tau_1 \) has \( \tau'_1.\ell = \tau'_1.u = \tau_1.\ell \) and for every instant \( t \) in \( \tau'_1.\ell \) where \( \tau_1.\xi(A) \) is undefined, \( \tau'_1.\xi(A)(t) = b \)

Thus \( \text{eval}(\mu \subseteq [A#b])(\tau'_1) = (1,1) \) (by Lemma 5.3)

Thus in \( \sigma(\tau'_1; \mu \subseteq [A#b]) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that

\[
\tau'.\ell = \tau'_1.\ell \quad \text{(by def of } \sigma) \\
= \tau_1.\ell \\
= \tau.\ell \\
\subseteq \tau_3.\ell
\]

Hence, \( \tau' \in \sigma(\tau'_1;\mu \subseteq [A#b]) \) cannot be a completion of \( \tau_3 \in \tau_3 \).

Thus \( \sigma(\tau'_1;\mu \subseteq [A#b]) \) cannot be a completion \( \tau_3 \).

Subcase (b)

Choose a completion \( \tau'_1 \) of \( \tau_1 \) where the tuple \( \tau'_1 \in \tau'_1 \) which is key equivalent to \( \tau_1 \in \tau_1 \) has \( \tau'_1.\ell = \tau'_1.u = \tau_1.\ell \) and for every instant \( t \) in \( \tau'_1.\ell \) where \( \tau_1.\xi(A) \) is undefined, \( \tau'_1.\xi(A)(t) = b \)

\[
[A#b](\tau_1).\ell = \tau'_1.\xi(A) \not\subseteq b \) \cap (\tau_1.\ell) \cap [0,\text{NOW}]
= \tau'_1.\xi(A) \not\subseteq b \) \cap (\tau'_1.\ell) \cap [0,\text{NOW}]
= [A#b](\tau'_1).\ell
= [A#b](\tau'_1).u \text{ (since } \tau'_1 \text{ is a completion)}
\]

Since \( \mu \not\subseteq [A#b](\tau_1).\ell \) hence \( \mu \not\subseteq [A#b](\tau'_1).u \)

Hence, \( \text{eval}(\mu \subseteq [A#b])(\tau'_1) = (\emptyset,\emptyset) \)

Hence, in \( \sigma(\tau'_1;\mu \subseteq [A#b]) \) there is no tuple key-equivalent to \( \tau_3 \in \tau_3 \) and since \( \tau.\ell \subseteq \tau_3.\ell \) we know that \( \tau_3.\ell \neq \emptyset \)

Thus \( \sigma(\tau'_1;\mu \subseteq [A#b]) \) cannot be a completion \( \tau_3 \).

Case (ii): \( \tau.\ell \cap \tau_3.\ell \)

There are two sub-cases

(a) There is a tuple \( \tau_1 \in \tau_1 \) key-equivalent to \( \tau \in \sigma(\tau'_1;\mu \subseteq [A#b]) \) such that

\[
\text{eval}(\mu \subseteq [A#b])(\tau_1) = (1,1) \quad \text{i.e. } \mu \subseteq ([A#b])(\tau_1).\ell
\]

(b) There is a tuple \( \tau_1 \in \tau_1 \) key-equivalent to \( \tau \in \sigma(\tau'_1;\mu \subseteq [A#b]) \) such that

\[
\text{eval}(\mu \subseteq [A#b])(\tau_1) = (\emptyset,1) \quad \text{i.e. } \mu \not\subseteq ([A#b])(\tau_1).\ell \text{ but } \mu \subseteq ([A#b])(\tau_1).u
\]
Subcase (a) In this case \( \tau = \tau_1 \)
Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has \( \tau'_1.\ell = \tau'_1.u = \tau_1.u \) and for every instant \( t \) in \( \tau'_1.\u \) where \( \tau_1.\xi(A) \) is undefined, \( \tau'_1.\xi(A)(t) = b_1(\#b) \)
Thus \( \text{eval}(\mu \subseteq \mathbb{[A\#b]})(\tau'_1) = (1,1) \) (by Lemma 5.3)
Thus in \( \sigma(r'_1; \mu \subseteq \mathbb{[A\#b]}_3) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that
\[ \tau'.u = \tau'_1.u \] (by defn of \( \sigma \))
\[ = \tau_1.u \]
\[ = \tau.u \]
\[ \cup \tau_3.u \]
Hence, \( \tau' \in \sigma(r'_1; \mu \subseteq \mathbb{[A\#b]}_3) \) cannot be a completion of \( \tau_3 \in r_3 \).
Thus \( \sigma(r'_1; \mu \subseteq \mathbb{[A\#b]}_3) \) cannot be a completion \( r_3 \).

Subcase (b)
Choose a completion \( r'_1 \) of \( r_1 \) where the tuple \( \tau'_1 \in r'_1 \) which is key equivalent to \( \tau_1 \in r_1 \) has \( \tau'_1.\ell = \tau'_1.u = \tau_1.u \) and for every instant \( t \) in \( \tau'_1.\u \) where \( \tau_1.\xi(A) \) is undefined, \( \tau'_1.\xi(A)(t) = b_1(\#b) \)
\[ \mathbb{[A\#b]}(\tau_1).u = (\tau_1.u \cap [0,\text{NOW}]) - \mathbb{[\tau_1.\xi(A) = b]} \]
\[ = (\tau'_1.u \cap [0,\text{NOW}]) - \mathbb{[\tau'_1.\xi(A) \neq b]} \]
\[ = \mathbb{[A\#b]}(\tau'_1).u \]
\[ = \mathbb{[A\#b]}(\tau'_1).\ell \] (since \( \tau'_1 \) is a completion)
Since \( \mu \subseteq \mathbb{[A\#b]}(\tau_1).u \) hence \( \mu \subseteq \mathbb{[A\#b]}(\tau'_1).\ell \)
Hence, \( \text{eval}(\mu \subseteq \mathbb{[A\#b]})(\tau'_1) = (1,1) \)
Hence, in \( \sigma(\tau'_1; \mu \subseteq \mathbb{[A\#b]}_3) \) there is a tuple \( \tau' \) key-equivalent to \( \tau \) such that
\[ \tau'.u = \tau'_1.u \]
\[ = \tau_1.u \]
\[ = \tau.u \]
\[ \cup \tau_3.u \]
Hence \( \tau' \) is not a completion of \( \tau_3 \).
Thus \( \sigma(\tau'_1; \mu \subseteq \mathbb{[A\#b]}_3) \) cannot be a completion \( r_3 \).

Case (iii): For some \( C \in R \), \( \tau_3.\xi(C) \) is defined at some instant \( t \) in \( \tau_3.\u \) but in the key-equivalent tuple \( \tau \in \sigma(r_1; \mu \subseteq \mathbb{[A\#b]}_3) \), \( \tau.\xi(C) \) is not defined at \( t \).
Hence \( \tau_3.\xi(C)(t) = c \) (for some \( c \)) and \( \tau.\xi(C) \) is not defined at \( t \).
The instant $t \in \tau_3.\ u$. Hence, $t \in \tau.\ u$ (since $\tau_3$ is an extension of $\tau$)
There is a tuple $\tau_1 \in r_1$ key-equivalent to $\tau \in \sigma(r_1; \mu \subseteq [A \# b]]$ such that $\tau_1.\xi(C) = \tau.\xi(C)$
Hence, since $\tau.\xi(C)$ is not defined at $t$, so $\tau_1.\xi(C)$ is not defined at $t$
Choose a completion $r'_1$ of $r_1$ where the tuple $\tau'_1 \in r'_1$ which is key equivalent to $\tau_1 \in r_1$ is as follows
(i) $\tau'_1.\ell = r'_1.\mu = r_1.\mu$
(ii) At every instant $t'$ in $r'_1.\ u$ where $\tau_1.\xi(A)$ is undefined let $\tau'_1.\xi(A) = b_1(\#b)$
(iii) At instant $t$ (where $\tau_1.\xi(C)$ is not defined in $r_1$ but which is in $r_1.\ u = \tau_1.\ u$), $\tau_1.\xi(C)(t) = c_1$ (where $c \neq c_1$).
Hence eval($\mu \subseteq [A \# b]](r'_1) = \langle 1, 1 \rangle$ (see case (ii) subcase a,b)
Then in $\sigma(r_1; \mu \subseteq [A \# b]]$ there is a tuple $\tau'$ key-equivalent to $\tau$ such that $\tau'.\xi(C)(t) = c_1$ (note that $t \in \tau'.\ u$)
Hence, $\tau'$ is not a completion of $\tau_3$ since they do not agree everywhere both are defined (specifically they do not agree at $t$)
Hence, $\tau' \in \sigma(r_1; \mu \subseteq [A \# b]]$ cannot be a completion of $\tau_3$.
Hence, $\sigma(r_1; \mu \subseteq [A \# b]]$ cannot be a completion of $r_3$.

Selection of the form $\sigma(r_1; \mu \subseteq [A \& B]]$. similar to $\sigma(r_1; \mu \subseteq [A \& B]]$.

Restructuring.

Since $r_3$ is a proper extension of $I_K(r_1)$ at least one of the following holds for some pair of key-equivalent tuples $\tau \in I_K(r_1)$ and $\tau_3 \in r_3$
    i) $\tau.\ell \subseteq \tau_3.\ell$
    ii) $\tau.\mu \subseteq \tau_3.\mu$
    iii) For some $A \in R$, $\tau_3.\xi(A)$ is defined somewhere in $\tau_3.\ u$ where $\tau.\xi(A)$ is not defined.

Case (i): $\tau.\ell \subseteq \tau_3.\ell$

Choose a completion $r'_1$ of $r_1$ s.t.
for each tuple $\tau_1 \in r_1$ if $\tau_1.\ell \neq \emptyset$ there is a tuple $\tau'_1$ in $r'_1$ such that $\tau'_1.\ell = r'_1.\ell = r_1.\ell$
for each tuple $\tau_1 \in r_1$ if $\tau_1.\ell = \emptyset$ there is no key-equivalent tuple in $r'_1$. 

Then if \( r.t = \emptyset \) there is no key-equivalent tuple in \( I_K(r'_1) \) but since \( r.t \subseteq r_3.t \), \( r_3.t \neq \emptyset \). So there is no key-equivalent tuple in \( I_K(r'_1) \) although \( r_3.t \neq \emptyset \).

If \( r.t \neq \emptyset \) there is a key-equivalent tuple in \( I_K(r'_1) \) such that \( r'.t = r.t \subseteq r_3.t \).

Hence \( r' \) is not a completion of \( r_3 \).

Thus \( I_K(r'_1) \) cannot be a completion \( r_3 \).

Case (ii): \( r.u \cap r_3.u \)

Construct a completion \( r'_1 \) of \( r_1 \) s.t.

for each tuple \( r_1 \in r_1 \) if \( r_1.t \neq \emptyset \) there is a tuple \( r'_1 \) in \( r'_1 \) such that \( r'_1.t = r_1.t \)

Then for the tuple \( r \in I_K(r'_1) \) there is a key-equivalent tuple \( r' \) in \( I_K(r'_1) \) such that \( r'.u = r.u \cup r_3.u \).

Hence \( r' \) is not a completion of \( r_3 \).

Thus \( I_K(r'_1) \) cannot be a completion \( r_3 \).

Case (iii): For some \( A \in R \), \( r_3.\xi(A) \) is defined somewhere in \( r_3.u \) where \( r.\xi(A) \) is not defined.

Then at some instant \( t \), \( r_3.\xi(A) = a \) (for some \( a \)) but \( r.\xi(A) \) is not defined at \( t \).

Construct a completion \( r'_1 \) of \( r_1 \) s.t.

for each tuple \( r_1 \in r_1 \) there is a tuple \( r'_1 \) in \( r'_1 \) such that \( r'_1.t = r_1.t \).

At instant \( t \) if \( t \in r'_1.u \) and \( r_1.\xi(A) \) is not defined at \( t \) then \( r'_1.\xi(A)(t) = a_1 \)

Then there is a \( r' \in I_K(r'_1) \) key-equivalent to \( r \) such that \( r' \) and \( r_3 \) do not agree everywhere they are defined (specifically they do not agree at \( t \)).

Hence \( r' \) is not a completion of \( r_3 \).

Thus \( I_K(r'_1) \) cannot be a completion \( r_3 \).

This theorem shows that if the results of the operations listed above were extended to be more informative then we would lose the property of reliability of our results.

Example 8.1. Theorem 8.2 does not hold for selection operations of the form \( \sigma(r; [[A\theta b]]) \) etc where \( \theta \) is \( = \). Let \( r \) be the relation over the scheme \( AB \) with key \( A \) as shown in Figure 8.1(a). Then the result of \( \sigma(r; [[B=b]]) \) is shown in Figure 8.1(b). The relation \( r_3 \) shown in Figure 8.1(c) is an extension of \( \sigma(r; [[B=b]]) \) (the term extension was defined in Section 5). However, for every completion \( r' \) of \( r, \sigma(r'; [[B=b]]) \) will be

\[ \]

\[ \]

\[ \]
a completion of $r_3$. Note that this selection could be easily defined so that $r_3$ is in fact the result of the selection. In that case, we would be paralleling [Bi83].

We further remark that the case $\sigma(r; ;[[B=b]])$ would work if $\text{dom}(B)$ has at least three elements. We have implicitly assumed this to be the case. Similarly, for $\sigma(r; ;[[B< b]])$ we assume there are at least two elements $< b$ and for $\sigma(r; ;[[B>b]])$ we assume there are at least two elements $> b$.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,10]a</td>
<td>[0,5]b</td>
</tr>
<tr>
<td>[0,5]</td>
<td>[0,5]</td>
</tr>
<tr>
<td>[0,10]</td>
<td>[0,10]</td>
</tr>
</tbody>
</table>

(a) The relation $r$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,10]a</td>
<td>[0,5]b</td>
</tr>
<tr>
<td>[0,5]</td>
<td>[0,5]</td>
</tr>
<tr>
<td>[0,10]</td>
<td>[0,10]</td>
</tr>
</tbody>
</table>

(b) $\sigma(r; ;[[B=b]])$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,10]a</td>
<td>[0,10]b</td>
</tr>
<tr>
<td>[0,5]</td>
<td>[0,5]</td>
</tr>
<tr>
<td>[0,10]</td>
<td>[0,10]</td>
</tr>
</tbody>
</table>

(c) The relation $r_3$

Figure 8.1. A counterexample for $\sigma(r; ;[[B=b]])$

Example 8.2. Theorem 8.2 does not hold for arbitrary algebraic operations. Let $r$ be the relation over the scheme $AB$ with key $A$ as shown in Figure 8.2(a). Then the result of $\sigma(\sigma(r; ;[[B< b]]); ;[[B>b]])$ is shown in Figure 8.2(b). Let the relation $r_3$ be the empty relation over $AB$. Then $r_3$ is an extension of $\sigma(\sigma(r; ;[[B< b]]); ;[[B>b]])$. However, for every completion $r'$ of $r$, $\sigma(\sigma(r; ;[[B< b]]); ;[[B>b]])$ will be empty and hence a completion of $r_3$. □

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0,10]a</td>
<td>[0,5]b</td>
</tr>
<tr>
<td>[0,5]</td>
<td>[0,5]</td>
</tr>
<tr>
<td>[0,10]</td>
<td>[0,10]</td>
</tr>
</tbody>
</table>

(a) The relation $r$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6,10]a</td>
<td>∅</td>
</tr>
<tr>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>[6,10]</td>
<td>[6,10]</td>
</tr>
</tbody>
</table>

(b) $\sigma(\sigma(r; ;[[B< b]]); ;[[B>b]])$

Figure 8.2. A counterexample for $\sigma(\sigma(r; ;[[B< b]]); ;[[B>b]])$
9. UPDATES FOR TEMPORAL DATABASES
WITH INCOMPLETE INFORMATION

In this section we formally define the semantics of update operations that enable us to explicitly change the state of the database. In classical relational databases, the operations insert, delete and modify are used to insert, delete and modify tuples in a relation. However, for the temporal model for complete information these operations have to be redefined since each attribute value is now no longer assigned a single atomic value from a domain but rather a function \( \xi \) from time instants to domain values [Bh89].

These operations have to be further extended in our model since now each attribute is of the form \( \xi \ell u \) where \( \xi \) is a temporal assignment and \( \ell \) and \( u \) are temporal elements. Thus, not only can values be modified but instants at which we did not know the values before can now be assigned values (and vice versa i.e. values at some instants may be in error and we may not know the correct values. Also, the values \( \ell \) and \( u \) may change as our knowledge becomes more (or less) certain. We note that the fact that the \( \ell \) values (and the \( u \) values) are the same for each attribute in a tuple allows us to simplify the syntax of these operations as well as the description of the semantics. Our update operations generalize the update operations in [Bh89] for temporal databases with complete information.

We first introduce some auxiliary definitions that will be used to describe the semantics of the operations. An extended temporal assignment \( \xi \) to an attribute \( A \) is a function from a temporal element \( \mu \) such that for all \( t \in \mu \), \( \xi(t) \in \text{dom}(A) \cup \{\bot\} \). For an extended temporal assignment \( \xi \) we define \( [[\xi]] = \{t: \xi \text{ is defined at } t \text{ and } \xi(t) \neq \bot\} \). For an extended temporal assignment we define \( [\xi] = \{t: \xi \text{ is defined at } t\} \). For a temporal assignment \( \xi \) we define \( [\xi] = \{a: \xi(t) = a \text{ for some } a \in \text{dom}(A)\} \).

The operations we define are the create, change and changekey operations. The create operation allows us to create a new object in a relation. The change operation allows us to modify values of non-key attributes, to modify the domain of the temporal
assignment in partial temporal assignments to attributes, and to modify the \( \ell \) value and \( u \) value of a tuple. It also allows us to delete a tuple from a relation. The changekey operation allows us to modify values in the key attributes of tuples.

9.1. The Create Operation

The create operation allows the user to create a new object in a relation \( r \). Suppose a relation \( r \) has a scheme \( R = A_1A_2...A_n \). The syntax of the create operation is as follows:

\[
\text{create } (A_1:\xi_1; A_2:\xi_2;...;A_n:\xi_n; \ell, u) \text{ in } r
\]

In the create operation specified above, \( \xi_1, \xi_2,...,\xi_n \) are temporal assignments and \( \ell \) and \( u \) are temporal elements. The following conditions must be satisfied for the operation to be accepted: (i) \( [\xi_i] \subseteq u \) for each \( i, 1 \leq i \leq n \), (ii) \( \ell \subseteq u \), and (iii) if \( A_j, 1 \leq j \leq n \), is a key attribute then \( [\xi_j] = u \) and \( |\xi_j| \) is a singleton set. The tuple \( r = \langle A_1:\xi_1 \ell u; A_2:\xi_2 \ell u;...; A_n:\xi_n \ell u \rangle \) is added to the relation \( r \) if \( r \) is not key-equivalent to any tuple already present in \( r \), otherwise it is rejected by the system.

Example 9.1. The create operation specified below creates the tuple for John in the emp relation shown in Figure 5.2.

\[
\text{create } (\text{NAME}; \{[0,100] \rightarrow \text{John}\}; \\
\text{SALARY}; \{[10,40] \rightarrow 30K, [41,45] \rightarrow 40K\}; \\
\text{DEPT}; \{[10,30] \rightarrow \text{Toys}, [31,55] \rightarrow \text{Shoes}\}; \\
\ell = [0,50], u = [0,100]) \\
\text{in } \text{emp} \\
\]

Example 9.2. The create operation specified below creates the tuple shown in Figure 9.1. in the emp relation.

\[
\text{create } (\text{NAME}; \{[70,\text{NOW}] \rightarrow \text{Mary}\}; \\
\text{SALARY}; \{[70,\text{NOW}] \rightarrow 30K\}; \\
\text{DEPT}; \{[70,75] \rightarrow \text{Toys}\}; \\
\ell = [70,\text{NOW}], u = [70,\text{NOW}]) \\
\text{in } \text{emp} \\
\]
<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>[70,NOW]Mary</td>
<td>[70,NOW]30K</td>
<td>[70,75]Toys</td>
</tr>
<tr>
<td>[70,NOW]</td>
<td>[70,NOW]</td>
<td>[70,NOW]</td>
</tr>
<tr>
<td>[70,NOW]</td>
<td>[70,NOW]</td>
<td>[70,NOW]</td>
</tr>
</tbody>
</table>

Figure 9.1 Mary's tuple in emp relation

Example 9.3. The following create operation will be rejected by the system since \([\text{NAME}] \notin u\).

\[
\text{create (NAME: ([80,NOW]-Harry);}
\text{SALARY: ([80,NOW]-50K);}
\text{DEPT: ([70,75]-Toys);}
\text{\(t = [70,\text{NOW}], u = [70,\text{NOW}]\)})}
\]

\[
in \text{emp}
\]

Example 9.4. The following create operation will be rejected by the system since a tuple for Mary already exist in the relation emp.

\[
\text{create (NAME: ([80,NOW]-Mary);}
\text{SALARY: ([80,NOW]-50K);}
\text{DEPT: ([70,75]-Toys);}
\text{\(t = [70,\text{NOW}], u = [80,\text{NOW}]\)})}
\]

\[
in \text{emp}
\]

9.2 The Change Operation

The change operation can be used for the following purposes:

(i) to modify values of non-key attributes of a tuple (this includes modifications because of changes in the real world and those due to errors in our knowledge of past or present values). This also includes assigning values to the attributes at instants at which previously the attribute was undefined.

(ii) modify the temporal domain of the \(\xi\) part of any attribute

(iii) to modify the \(\xi\) values and \(u\) values in a tuple

(iv) to delete a tuple.
Since our knowledge of the $l$ values or $u$ values in a given tuple may be incorrect, we must provide a mechanism to update them. Sometimes we may need to set them to a particular temporal element. At other times we may just need to increase or decrease it by a constant amount. We first introduce some additional definitions that will be used to describe the semantics of the change operation. An $l$-$exp$ is of the form $\mu, l \cup \mu, l \cap \mu$ and $l - \mu$ where $\mu$ is a temporal element. Given a tuple $\tau$, an $l$-$exp$ is evaluated using the following rules: $\mu(\tau) = \mu, (l \cup \mu)(\tau) = \tau \cup \mu, (l \cap \mu)(\tau) = \tau \cap \mu$ and $(l - \mu)(\tau) = \tau - \mu$. A $u$-$exp$ and its evaluation are defined similarly.

Suppose the key of a relation $r$ is $A_1 A_2 \ldots A_k$. Suppose $B_1 \ldots B_n \subseteq R - A_1 \ldots A_k$. Then a change operation is of the form shown below:

change $(A_1 : a_1; A_2 : a_2; \ldots; A_k : a_k)$ to $(\xi_1; \xi_2; \ldots; \xi_n)$ \[ l = l$-$exp \] \[ u = u$-$exp \]

in $r$.

In the above operation each $\xi_m, 1 \leq m \leq n$, is an extended assignment. The operation transforms the tuple $\tau$ with key $(a_1, a_2, \ldots, a_k)$ to a tuple $\tau'$ as follows:

(i) $\tau'.l = l$-$exp(\tau), \tau'.u = u$-$exp(\tau)$

(ii) For each key attribute $A_i, 1 \leq i \leq k, \tau'.\xi(A_i) = \tau'.u-a_i$.

(iii) For each non-key attribute $B$ not in the list $B_1 B_2 \ldots B_n$

$\tau'.\xi(B) = \tau.\xi(B)$

(iv) For each non-key attribute $B_m, 1 \leq m \leq n,$

if $t \in [\xi_m]$ then $\tau'.\xi(B_m)(t) = \xi_m(t)$

else if $t \in \tau.\xi(B_m) - [\xi_m]$ then $\tau'.\xi(B_m)(t) = \tau.\xi(B_m)(t)$

else $\tau'.\xi(B_m)(t)$ is not defined at $t$.

The operation is accepted if $\tau'$ is a valid tuple i.e. if the following conditions are satisfied: (i) $\tau'.l \subseteq \tau'.u$ and (ii) for each attribute $A [\tau'.\xi(A)] \subseteq \tau'.u$. Otherwise the update is rejected by the system.

**Example 9.5.** Suppose Mary got a raise at time $t = 80$ so that her new $\text{SALARY}$ is $40K$. The change operation specified below incorporates this change. The tuple shown in Figure 9.1 is modified to the tuple shown in Figure 9.2.

change (NAME: Mary)

to (SALARY: $(80, \text{NOW}\rightarrow 40K)$)
in emp
Example 9.6. Suppose she gets a further raise at time $t = 90$ to 50K. We also now realize that her DEPT during $[74,75]$ was not Toys; in fact we don't know what it was. These changes are specified by the change operation below. The change operation modifies the tuple shown in Figure 9.2. to the tuple shown in Figure 9.3.

\[
\text{change (NAME: Mary)} \\
\text{to (SALARY: $([90,\text{NOW}]\rightarrow 50K)$)} \\
\text{DEPT: $([74,75]\rightarrow \xi)$)} \\
in \text{emp} \]

Example 9.7. Suppose we learn that Mary left the organization at $t = 92$. The change operation specified below allows us to incorporate this change. The change operation modifies the tuple shown in Figure 9.3. to the tuple shown in Figure 9.4.

\[
\text{change (NAME: Mary)} \\
\text{to (NAME: $([92,\text{NOW}]\rightarrow \xi)$)} \\
\text{SALARY: $([92,\text{NOW}]\rightarrow \xi)$} \\
\text{DEPT: $([92,\text{NOW}]\rightarrow \xi)$} \]
\( \ell = \ell - [92, \text{NOW}], u = u - [92, \text{NOW}] \)

in emp

**Example 9.8.** The following change operation deletes the tuple for Mary from the `emp` relation. We may delete Mary’s tuple because we realize that Mary was never any employee in the organization or because the information has become very old and we do not want to store it any more.

\[
\text{change (NAME: Mary)}
\]
\[
\text{to (NAME: } [0, \text{NOW}]^{-1} \text{)}
\]
\[
\text{SALARY: } [0, \text{NOW}]^{-1}
\]
\[
\text{DEPT: } [0, \text{NOW}]^{-1}
\]
\[
\ell = \emptyset, u = \emptyset
\]

in emp

Although we do not introduce any shortcuts, a realistic system may allow a query such as the one in Example 9.8 to be specified as `change (NAME: Mary) to (u = \emptyset)` in `emp`.

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>[70,91]Mary</td>
<td>[70,79] 30K</td>
<td>[70,73]Toys</td>
</tr>
<tr>
<td>[70,91]</td>
<td>[70,91] 40K</td>
<td>[70,91]</td>
</tr>
<tr>
<td>[70,91]</td>
<td>[70,91] 40K</td>
<td>[70,91]</td>
</tr>
</tbody>
</table>

Figure 9.4 Mary's tuple after still further change

9.3. The Changekey Operation

The `changekey` operation allows the value of key attributes of one or more objects to be changed. Suppose the key of a relation `r` is `A_1...A_n`. We define an *update tag* to be a structure of the form \((A_1:a_1,...,A_n:a_n)\). The changekey operation is of the form

\[
\text{changekey \( (UT_1 \text{ to } UT_2 \ldots; UT_{m-1} \text{ to } UT_m; UT_m \text{ to } UT_1) \)}
\]

In the above specification, `UT_1,...,UT_m` are update tags.
The result of the operation is that the key of the tuple having key $UT_1$ is changed to $UT_2$. If more tags are specified then a cyclic modification of keys is performed. It is necessary to allow cyclic modification of keys in one atomic step because sometimes the key of two (or more) objects may be interchanged. Then the modification of one key at a time will not be successful because of the constraint that there be only one tuple with a given key.

**Example 9.9.** We realize that the person we thought was John is actually Joan. We also know that there is no tuple with key Joan in the emp relation. This change can be made by the following changekey operation.

\[
\text{changekey} \ (\text{NAME}:\text{John} \ \text{to} \ \text{NAME}:\text{Joan}).
\]

**Example 9.10.** A case where the identities of John and Tom were swapped can be corrected by the following changekey operation. Note that this cannot be done by two changekey operations of the type in Example 9.9.

\[
\text{changekey} \ (\text{NAME}:\text{John} \ \text{to} \ \text{NAME}:\text{Tom}; \ \text{NAME}:\text{Tom} \ \text{to} \ \text{NAME}:\text{John})
\]
10. CONCLUSIONS

In this thesis we have used a model for temporal databases [GY88] with complete information as a basis for our research. We have presented a model for temporal databases with incomplete information. Our model allows the storage of incomplete temporal information and provides a powerful algebra to query the incomplete information.

Our model clearly generalizes the model for temporal databases with complete information. Our notions of partial temporal elements, partial temporal assignments, partial temporal relation and partial temporal databases generalize the corresponding concepts in the model for complete information. Furthermore, if the database had complete information, the result produced by the model for incomplete information for any algebraic expression would be the same as that obtained in the model for complete information, at least up to an isomorphism.

Using the notion of completions of relations and databases, we showed that the algebraic expressions produce reliable results even when we have incomplete information in the database. We then introduced the notion of information content, and showed that except for certain cases of selection, if the definition of our operators were strengthened to give more information, we could get results that are not reliable.

To build relational systems a user must be provided with a way to change the state of the database to reflect changes in the real world or to correct errors in the database. We have extended the operations create, change, and changekey from [Bh89] to operate on databases with incomplete information. The create operation allows us to add a tuple to a relation. The change operation allows us to modify values of non-key attributes, to modify domains of partial temporal assignments, to modify the $\ell$ value and $u$ value of a tuple, and also to delete a tuple. The changekey operation allows us to change the value of key attributes in a tuple.

We end the thesis with a few remarks about our work and how our work can be further extended.
Seamlessness and inherent querying capability of the algebra. We term the queries in a complete information model as standard. The standard queries can be submitted to the incomplete information model without any changes in the syntax. This means our algebra is a seamless extension of the algebra for complete information. In addition our algebra has the inherent capability to express nonstandard queries i.e. queries involving uncertainty. To achieve the said inherent capability, we only have to add primitives for constructs mentioned in Section 5.2. For example, if we define \( e(A) = A.t - \lceil A.t \rceil \), then this primitive captures the time when the object should be definitely present in the relation but its A-values are unknown. Then the query **give salaries of employees when their department was unknown while they surely worked for the organization** is expressed as \( \Pi_{\text{Name}, \text{Salary}} (\sigma(\text{emp}; \text{true}; e(\text{dept}))) \).

Our work as generalization of Biskup's. Except for some differences, our formalism is a generalization of Biskup's formalism when key values are required to be known. In Biskup's formalism, information is maintained only for the current instant \( \text{NOW} \) and the concept of maybe tuples is introduced by adding a STATUS column which has a 'd' for definite tuples and an 'm' for maybe tuples. In our model this can be achieved by restricting the upper limits of partial temporal assignments to \([\text{NOW}, \text{NOW}]\). Then definite tuples have lower limit \( [\text{NOW}, \text{NOW}] \) and maybe tuples have lower limit \( \emptyset \).

Further work. Imielinski and Lipski [IL84] introduce condition tables in which they use marked nulls and an additional column to store conditions which must be satisfied by each tuple. This helps them obtain a theorem which states that all valid conclusions expressible by relational expressions are in fact derivable in their system. It would be useful to investigate these ideas in the context of incomplete temporal information.
BIBLIOGRAPHY


[Li79] Lien, Y. E. Multivalued dependencies with null values in relational databases. Proc. of the Fifth International Conference on Very Large Data Bases, Rio De Janeiro, Brazil, 1979.


[Li84] Lipski, Witold. On relational algebra with marked nulls. Proc. of the ACM


ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Shashi Gadia, for his constant help and guidance during the course of this work. Dr. Gadia extended the major professor-student relationship to one characterized by trust, friendship and availability. I would also like to thank him for the unrestricted use of his Personal Computer throughout the course of this work.

I would also like to thank the other members of my committee – Dr. R. S. Dahiya, Dr. L. Miller, Dr. D.L. Pigozzi, and Dr. A.E. Oldehoeft – for their time and effort. Dr. Miller's comments on a previous version of this dissertation helped to improve its presentation.

I take this opportunity to thank my parents for their constant and selfless efforts. I can never repay the debt I owe them. Also, a big thank you to my wife, Rita, for her love and affection.