Unification of Temporal Data Models

Christian S. Jensen[1],
Michael D. Soo[2], *and Richard T. Snodgrass*[2]

TR 92-15

July 2, 1992

Abstract

To add time support to the relational model, both first normal form (1NF) and non-1NF approaches have been proposed. Each has associated difficulties. Remaining within 1NF when time support is added may introduce data redundancy. The non-1NF models may not be capable of directly using existing relational storage structures or query evaluation technologies. This paper describes a new, conceptual temporal data model that better captures the time-dependent semantics of the data while permitting multiple data models at the representation level. This conceptual model effectively moves the distinction between the various existing data models from a semantic basis to a physical, performance-relevant basis.

We define a conceptual notion of a bitemporal relation where tuples are stamped with sets of two-dimensional chronons in transaction-time/valid-time space. Next, we describe three representation schemes: a tuple-timestamped 1NF representation, a backlog relation composed of 1NF timestamped change requests, and a non-1NF attribute value-timestamped representation. We further investigate several variants of these representations. We use snapshot equivalence to relate the three representational data models with the conceptual bitemporal data model.

We then consider querying within the two-level framework. To do so, we define first an algebra at the conceptual level. We proceed to map this algebra to the representation level in such a way that new operators compute equivalent results for different representations of the same conceptual bitemporal relation. This demonstrates that all of these representations are faithful to the semantics of the conceptual data model, with many choices available that may be exploited to gain improved performance.

[1]Department of Mathematics and Computer Science
Aalborg University
Fredrik Bajers Vej 7E
DK-9220 Aalborg Ø, DENMARK
csj@iesd.auc.dk

[2]Department of Computer Science
University of Arizona
Tucson, AZ 85721
{soo,rts}@cs.arizona.edu

Unification of Temporal Data Models

Contents

1 Introduction

Adding time to the relational model has been a daunting task [BADW82, McK86, SS88, Soo91]. More than a dozen extended data models have been proposed over the last decade [Sno90]. Most of these models support *valid time*, that is, the time a fact was valid in the modeled reality. A few, notably [BZ82, BG89, Sno87, Sno93], have also supported *transaction time*, the time a fact was recorded in the database; such models are termed *bitemporal*, because they support both kinds of time.

While these data models differ on many dimensions, perhaps the basic distinction that has been oft stated is between first normal form (1NF) and non-1NF. A related distinction is between tuple timestamping and attribute value timestamping. Each has associated difficulties. Remaining within 1NF (an example being the timestamping of tuples with valid and transaction start and end times [Sno87]) may introduce redundancy because attribute values that change at different times are repeated in multiple tuples. The non-1NF models, one being timestamping attribute values with sets of intervals [Gad88], may not be capable of directly using existing relational storage structures or query evaluation techniques that depend on atomic attribute values.

It is our contention that focusing on data *presentation* (how temporal data is displayed to the user), on data *storage*, with its requisite demands of regular structure, and on efficient *query evaluation* has complicated the primary task of capturing the time-varying semantics. The result has been a plethora of incompatible data models and query languages, and a corresponding surfeit of database design and implementation strategies that may be employed across these models.

We advocate instead a very simple *conceptual* data model that captures the essential semantics of time-varying relations, but has no illusions of being suitable for presentation, storage, or query evaluation. We instead rely on existing data model(s) for these tasks, by demonstrating equivalence mappings between the conceptual model and several *representational* models. This equivalence is based on *snapshot equivalence*, which says that two relation instances are equivalent if all their snapshots, taken at all times (valid and transaction), are identical. Snapshot equivalence provides a natural means of comparing rather disparate representations. Finally, while not addressed here, we feel that the conceptual data model is the appropriate location for database design and query optimization.

In essence, we advocate moving the distinction between the various existing temporal data models from a semantic basis to a physical, performance-relevant basis, utilizing our proposed conceptual data model to capture the time-varying semantics.

The paper has the following outline. In the next section we define the conceptual model. We then examine three representational data models that have been previously proposed: tuple timestamping (e.g., [BZ82, NA89, Sad87, Sar90, Sno87, Sno93]), backlogs (e.g., [Kim78, JMRS92]), and attribute value timestamping (e.g., [CC87, Tan86, Gad88, LJ88, MS91]). We provide mappings between the conceptual model and these representational models. We also discuss *covering functions* that trade off space efficiency with operator simplicity and execution time efficiency.

Having presented both the conceptual data model and the representational data models, Section 4 presents an overview of the interaction among the data models. Snapshot equivalence is the subject of Section 5. Ironically, while definitions of snapshot equivalence are particular to individual data models, the definitions rely on model-specific operations, the notion of snapshot equivalence allows us to relate relation instances, as well as operators, of different representations, and also allows us to relate representations to the semantics ascribed to the conceptual model. Section 6 is devoted to generalizing algebraic operators of the relational model to apply to objects in the conceptual bitemporal model as well as the timestamped tuple representational model. As with data instances, we demonstrate correspondence of these operators.

After summarizing, we outline the next steps to be taken in utilizing the conceptual model

to integrate existing temporal data models. Proofs of all theorems may be found in the appendix.

2 Conceptual Bitemporal Relations

The primary reason behind the success of the relational model is its simplicity. A bitemporal relation is necessary more complex. Not only must it associate values with facts, as does the relational model, it must also specify *when* the facts were valid in reality, as well as *when* the facts were current in the database. Since our emphasis is on semantic clarity, we will extend the conventional relational model as small an extent as necessary to capture this additional information.

2.1 Definition

Tuples in a conceptual bitemporal relation instance are associated with time values from two orthogonal time domains, namely valid time and transaction time. Valid time is used for capturing the time-varying nature of the part of reality being modeled, and transaction time models the update activity of the relation. For both domains, we assume that the database system has limited precision, and we term the smallest time unit a *chronon*. As we can number the chronons, the domains are isomorphic to the domain of natural numbers.

In general, the schema of a conceptual bitemporal relation, $\mathcal{R}$, consists of an arbitrary number of explicit attributes, A_1, A_2, ..., A_n, encoding some fact (possibly composite) and an implicit timestamp attribute, T. Thus, a tuple, $x = (a_1, a_2, \ldots, a_n, t)$, in a conceptual bitemporal relation instance, $r(\mathcal{R})$, consists of a number of attribute values associated with a timestamp value.

An arbitrary subset of the domain of valid times is associated with each tuple, meaning that the fact recorded by the tuple is *true in the modeled reality* during each valid time chronon in the subset. Each individual valid time chronon of a single tuple has associated an arbitrary subset of the domain of transaction times, meaning that the fact, valid during the particular chronon, is *current in the relation* during each of the transaction time chronons in the subset.

Associated with a tuple is a set of so-called *bitemporal chronons* (tiny rectangles) in the two-dimensional space spanned by valid time and transaction time. Such a set is termed a *bitemporal element*[1], denoted t_b. Because no two tuples with mutually identical explicit attribute values (termed *value-equivalent*) are allowed in a bitemporal relation instance, the full time history of a fact is contained in a single tuple.

EXAMPLE: Consider a relation recording employee/department information, such as "Jake works for the shipping department." We assume that the granularity of chronons is one day for both valid time and transaction time, and the period of interest is the month of June 1992.

Figure 1 shows how the bitemporal element in an employee's department tuple changes. The x-axis denotes transaction time, and the y-axis denotes valid time. Employee Jake was hired by the company as temporary help in the shipping department for the interval from June 10th to June 15th, and this fact is recorded in the database proactively on June 5th. This is shown in Figure 1(a). The arrows pointing to the right signify that the tuple has not been logically deleted; it continues through to the transaction time NOW. On June 10th, the personnel department discovers an error. Jake had really been hired for the valid time interval from June 5th to June 20th. The database is corrected on June 10th, and the updated bitemporal element is shown in Figure 1(b). On June 15th, the personnel department is informed that the correction was itself incorrect; Jake really was hired for the original time interval, June 10th to June 15th, and the database is corrected the same day. This is shown in Figure 1(c). Lastly, Figure 1(d) shows the

[1] This term is a generalization of *temporal element*, used to denotes a set of single dimensional chronons [Gad88]. An alternative, equally desirable term is *bitemporal lifespan* [CC87].

result of three updates to the relation, all of which take place on June 20th. While the the period of validity was correct, it was discovered that Jake was not in the shipping department, but in the loading department. Consequently, the fact (Jake, Ship) is removed from the current state and the fact (Jake, Load) is inserted. A new employee, Kate, is hired for the shipping department for the interval from June 25th to June 30th.

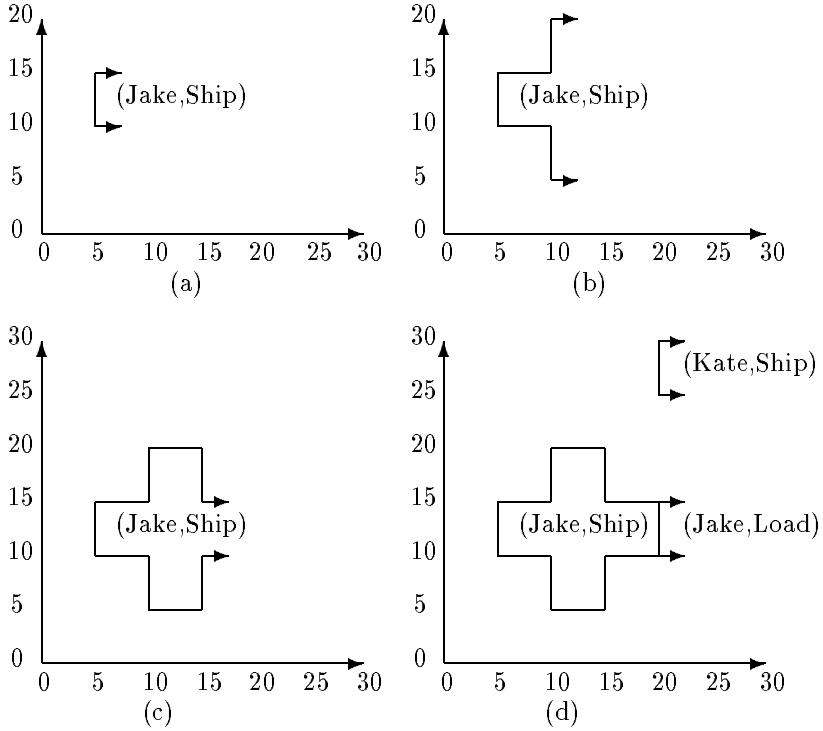

Figure 1: Bitemporal Elements

We note that the number of bitemporal chronons in a given bitemporal element is the area enclosed by the bitemporal element. The bitemporal element for (Jake, Ship) contains 140 bitemporal chronons.

The example illustrates how transaction time and valid time are handled. As time passes, i.e., as the computer's internal clock advances, the bitemporal elements associated with current facts are updated. For example, when (Jake, Ship) was first inserted, the six valid time chronons from 10 to 15 had associated the transaction time chronon NOW. At time 5, the six new bitemporal chronons, $(5, 10), \ldots, (5, 15)$, were appended. This continued until time 9, after which the valid time was updated. Thus, starting at time 10, 16 bitemporal chronons are added at every clock tick.

The actual bitemporal relation corresponding to the graphical representation in Figure 1(d) is shown below. This relation contains three facts. The timestamp attribute T shows each transaction time chronon associated with each valid time chronon as a set of ordered pairs.

Emp	Dept	T
Jake	Ship	$\{(5,10),\ldots,(5,15),\ldots,(9,10),\ldots,(9,15),$ $(10,5),\ldots,(10,20),\ldots,(14,5),\ldots,(14,20),$ $(15,10),\ldots,(15,15)\ldots,(19,10),\ldots,(19,15)\}$
Jake	Load	$\{(NOW,10),\ldots,(NOW,15)\}$
Kate	Ship	$\{(NOW,25),\ldots,(NOW,30)\}$

□

2.2 Update

We consider the three forms of update, insertion, deletion, and modification, in turn.

An insertion is issued when we want to record in bitemporal relation instance r that a currently unrecorded fact $(a_1, \ldots, a_n)$ is true for some period(s) of time. These periods of time are represented by a valid-time element, i.e., a set of valid-time chronons, t_v. When the fact is stored, its valid-time element stamp is transformed into a bitemporal element stamp to capture that, from now on, the fact is current in the relation. We indicate this with a special value in the domain of transaction chronon identifiers, NOW.

The arguments to the `insert` routine are the relation into which a fact is to be inserted, the explicit values of the fact, and the set of valid-time chronons, t_v, during which the fact was true in reality. Insert returns the new, updated version of the relation. There are three cases to consider. First, if $(a_1, \ldots, a_n)$ was never recorded in the relation, a completely new tuple is appended. Second, if $(a_1, \ldots, a_n)$ was part of some previously current state, the tuple recording this is updated with the new valid time information. Third, if $(a_1, \ldots, a_n)$ is already current in the relation, a modification is required, and the insertion is rejected. (In the following, we denote valid-time chronons with c_v and transaction-time chronons with c_t.)

$$\texttt{insert}(r, (a_1, \ldots, a_n), t_v) = \begin{cases} r \cup \{(a_1, \ldots, a_n, \{NOW\} \times t_v)\} & \text{if } \neg\exists\, t\ ((a_1, \ldots, a_n, t) \in r) \\ r - \{(a_1, \ldots, a_n, t_b)\} & \\ \quad \cup\{(a_1, \ldots, a_n, t_b \cup \{NOW\} \times t_v)\} & \text{if } \exists\, t_b\ ((a_1, \ldots, a_n, t_b) \in r\ \wedge\ \neg\exists\, (NOW, c_v) \in t_b) \\ r & \text{otherwise} \end{cases} \quad (1)$$

The `insert` routine adds bitemporal chronons with a transaction time of NOW.

As time passes, new chronons must be added. We assume that a special routine `ts_update` is applied to all bitemporal relations at each clock tick. We also assume that the transaction-time granularity is sufficiently small that only one transaction can execute within a transaction-time chronon. This function simply updates the timestamps to include the new transaction time value. The timestamp of each tuple is examined in turn. When a bitemporal chronon of the type (NOW, c_v) is encountered in the timestamp, a new bitemporal chronon (c_t, c_v), where time c_t is the new transaction-time value, is made part of the timestamp.

```
ts_update(r, c_t) :
    for each x ∈ r
        for each (NOW, c_v) ∈ x[T]
            x[T] ← x[T] ∪ {(c_t, c_v)};
```

Deletion concerns the (logical) removal of a complete tuple from the current valid-time state of the bitemporal relation. We distinguish between the case when there is a tuple to delete and the case when no tuple matching the one to be deleted exists.

$$\texttt{delete}(r, (a_1, \ldots, a_n)) = \begin{cases} r - \{(a_1, \ldots, a_n, t_b)\} \cup \{(a_1, \ldots, a_n, t_b - now\_ts(t_b))\} & \text{if } \exists\, t\ ((a_1, \ldots, a_n, t_b) \in r) \\ r & \text{otherwise} \end{cases} \quad (2)$$

where now_ts is defined as follows.

$$now\_ts(t_b) = \{(NOW, c_v) \mid (NOW, c_v) \in t_b\}$$

Finally, a modification of an existing tuple may be defined by a deletion followed by an insertion as follows.

$$\texttt{modify}(r, (a_1, \ldots, a_n), t_v) = \texttt{delete}(r, (a_1, \ldots, a_n)) \cup \texttt{insert}(r, (a_1, \ldots, a_n), t_v) \quad (3)$$

EXAMPLE: The sequence of bitemporal elements shown in Figure 1 is created by the following sequence of commands, invoked at the indicated transaction time.

Command	*Transaction Time*
`insert(dept,("Jake","Ship"),[6/10,6/15])`	6/5
`modify(dept,("Jake","Ship"),[6/5,6/20])`	6/10
`modify(dept,("Jake","Ship"),[6/10,6/15])`	6/15
`delete(dept,("Jake","Ship"))`	6/20
`insert(dept,("Jake","Load"),[6/10,6/15])`	6/20
`insert(dept,("Kate","Ship"),[6/25,6/30])`	6/20

□

Valid time relations and transaction time relations are special cases of bitemporal relations that support only valid time and transaction time, respectively. Thus an valid-time tuple has associated a set of valid time chronons (termed a *valid-time element* and denoted t_v), and a transaction-time tuple has associated a set of transaction time chronons (termed a *transaction-time element* and denoted t_t). For clarity, we use the term snapshot relation for a conventional relation. Snapshot relations support neither valid time nor transaction time.

3 Representation Schemes

A conceptual bitemporal relation is structurally simple—it is a set of facts, each timestamped with a bitemporal element which is a set of bitemporal chronons. In this section we examine three representations of bitemporal relations that have been previously proposed. For each, we briefly specify the objects defined in the representation, provide the mapping to and from conceptual bitemporal relations to demonstrate that the same information is being stored, and show how updates of conceptual bitemporal relations may be mapped into updates on relations in the representation. We end by briefly considering two additional representations.

3.1 A Tuple Timestamped Representation Scheme

In the conceptual model, the timestamp associated with a tuple is an arbitrary set of bitemporal chronons. As such, a relation schema in the conceptual model is non-1NF, which represents difficulties if directly implemented. We describe here how to represent conceptual relations by 1NF snapshot relations, allowing the use of existing, well-understood implementation techniques.

Let a bitemporal relation schema $\mathcal{R}$ have the attributes $A_1, \ldots, A_n, \mathrm{T}$ where T is the timestamp attribute defined on the domain of bitemporal elements. This schema is represented by a snapshot relation schema R as follows.

$$R \;=\; (A_1, \ldots, A_n, \mathrm{T}_s, \mathrm{T}_e, \mathrm{V}_s, \mathrm{V}_e)$$

The additional attributes T_s, T_e, V_s, V_e are atomic-valued timestamp attributes containing a starting and ending transaction time chronon and a starting and ending valid time chronon, respectively. These four values represent the bitemporal chronons in a rectangular region, and the

idea is to divide the complete region, covered by the bitemporal element of a single tuple in a conceptual relation instance, into a number of rectangles and then represent the conceptual tuple by a set of tuples, one for each rectangle.

There is a multitude of possible ways of covering a bitemporal element. For any function, *cover*, that covers a bitemporal element $x[\mathrm{T}]$ of a bitemporal tuple x, we require that

1. Any bitemporal chronon in $x[\mathrm{T}]$ must be contained in at least one rectangle.
2. Each bitemporal chronon in a rectangle must be contained in $x[\mathrm{T}]$.

Apart from these requirements, the covering function is purposefully left unspecified—an implementation is free to choose a covering with properties it finds desirable. For example, a set of covering rectangles need not be disjoint. Overlapping rectangles may reduce the number of tuples needed in the representation, at the possible expense of additional processing during update.

EXAMPLE: The 1NF relation corresponding to the conceptual relation in Figure 1(d) is shown below.

Emp	*Dept*	T_s	T_e	V_s	V_e
Jake	Ship	6/5	6/9	6/10	6/15
Jake	Ship	6/10	6/14	6/5	6/20
Jake	Ship	6/15	6/19	6/10	6/15
Jake	Load	6/20	*NOW*	6/10	6/15
Kate	Ship	6/20	*NOW*	6/25	6/30

Here we use a non-overlapping covering function that partitions the bitemporal element by transaction time. □

Throughout the paper, we will use R and S to denote relation schemas. Relation instances are denoted r, s, and t, and $r(R)$ means that r is an instance of R. Attributes are denoted A_i, B_i, and C_i. For brevity, we let A denote the set of all attributes A_i. For tuples we use x, y, and z (possibly indexed), and the notation $x[A_i]$ is defined to be the A_i^{th} attribute value of tuple x. As a shorthand, we define $x[V]$ to be the closed interval from $x[\mathrm{V}_s]$ to $x[\mathrm{V}_e]$ (a set of one-dimensional valid-time chronons), and similarly for $x[\mathrm{T}]$, a set of transaction-time chronons.

The following functions convert between a conceptual bitemporal relation instance and a corresponding instance in the representation scheme. The second argument, *cover*, of the routine `conceptual_to_snap` is a covering function. It returns a set of rectangles, each denoted by a set of bitemporal chronons. Note that the functions are the inverse of each other, i.e., for any conceptual relation instance r',

$$\texttt{snap\_to\_conceptual}(\texttt{conceptual\_to\_snap}(r', \mathit{cover})) = r'.$$

```
conceptual_to_snap(r', cover):
    s ← ∅;
    for each x ∈ r'
        z[A] ← x[A];
        for each t ∈ cover(x[T])
            z[T_s] ← min_1(t); z[T_e] ← max_1(t);
            z[V_s] ← min_2(t); z[V_e] ← max_2(t);
            s ← s ∪ {z};
    return s;
```

```
snap_to_conceptual(r):
    s ← ∅;
    for each z ∈ r
        r ← r − {z};
        x[A] ← z[A];
        x[T] ← bi_chr(z[T], z[V]);
        for each y ∈ r
            if z[A] = y[A]
                r ← r − {y};
                x[T] ← x[T] ∪ bi_chr(y[T], y[V]);
            s ← s ∪ {x};
    return s;
```

In the conversion routines above, the functions *min_1* and *min_2* select a minimum first and second component, respectively, in a set of binary tuples. The function *max_1* returns the value NOW if encountered as a first component; otherwise, it returns a maximum first component. The function *max_2* selects a maximum second component. Finally, the function *bi_chr* computes the bitemporal chronons covered by the argument rectangular region.

The `conceptual_to_snap` routine generates possibly many representational tuples from each conceptual tuple, each corresponding to a rectangle in valid/transaction time space. The `snap_to_conceptual` routine merges the rectangles associated with a single fact into a single bitemporal element.

For the update routines, the most convenient covering function partitions on transaction time, and does not permit overlap. The current transaction time is c_t.

```
insert(r, (a_1, ..., a_n), t_v, cover_v):
    for each t ∈ cover_v(t_v)
        for each x ∈ r
            if x[T_e] = NOW and x[A] = (a_1, ..., a_n) and
                    x[V] ∩ t ≠ ∅
                r ← r − {s};
                x[T_e] ← c_t;
                z[A] ← x[A];
                z[T_s] ← c_t;  z[T_e] ← NOW;
                z[V_s] ← min(z[V] ∪ t);  z[V_e] ← max(z[V] ∪ t);
                r ← r ∪ {x, z};
    return r;
```

```
delete(r, (a_1, ..., a_n)):
    s ← ∅;
    for each x ∈ r
        if x[A] = (a_1, ..., a_n) and
                x[T_e] = NOW
            x[T_e] ← c_t;
        s ← s ∪ {x};
    return s;
```

The `insert` routine logically deletes any bitemporal rectangles with a transaction time extending to NOW that overlaps in valid time with t_v, and inserts a new rectangle with the inclusive valid time. Function $cover_v$ returns a set of valid time intervals (each a set of contiguous valid-time chronons). The `delete` routine simply replaces the transaction end time with the current time, c_t.

As for the conceptual data model, `modify` is simply a combination of `append` and `delete`.

3.2 A Backlog-Based Representation Scheme

The previous representation scheme presented a very natural and frequently used way of representing a bitemporal relation by a snapshot relation.

In the backlog-based representation scheme bitemporal relations are represented by backlogs, which are also 1NF relations [Kim78, JMRS92]. The most important difference between this and the previous schemes is that tuples in backlogs are never updated, i.e., backlogs are append-only. Therefore, this representation scheme is well-suited for log based storage of bitemporal relations, and it opens the possibility of using cheap write-once optical disk storage devices. This is highly desirable since the information content of bitemporal relations is ever-growing, resulting in very large relations.

A bitemporal relation schema $\mathcal{R} = (A_1, \ldots, A_n, \mathrm{T})$ is represented by a backlog relation schema R as follows.

$$R \quad = \quad (A_1, \ldots, A_n, \mathrm{V}_s, \mathrm{V}_e, \mathrm{T}, \mathrm{Op})$$

As in the previous representation scheme, the attributes V_s and V_n store starting and ending valid time chronons, respectively. Attribute T stores the transaction time when the tuple was inserted into the backlog. Tuples, termed update requests, are either insertion requests or deletion requests, as indicated by the values, I, and D, of attribute Op. The fact in an insertion request is current starting at its transaction timestamp and until a matching deletion request with same explicit and

valid time attribute values is recorded. Modifications are recorded by a pair of a deletion request and an insertion request, both with the same T value.

EXAMPLE: The backlog relation corresponding to the conceptual relation in Figure 1(d) is shown below.

Emp	*Dept*	V_s	V_e	T	Op
Jake	Ship	6/10	6/15	6/5	*I*
Jake	Ship	6/10	6/15	6/10	*D*
Jake	Ship	6/5	6/20	6/10	*I*
Jake	Ship	6/5	6/20	6/15	*D*
Jake	Ship	6/10	6/15	6/15	*I*
Jake	Ship	6/10	6/15	6/20	*D*
Jake	Load	6/10	6/15	6/20	*I*
Kate	Ship	6/25	6/30	6/20	*I*

□

Next, we consider the conversion between a bitemporal relation and its backlog representation. The first function, `conceptual_to_back`, takes a conceptual relation as its first argument. The second argument is an arbitrary covering function as described in Section 3.1. The result is a backlog relation. Each conceptual tuple, x, is treated in turn. For each rectangle of bitemporal chronons in the cover of the timestamp of x, an insertion request is appended to the result. Further, if the rectangle has an ending transaction time different from NOW then a deletion request is inserted.

```
conceptual_to_back(r', cover):
    r ← ∅;
    for each x ∈ r'
        for each t ∈ cover(x[T])
            z[A] ← x[A];
            z[V_s] ← min_2(t); z[V_e] ← max_2(t);
            z[Op] ← I; z[T] ← min_1(t);
            r ← r ∪ {z};
            if max_1(t) ≠ NOW
                z[Op] ← D; z[T] ← max_1(t);
                r ← r ∪ {z};
    return r;
```

```
back_to_conceptual(r):
    r' ← ∅;
    for each z_1 ∈ r
        if z_1[Op] = I
            a ← min_2(z_1[V_s]); b ← max_2(z_1[V_e]);
            c ← z_1[T]; d ← NOW;
            x_1[A] ← z_1[A];
            r ← r − {z_1};
            for each z_2 ∈ r
                if z_2[A] = z_1[A] and z_2[V] = z_1[V] and
                        z_2[Op] = D and z_1[T] < z_2[T] < d
                    d ← z_2[T];
                    z_3 ← z_2;
            if d ≠ NOW
                r ← r − {z_3};
            x_1[T] ← bi_chr((c, d), (a, b));
            for each x_2 ∈ r'
                if x_2[A] = x_1[A]
                    x_1[T] ← x_1[T] ∪ x_2[T];
                    r' ← r' − {x_2};
            r' ← r' ∪ {x_1};
    return r';
```

The second function, `back_to_conceptual`, is the inverse transformation. It is rather complex because not only is information about a single fact spread over a set of update requests, but, as we just saw, one element in a covering may also be recorded in two change requests. The change requests in the argument backlog relation are treated in turn. First, an insertion request is located, and its attribute values are recorded as appropriate. It is initially assumed that the ending transaction time is NOW. Then, in the second loop, the backlog is scanned for a matching

deletion request with a larger transaction time. If more than one exists, the earliest is chosen. Now, the correct rectangular region of bitemporal chronons has been computed, and this can be recorded in conceptual bitemporal relation. If other chronons have already been computed and recorded for the same fact, the two sets of chronons are simply merged.

As expected, insertion into backlogs, where tuples are never changed, is straightforward. For each set of consecutive valid-time chronons returned by the argument covering function, an insertion request with the appropriate attribute values is created. The current transaction time is assumed to be c_t.

Deletion follows the same pattern, the only complication being that a deletion request can only be inserted if a value-equivalent, previously entered and so far undeleted insertion request is found. First, the backlog is scanned to locate a matching insertion request. Second, it is ensured that the located insertion request has not previously been deleted. For every undeleted, matching insertion request that is found, a deletion request is inserted.

```
insert(r,(a_1,...,a_n),t_v,cover_v):
    for each t ∈ cover_v(t_v)
        r ← r ∪ {(a_1,...,a_n,min(t),max(t),c_t,I)};
    return r;
```

```
delete(r,(a_1,...,a_n)):
    r' ← r;
    for each x_1 ∈ r
        if x_1[A] = (a_1,...,a_n) and x_1[Op] = I
            found ← TRUE;
            for each x_2 ∈ r
                if x_2[A] = x_1[A] and x_2[V] = x_1[V] and
                   x_2[OP] = D and x_2[T] > x_1[T]
                    found ← FALSE;
            if found
                r' ← r' ∪ {(a_1,...,a_n,x_1[V_s],x_1[V_e],c_t,D)};
    return r';
```

3.3 An Attribute Value Timestamped Representation Scheme

Non-1NF representations group all information about an object within a single tuple. As such, attribute value timestamped representations have become popular for their flexibility in data modeling. We describe here how to represent conceptual relations by non-1NF attribute value timestamped relations.

Let a bitemporal relation schema $\mathcal{R}$ have the attributes $A_1, \ldots, A_n$, T, where T is the timestamp attribute defined on the domain of bitemporal elements. Then bitemporal relation schema $\mathcal{R}$ is represented by an attribute value timestamped relation schema R as follows.

$$R \quad = \quad (\{([\mathrm{T}_s, \mathrm{T}_e], [\mathrm{V}_s, \mathrm{V}_e]\ A_1)\}, \ldots, \{([\mathrm{T}_s, \mathrm{T}_e], [\mathrm{V}_s, \mathrm{V}_e]\ A_n)\})$$

A tuple is composed of n sets. Each set element a is a triple of a transaction-time interval pair $[\mathrm{T}_s, \mathrm{T}_e]$, a valid-time interval pair $[\mathrm{V}_s, \mathrm{V}_e]$, representing in concert a rectangle of bitemporal chronons, and an attribute value, denoted $a.val$. As shorthand we will use T to denote the transaction time interval $[\mathrm{T}_s,\mathrm{T}_e]$, and, similarly, V for $[\mathrm{V}_s,\mathrm{V}_e]$, and will refer to them as a.T and a.V.

EXAMPLE: In an attribute value timestamped representation, the grouping of information within a tuple can be based on the value of any attribute or set of attributes. For example, we could represent the conceptual relation in Figure 1(d) by grouping on the employee attribute. Then all information for an employee is contained within a single tuple, as shown below [Gad92].

Emp		Dept	
[20,NOW] × [25,30]	Kate	[20,NOW] × [25,30]	Ship
[5,9] × [10,15]	Jake	[5,10] × [10,15]	Ship
[10,14] × [5,20]	Jake	[10,15] × [5,20]	Ship
[15,19] × [10,15]	Jake	[15,20] × [10,15]	Ship
[20,NOW] × [10,15]	Jake	[20,NOW] × [10,15]	Load

A tuple in the above relation shows all departments for which a single employee has worked. A different way to view the same information is to perform the grouping by department. A single tuple then contains all information for a department, i.e., the full record of employees who have worked for the department.

Emp		Dept	
[20,NOW] × [10,15]	Jake	[20,NOW] × [10,15]	Load
[5,9] × [10,15]	Jake	[5,10] × [10,15]	Ship
[10,14] × [5,20]	Jake	[10,15] × [5,20]	Ship
[15,19] × [10,15]	Jake	[15,20] × [10,15]	Ship
[20,NOW] × [25,30]	Kate	[20,NOW] × [25,30]	Ship

Grouping by both attributes would yield three tuples, (Jake, Load), (Jake, Ship), and (Kate, Ship). □

Next we consider the conversion between a conceptual relation and an attribute value timestamped representation. The first function, `conceptual_to_att`, takes three arguments, r', a conceptual relation, *cover*, a covering function, and *group*, a grouping function. r' and *cover* are as described for the other representation schemes. *group* partitions r' into disjoint subsets where all tuples in a subset agree on the values of a particular attribute or set of attributes, as illustrated in the above example. Each group of conceptual tuples produces one representation tuple.

```
conceptual_to_att(r', cover, group):
    s ← ∅;
    G ← group(r');
    for each g ∈ G
       for each x ∈ g
          z ← (∅, ..., ∅);
          for each t ∈ cover(x[T])
             for i ← 1 to n
                z[A_i] ← z[A_i] ∪
                   {([min_1(t), max_1(t)],
                      [min_2(t), max_2(t)] x[A_i])};
          s ← s ∪ {z};
    return s;
```

```
att_to_conceptual(r):
    s ← ∅;
    for each z ∈ r
       for i ← 1 to n
          g[i] ← ∅;
          while z[A_i] ≠ ∅
             y ← next(z[A_i]);
             a.val ← y.val;
             a.t ← bi_chr(y.T, y.V);
             z[A_i] ← z[A_i] − y;
             for each y ∈ z[A_i]
                if a.val = y.val
                   a.t ← a.t ∪ bi_chr(y.T, y.V);
                   z[A_i] ← z[A_i] − {y};
             g[i] ← g[i] ∪ {a};
       f ← facts(g);
       for each (a_1, a_2, ..., a_n) ∈ f
          t ← a_1.t;
          for i ← 2 to n
             t ← t ∩ a_i.t;
          if t ≠ ∅
             for i ← 1 to n
                x[A_i] ← a_i.val;
             x[T] ← t;
             s ← s ∪ {x};
    return s;
```

The second function, `att_to_conceptual`, performs the inverse transformation. Given an attribute value timestamped representation it produces the equivalent conceptual relation. If we regard the transaction/valid times associated with an attribute value as rectangles, then the function simply constructs these rectangles for each attribute value in a tuple and then uses intersection semantics to determine the equivalent tuple timestamp. In this transformation, the grouping is ignored.

In the above, the function *next* returns an attribute value and its rectangle from an attribute in a tuple. Subsequent calls to *next* cycle through the attribute values. The *facts* function computes, for an array of attribute value/rectangle sets, all combinations of facts that can be constructed from those attribute values.

$$facts(g) = \{((a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)) \mid \forall i \; 1 \leq i \leq n((a_i, t_i) \in g[i])\}$$

As before the function *bi_chr* computes the bitemporal chronons represented by a given rectangle.

While insertion and deletion functions can be defined, their implementations provide little insight. A simple, but very inefficient, way to implement these functions is to convert the representation to a conceptual relation, perform the update, and then convert the modified conceptual relation back to the attribute value timestamped representation.

3.4 Covering Functions

In Sections 3.1 to 3.3, we used covering functions when representing bitemporal elements of conceptual tuples by sets of rectangles. Any covering function that covered every bitemporal chronon in an argument bitemporal element and did not cover bitemporal chronons not in the bitemporal element was permitted. In this sense, the results presented in this paper are independent of particular covering functions. Here, we briefly present some types of covering functions to illustrate the range of possibilities.

Figure 2 illustrates three ways of covering the bitemporal element associated with the fact (Jake, Ship) in Figure 1(d). We may distinguish between those covering functions that partition the argument set into disjoint rectangles and those that allow overlap between the result rectangles. Figure 2(a) and Figure 2(b) are examples of partitioned coverings while the covering in Figure 2(c) has overlapping rectangles.

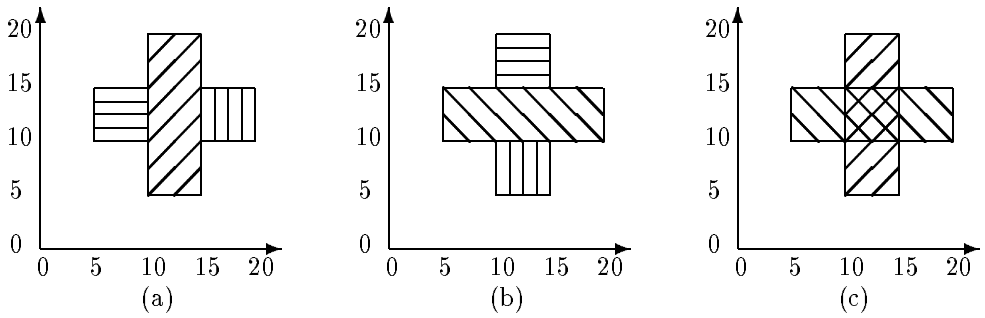

Figure 2: Example Coverings of a Bitemporal Element

Figure 2(a) illustrates a type of covering where regions are partitioned by transaction time. Maximal transaction time intervals are located so that each transaction time in an interval has the same interval of valid times associated. In the figure, the transaction time interval (5,9) is maximal, and the associated valid time interval is (10,15). Thus, the rectangle with corners (5,10) and (9,15) is part of the result. Similarly, the two rectangles with corners ((10,5), (14,20)), and ((15,10), (19,15)) are in the result. Due to the semantics of transaction time [JMRS92], this is

perhaps the most natural choice of covering [Sno87]. Indeed, all the examples of representations of the employee bitemporal relation use covering functions that partition by transaction time.

Figure 2(b) illustrates the symmetric partitioning by valid time. Here, three rectangles are created with corners at ((5,10), (19,15)), ((10,5), (14,10)), and ((10,15), (14,20)).

Figure 2(c) exemplifies a type of covering that allows overlaps. The two rectangles in this covering have corners at ((5,10), (19,15)) and ((10,5), (14,20)). The overlap of these rectangles means that two tuples will express the fact that Jake was in the shipping department from June 10th to June 15th, recorded as current information from June 10th to June 14th.

The example demonstrates that a covering function that allows overlap may result in a smaller number of covering rectangles, and therefore may yield a more compressed representation, than a covering function that partitions. However, this repetition of information makes some updates more time consuming.

3.5 Other Representations

The three representations just discussed are not the only ones that have been proposed to support both valid and transaction time. BenZvi introduced the first bitemporal representation, similar to the tuple timestamping scheme in Section 3.1, but with five timestamps: (1) valid begin, (2) valid end, (3) the transaction time that valid begin was recorded, (4) the transaction time that valid end was recorded, and (5) the transaction time that the tuple was logically deleted [BZ82].

Another representation often mentioned is a sequence of *historical states* indexed by transaction time [SA85]. This representation is derived by first partitioning the transaction time dimension according to the beginning and ending points of the transaction time intervals of all the tuples in the bitemporal relation. Second, for each partition, all tuples current in the partition are collected along with their valid time intervals. These sets are historical relations indexed by transaction time. The transaction time interval of a partition is the existence interval of the historical relation, i.e., the time when the entire historical relation was the current state of the bitemporal relation. Alternatively, we can envision a bitemporal relation as a sequence of transaction time states indexed by valid time.

It is possible to also devise mapping functions between these two additional representation schemes (BenZvi's tuple-timestamped and a sequence of historical states) and conceptual bitemporal relations. Also, the results of the rest of the paper could be applied to these other representations.

4 Data Model Interaction

The previously proposed representations arose from several considerations. They were all extensions of the conventional relational model that attempted to capture the time-varying semantics of both the enterprise being modeled and the state of the database. They attempted to retain the simplicity of the relational model; the tuple timestamping model was perhaps most successful in this regard. They attempted to present all the information concerning an object in one tuple; the attribute value timestamped model was perhaps best at that. And they attempted to ensure ease of implementation and query evaluation efficiency; the backlog representation may be advantageous here.

It is clear from the number of proposed representations that meeting all of these goals simultaneously is a difficult, if not impossible task. We therefore advocate a separation of concerns. The time-varying semantics is obscured in the representation schemes by other considerations of presentation and implementation. We feel that the conceptual bitemporal data model proposed in this paper is the most appropriate basis for expressing this semantics. This data model is notable

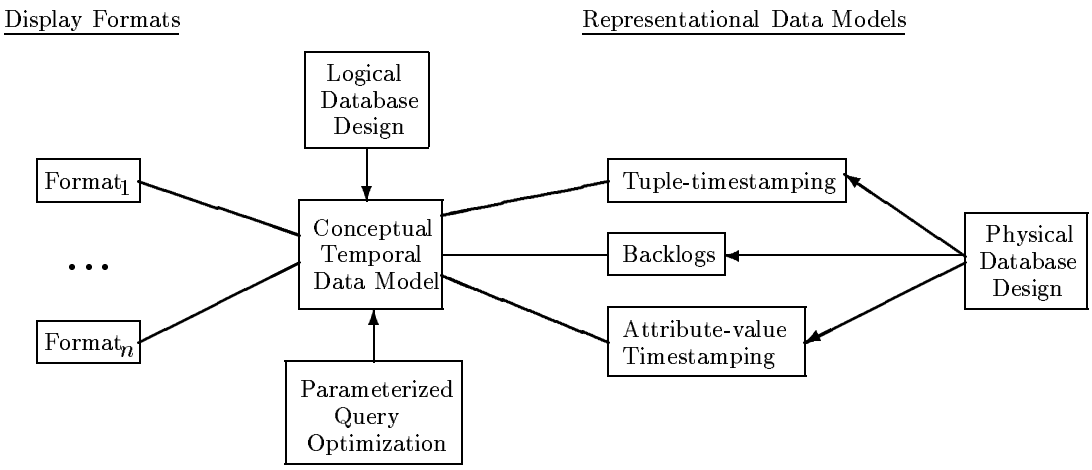

Figure 3: Interaction of Conceptual and Representational Data Models

in its use of bitemporal chronons to stamp facts. Clearly, in most situations, this is not the most appropriate way to present the stored data to users, nor is it the best way to physically store the data. However, since there are mappings to several representations that, in many situations, may be more amenable to presentation and storage, those representations can be employed for those purposes, while retaining the semantics of the conceptual data model. Figure 3 shows the placement of the conceptual bitemporal data model with respect to the tasks of logical and physical database design, storage representation, query optimization, and display. As the figure shows, logical database design produces the conceptual relation schemas, which are then refined into relation schemas in some representational data model(s). Query optimization may be performed on the logical algebra, parameterized by the cost models of the representation(s) chosen for the stored data. Finally, display presentation should be decoupled from the storage representation. Section 3 gave four different representations of the example conceptual relation introduced in Section 2.1. Each of these is an appropriate presentation for some situations, independent of how the relation is stored. The backlog presentation is quite useful during an audit, and the first attribute value timestamped presentation is suitable when the history of an employee is desired.

Note that this arrangement hinges on the semantic equivalence of the various data models. It must be possible to map between the conceptual model and the various representational models, as discussed next.

5 Semantic Equivalence

The previous section claimed that many equivalent representations of the same conceptual relation may co-exist. In this and the next section, we explore in more detail this relationship between the conceptual data model and the representational data models. We focus here on the objects in the models; the next section will examine operations on these objects.

5.1 The Rollback and Timeslice Operators

We use snapshot equivalence to formalize the notion of equivalent representations. Snapshot equivalence makes use of the notions of rollback and timeslice, which we define for each representation.

The *rollback* operator, ρ^{B}, takes two arguments, a bitemporal relation and a time value, the latter appearing as a subscript. The result is a valid-time relation. In order to explain the semantics of $trho$, we describe its operation on a conceptual bitemporal relation. Each tuple is examined in

turn. If any of its associated bitemporal chronons have a transaction time matching the argument time, the explicit attribute values and each of the valid time chronons with a matching transaction time become a tuple in the result. The rollback operator may also be applied to a transaction time relation, in which case the result is a snapshot relation.

The *timeslice* operator, τ^{B}, is very similar. It also takes two arguments, a bitemporal relation and a time value. The difference is that this operator does the selection on the valid time and produces a transaction time relation. The timeslice operator may also be applied to a valid time relation, in which case the result is a snapshot relation.

In the following, let t denote an arbitrary time value and let t' denote a time not exceeding NOW.

DEFINITION: (Tuple Timestamped Data Model). Define a relation schema $R = (A_1, \ldots, A_n, \mathrm{T}_s, \mathrm{T}_e, \mathrm{V}_s, \mathrm{V}_e)$, and let r be an instance of this schema.

$$\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n+2)} \mid \exists x \in r\,(z[A] = x[A] \wedge z[\mathrm{V}] = x[\mathrm{V}] \wedge t' \in x[\mathrm{T}])\} \quad (4)$$

$$\tau^{\mathrm{B}}_{t}(r) = \{z^{(n+2)} \mid \exists x \in r\,(z[A] = x[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \wedge t \in x[\mathrm{V}])\} \quad (5)$$

□

DEFINITION: (Backlog Data Model). Define a relation schema $R = (A_1, \ldots, A_n, \mathrm{V}_s, \mathrm{V}_e, \mathrm{T}, \mathrm{Op})$, and let r be an instance of this schema. Let t denote an arbitrary time value and let t' denote a time not exceeding NOW.

$$\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n+2)} \mid \exists x \in r\,(z[A] = x[A] \wedge z[\mathrm{V}] = x[\mathrm{V}] \wedge x[\mathrm{T}] \le t' \wedge x[\mathrm{Op}] = I \wedge (\neg\exists y \in r\,(y[A] = x[A] \wedge y[\mathrm{V}] = x[\mathrm{V}] \wedge y[\mathrm{Op}] = D \wedge x[\mathrm{T}] \le y[\mathrm{T}] \le t'))))\} \quad (6)$$

$$\tau^{\mathrm{B}}_{t}(r) = \{z^{(n+2)} \mid \exists x \in r\,(z[A] = x[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \wedge z[\mathrm{Op}] = x[\mathrm{Op}] \wedge t \in x[\mathrm{V}])\} \quad (7)$$

In the definition of rollback, an insertion request contributes to the result if it was entered before the argument transaction time t' and if it was not subsequently countered by a deletion request before t'. The non-symmetry of these two definitions betrays the emphasis accorded transaction time in this model. □

DEFINITION: (Attribute Value Timestamped Data Model).

$$\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n)} \mid \exists x \in r\,(\forall i\ 1 \le i \le n\ \forall a \in x[A_i](t' \in a.\mathrm{T} \Rightarrow (a.V\ a.val) \in z[A_i]) \wedge \forall b \in z[A_i](\exists a \in x[A_i](t' \in a.T \wedge b.val = a.val \wedge b.V = a.V))\} \quad (8)$$

$$\tau^{\mathrm{B}}_{t}(r) = \{z^{(n)} \mid \exists x \in r\,(\forall i\ 1 \le i \le n\ \forall a \in x[A_i](t \in a.V \Rightarrow (a.T\ a.val) \in z[A_i]) \wedge \forall b \in z[A_i](\exists a \in x[A_i](t \in a.V \wedge b.val = a.val \wedge b.T = a.T))\} \quad (9)$$

For each, the first line ensures that no chronon is left unaccounted for, and the second line ensures that no spurious chronons are introduced. □

For each of the three representation schemes, the rollback operator for transaction-time relations (ρ^{T}) and the timeslice operator for valid-time relations (τ^{V}) are straightforward special cases of these definitions. Note that the rollback and timeslice operators in the various representations all have the same names, ρ^{B}_t and τ^{B}_t. For the remainder of the paper, we will use the tuple-timestamped representation as a focus, and this representation will be assumed for these operators.

5.2 Snapshot Equivalence

We can now define snapshot equivalence so that it applies to each of the three representations.

DEFINITION: Two relation instances, r and s, are *snapshot equivalent*, $r \overset{S}{\equiv} s$, if for all times t_1 not exceeding NOW and all times t_2,

$$\tau^{V}_{t_2}(\rho^{B}_{t_1}(r)) = \tau^{V}_{t_2}(\rho^{B}_{t_1}(s)). \tag{10}$$

□

The concept of snapshot equivalence generalizes the notion of *weakly equivalent* valid time relations (c.f., [Gad88]). We use a new term to avoid confusion with the different notion of *weak equivalence* (e.g., [Ull82]).

There is no reason to apply ρ before τ in the definition of snapshot equivalence, as the following theorem states. Proofs of all theorems may be found in the appendix.

THEOREM **1** Let r be a bitemporal relation. Then for all times t_1 not exceeding NOW and for all times t_2,

$$\tau^{V}_{t_2}(\rho^{B}_{t_1}(r)) \overset{S}{\equiv} \rho^{T}_{t_1}(\tau^{B}_{t_2}(r)).$$ □

Snapshot equivalence precisely captures the notion that relation instances in the representation scheme have the same information content. More precisely, all representations of the same conceptual bitemporal relation instance are snapshot equivalent, and two bitemporal relations that are snapshot equivalent represent the same conceptual bitemporal relation.

THEOREM **2** Snapshot equivalent bitemporal relations represent the same conceptual bitemporal relation:

1. If $\texttt{conceptual\_to\_snap}(r', cover_1) = r_1$ and $\texttt{conceptual\_to\_snap}(r', cover_2) = r_2$, then $r_1 \overset{S}{\equiv} r_2$.
2. If $s_1 \overset{S}{\equiv} s_2$ then $\texttt{snap\_to\_conceptual}(s_1) = \texttt{snap\_to\_conceptual}(s_2)$. □

This theorem has important consequences. For each representation, and for a given covering function, snapshot equivalence partitions the relation instances into equivalence classes. Each instance in an equivalence class maps to the same conceptual bitemporal relation instance. The semantics of the representation instance is thus identical to that of the conceptual instance. This correspondence provides a way of converting instances between representations: this conversion can proceed through a conceptual instance. Finally, the correspondence provides a way of demonstrating that two instances in different representations are semantically equivalent, again by examining the conceptual instance(s) to which they map. For example, it may be shown that the representation instances given in Sections 3.1 through 3.3 are semantically equivalent to the conceptual bitemporal relation given in Section 2.1, and are thus semantically equivalent to each other.

6 An Algebra for Conceptual Bitemporal Relations

We now examine the operational aspects of the data models just introduced. A major goal is to demonstrate the existence of the operational counterpart of the structural equivalence established in the previous section. We first define operations on conceptual bitemporal relations and then define corresponding operations on the tuple-timestamped representation. We prove that the operators preserve snapshot equivalence and are natural generalizations of their snapshot counterparts. Finally, we examine two transformations that manipulate coverings in representations of bitemporal relation instances.

6.1 Definition

Define a relation schema $R = (A_1, \ldots, A_n, \mathrm{T})$, and let r be an instance of this schema. Let t denote an arbitrary time value and let t' denote a time not exceeding NOW. Then the timeslice and rollback operators, explained in Section 5.1, may be defined as follows.

$$\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n+1)} \mid \exists x \in r\,(z[A] = x[A] \wedge t' \in \mathit{elem\_1}(x[\mathrm{T}]) \wedge z[\mathrm{T}_v] = \mathit{elem\_2}(x[\mathrm{T}]))\} \quad (11)$$

$$\tau^{\mathrm{B}}_{t}(r) = \{z^{(n+1)} \mid \exists x \in r\,(z[A] = x[A] \wedge t \in \mathit{elem\_2}(x[\mathrm{T}]) \wedge z[\mathrm{T}_t] = \mathit{elem\_1}(x[\mathrm{T}]))\} \quad (12)$$

Here, *elem_1* selects all the transaction time chronons from a bitemporal element, and *elem_2* selects all the valid time chronons.

Let D be an arbitrary set of $|D|$ non-timestamp attributes of relation schema R. The projection on D of r, $\pi^{\mathrm{B}}_D(r)$, is defined as follows.

$$\pi^{\mathrm{B}}_D(r) = \{z^{(|D|+1)} \mid \exists x \in r\,(z[D] = x[D]) \wedge \forall y \in r\,(y[D] = z[D] \Rightarrow y[\mathrm{T}] \subseteq z[\mathrm{T}]) \wedge \forall t \in z[\mathrm{T}]\,\exists y \in r\,(y[D] = z[D] \wedge t \in y[\mathrm{T}])\} \quad (13)$$

The first line ensures that no chronon in any value-equivalent tuple of r is left unaccounted for, and the second line ensures that no spurious chronons are introduced.

Let P be a predicate defined on $A_1, \ldots, A_n$. The selection P on r, $\sigma^{\mathrm{B}}_P(r)$, is defined as follows.

$$\sigma^{\mathrm{B}}_P(r) = \{z \mid \exists x \in r\,(z = x \wedge P(x[A]))\} \quad (14)$$

To define the union operator, $\cup^{\mathrm{B}}$, let both r_1 and r_2 be instances of R.

$$r_1 \cup^{\mathrm{B}} r_2 = \{z^{(n+1)} \mid (\exists x \in r_1\,\exists y \in r_2\,(z[A] = x[A] = y[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \cup y[\mathrm{T}])) \vee (\exists x \in r_1\,(z[A] = x[A] \wedge (\neg\exists y \in r_2(y[A] = x[A])) \wedge z[\mathrm{T}] = x[\mathrm{T}])) \vee (\exists y \in r_2\,(z[A] = y[A] \wedge (\neg\exists x \in r_1(x[A] = y[A])) \wedge z[\mathrm{T}] = y[\mathrm{T}]))\} \quad (15)$$

The first clause handles value-equivalent tuples found in both r_1 and r_2; the second clause handles those found only in r_1; and the third handles those found only in r_2.

With r_1 and r_2 defined as above, relational difference is defined as follows.

$$r_1 -^{\mathrm{B}} r_2 = \{z^{(n+1)} \mid \exists x \in r_1\,(z[A] = x[A]) \wedge ((\exists y \in r_2\,(z[A] = y[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] - y[\mathrm{T}])) \vee (\neg\exists y \in r_2\,(z[A] = y[A]) \wedge z[\mathrm{T}] = x[\mathrm{T}]))\} \quad (16)$$

The last two lines compute the bitemporal element, depending on whether a value-equivalent tuple may be found in s.

In the bitemporal natural join, two tuples join if they match on the join attributes and have overlapping bitemporal element timestamps. Define r and s to be instances of R and S, respectively, and let R and S be bitemporal relation schemas given as follows.

$$R = (A_1, \ldots, A_n, B_1, \ldots, B_m, \mathrm{T})$$

$$S = (A_1, \ldots, A_n, C_1, \ldots, C_k, \mathrm{T})$$

The bitemporal natural join of r and s, $r \bowtie^{\mathrm{B}} s$, is defined below. As can be seen, the timestamp of a tuple in the join result is computed as the intersection of the timestamps of the two tuples that produced it.

$$r \bowtie^{\mathrm{B}} s = \{z^{(n+m+k+1)} \mid \exists x \in r\,\exists y \in s\,(x[A] = y[A] \wedge x[\mathrm{T}] \cap y[\mathrm{T}] \neq \emptyset \wedge z[A] = x[A] \wedge z[B] = x[B] \wedge z[C] = y[C] \wedge z[\mathrm{T}] = x[\mathrm{T}] \cap y[\mathrm{T}])\} \quad (17)$$

We have only defined operators for bitemporal relations. The similar operators for valid time and transaction time relations are special cases. The valid and transaction time natural joins are denoted $\bowtie^{\mathrm{V}}$ and $\bowtie^{\mathrm{B}}$, respectively; the conventional snapshot natural join is denoted $\bowtie^{\mathrm{S}}$. The same naming convention is used for the remaining operators.

6.2 Mapping the Algebra to a Representation Scheme

For each of the algebraic operators defined in the previous section, we now define counterparts for the first of the three representation schemes. Throughout this section, R and S denote tuple timestamped bitemporal relation schemas, and r and s are instances of these schemas. Initially, R is assumed to have the attributes $A_1, \ldots, A_n, \mathrm{T_s}, \mathrm{T_e}, \mathrm{V_s}, \mathrm{V_e}$.

As the rollback and timeslice operators were defined in Section 3.1, we only define projection, selection, union, difference, and natural join. We consider each operator in turn.

To define projection, let D be an arbitrary set of $|D|$ attributes among $A_1, \ldots, A_n$. The projection on D of r, $\pi^{\mathrm{B}}_D(r)$, is defined as follows.

$$\pi^{\mathrm{B}}_D(r) = \{z^{(|D|+4)} \mid \exists x \in r\ (z[D] = x[D] \wedge z[T] = x[T] \wedge z[V] = x[V])\} \tag{18}$$

Next, let P be a predicate defined on $A_1, \ldots, A_n$. The selection P on r, $\sigma^{\mathrm{B}}_P(r)$, is defined as follows.

$$\sigma^{\mathrm{B}}_P(r) = \{z^{(n+4)} \mid \exists x \in r\ (z = x \wedge P(x[A]))\} \tag{19}$$

To define the union operator, $\cup^{\mathrm{B}}$, let both r_1 and r_2 be instances of schema R.

$$r_1 \cup^{\mathrm{B}} r_2 = \{z^{(n+4)} \mid \exists x \in r_1\ \exists y \in r_2\ (z = x \vee z = y)\} \tag{20}$$

With r_1 and r_2 defined as above, relational difference is defined using several functions introduced in Section 3.1.

$$\begin{aligned} r_1 -^{\mathrm{B}} r_2 = \{z^{(n+4)} \mid \exists x \in r_1\ (&z[A] = x[A] \wedge \\ &\exists t \in cover(bi\_chr(x[\mathrm{T}], x[\mathrm{V}]) - \\ &\qquad \{bi\_chr(y[\mathrm{T}], y[\mathrm{V}]) \mid y \in r_2 \wedge y[A] = x[A]\}) \wedge \\ &z[\mathrm{T}_s] = min\_1(t) \wedge z[\mathrm{T}_e] = max\_1(t) \wedge \\ &z[\mathrm{V}_s] = min\_2(t) \wedge z[\mathrm{V}_e] = max\_2(t))\} \end{aligned} \tag{21}$$

The new timestamp is conveniently determined by set difference on bitemporal elements.

To define the bitemporal natural join, we need two bitemporal relation schemas R and S with overlapping attributes.

$$\begin{aligned} R &= (A_1, \ldots, A_n, B_1, \ldots, B_m, \mathrm{T_s}, \mathrm{T_e}, \mathrm{V_s}, \mathrm{V_e}) \\ S &= (A_1, \ldots, A_n, C_1, \ldots, C_k, , \mathrm{T_s}, \mathrm{T_e}, \mathrm{V_s}, \mathrm{V_e}) \end{aligned}$$

In the bitemporal natural join of r and s, $r \bowtie^{\mathrm{B}} s$, two tuples join if they match on the join attributes and overlap in both valid time and transaction time.

$$\begin{aligned} r \bowtie^{\mathrm{B}} s = \{z^{(n+m+k+4)} \mid \exists x \in r\ \exists y \in s\ (&z[A] = x[A] = y[A] \wedge x[\mathrm{T}] \cap y[\mathrm{T}] \neq \emptyset \wedge x[\mathrm{V}] \cap y[\mathrm{V}] \neq \emptyset \wedge \\ &z[B] = x[B] \wedge z[C] = y[C] \wedge \\ &z[\mathrm{T}] = x[\mathrm{T}] \cap y[\mathrm{T}] \wedge z[\mathrm{V}] = x[\mathrm{V}] \cap y[\mathrm{V}])\} \end{aligned} \tag{22}$$

6.3 Equivalence Properties

We have seen that a conceptual bitemporal relation is represented by a class of snapshot equivalent relations in the representation scheme. We now define the notion of an operator preserving snapshot equivalence.

DEFINITION: An operator α *preserves snapshot equivalence* if, for all parameters X and snapshot relation instances r and r' representing bitemporal relations,

$$r \stackrel{S}{\equiv} r' \Rightarrow \alpha_X(r) \stackrel{S}{\equiv} \alpha_X(r'). \quad (23)$$

This definition may be trivially extended to operators that accept two or more argument relation instances. □

In the snapshot relational algebra, an operator, e.g., natural join, must return identical results every time it is applied to the same pair of arguments. In our framework, we require only preservation of snapshot equivalence. Thus, we add flexibility in implementing the bitemporal operators by accepting that they return different, but snapshot equivalent, results when applied to identical arguments at different times.

The operators preserve snapshot equivalence. That is, given snapshot equivalent operands each operator produces snapshot equivalent results. This ensures that the result of an algebraic operation will be correct, irrespective of covering.

THEOREM **3** The algebraic operators preserve snapshot equivalence. □

The next step is to combine the conceptual and representation level transformation functions with the representation level operators to create corresponding conceptual level operators. Given a representation level operator, α^p, its corresponding conceptual level operators, α^{pc}, is defined as follows.

$$\alpha^{pc}_X(r') = \texttt{snap\_to\_conceptual}(\alpha^p_X(\texttt{conceptual\_to\_snap}(r')))$$

Theorems 2 and 3 in combination make this meaningful and ensure that the conceptual level operators behave like the snapshot relational algebra operators—with identical arguments, they always return identical results. This is required because, like snapshot relations, conceptual bitemporal relations are unique, i.e., two conceptual relations have the same information content if and only if they are identical.

Now, we have two sets of operators defined on the conceptual bitemporal relations, namely the directly defined operators in Section 6.1 and the induced operators. In fact, we have constructed the two sets of operators to be identical. Put differently, the operators in Section 6.1 are the explicitly stated conceptual level operators, induced from the representation level operators (Section 6.2) and the transformation algorithms in Section 3.1.

Next we show how the operators in the various data models, snapshot, transaction-time, valid-time, and bitemporal, are related. Specifically, we show that the semantics of an operator in a more complex data model reduces to the semantics of the operator in a simpler data model. Reducibility guarantees that the semantics of simpler operators are preserved in their more complex counterparts.

For example, the semantics of the transaction-time natural join reduces to the semantics of the snapshot natural join in that the result of first joining two transaction-time relations and then transforming the result to a snapshot relation yields a result equivalent to that obtained by first transforming the arguments to snapshot relations and then joining the snapshot relations. This is shown in Figure 4 and stated formally in the first equivalence of the following theorem.

THEOREM **4** Let t denote an arbitrary time that, when used with a rollback operator, does not exceed NOW. In each equivalence, let r and s be relation instances of the proper types for the given operators. Then the following hold.

$$\begin{aligned}
\rho^{\mathrm{T}}_t(r \bowtie^{\mathrm{T}} s) &\stackrel{S}{\equiv} \rho^{\mathrm{T}}_t(r) \bowtie^{\mathrm{S}} \rho^{\mathrm{T}}_t(s) \\
\tau^{\mathrm{V}}_t(r \bowtie^{\mathrm{V}} s) &\stackrel{S}{\equiv} \tau^{\mathrm{V}}_t(r) \bowtie^{\mathrm{S}} \tau^{\mathrm{V}}_t(s) \\
\tau^{\mathrm{B}}_t(r \bowtie^{\mathrm{B}} s) &\stackrel{S}{\equiv} \tau^{\mathrm{B}}_t(r) \bowtie^{\mathrm{T}} \tau^{\mathrm{B}}_t(s) \\
\rho^{\mathrm{B}}_t(r \bowtie^{\mathrm{B}} s) &\stackrel{S}{\equiv} \rho^{\mathrm{B}}_t(r) \bowtie^{\mathrm{V}} \rho^{\mathrm{B}}_t(s)
\end{aligned}$$

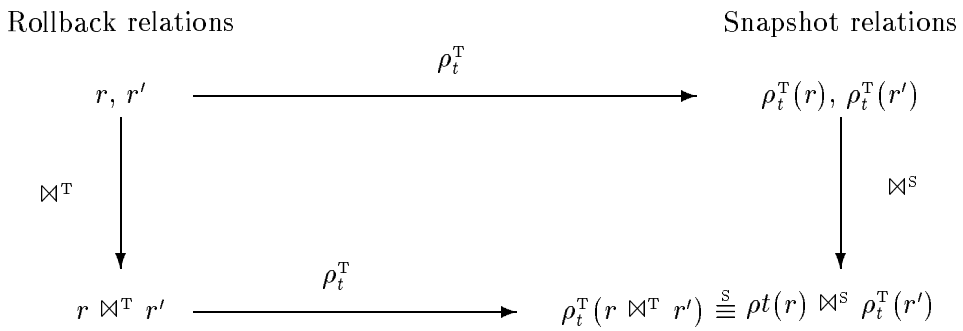

Figure 4: Reducibility of Rollback Natural Join to Snapshot Outer Natural Join.

A similar analysis can be made for the other operators. □

6.4 Covering Transformations

When a conceptual bitemporal relation is mapped to a representation scheme, a covering function is employed to represent bitemporal elements by sets of rectangles. The mappings were the topic of Sections 3.1 to 3.3, and different types of covering functions were discussed in Section 3.4. We now define two transformations that can change the covering in a representation without affecting the results of queries, as the transformations preserve snapshot equivalence. Both are generalizations of simpler transformations used in valid time data models.

The first transformation is termed coalescing. Informally, it states that two temporally overlapping or adjacent value-equivalent tuples may be collapsed into a single tuple [Sno87]. Coalescing may help reduce the number of tuples necessary for representing a bitemporal relation, and, as such, is a space optimization. We formally define coalescing and show that it preserves snapshot equivalence.

DEFINITION: *Coalescing*. Let $x = (a_1, \ldots, a_n, t_1, t_2, v_1, v_2)$ and $x' = (a_1, \ldots, a_n, t_3, t_4, v_3, v_4)$ be two distinct tuples belonging to the same bitemporal relation instance.

First, if $x[\mathrm{T}] = x'[\mathrm{T}]$ and $x[\mathrm{V}] \cup x'[\mathrm{V}] = [\min(v_1, v_3), \max(v_2, v_4)]$, the two tuples may be *coalesced* into the single tuple $y = (a_1, \ldots, a_n, t_1, t_2, \min(v_1, v_3), \max(v_2, v_4))$. Second, if $x[\mathrm{V}] = x'[\mathrm{V}]$ and $x[\mathrm{T}] \cup x'[\mathrm{T}] = [\min(t_1, t_3), \max(t_2, t_4)]$, the two tuples may be *coalesced* into the single tuple $y' = (a_1, \ldots, a_n, \min(t_1, t_3), \max(t_2, t_4), v_1, v_2)$.

A bitemporal relation instance is *coalesced* if no pair of tuples may be coalesced. □

THEOREM **5** Coalescing preserves snapshot equivalence. □

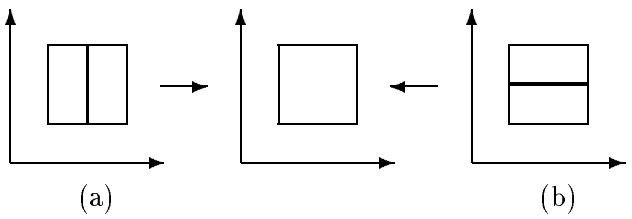

Figure 5: Coalescing

Coalescing of overlapping, value-equivalent tuples is illustrated in Figure 5. The figure shows how rectangles may be combined when overlap or adjacency occurs in transaction time (a) or

valid time (b). Note that it is only possible to coalesce rectangles when the result is a bitemporal rectangle. Compared to valid time relations with only one time dimension, this severely restricts the applicability of coalescing.

We now formalize the notion that a relation may have repeated information among tuples.

DEFINITION: A bitemporal relation instance r has *repetition of information* if it contains two distinct tuples $x = (a_1, \ldots, a_n, t_1, t_2, v_1, v_2)$ and $x' = (a_1, \ldots, a_n, t_3, t_4, v_3, v_4)$ such that $x[\mathrm{T}] \cap x'[\mathrm{T}] \neq \emptyset \wedge x[\mathrm{V}] \cap x'[\mathrm{V}] \neq \emptyset$. A relation with no such tuples has no repetition of information. □

While coalescing may both reduce the number of rectangles and reduce repetition of information, its applicability is restricted. The next equivalence preserving transformation may be employed to completely eliminate temporally redundant information, at the expense of adding extra tuples. We first define the transformation and then describe its properties.

DEFINITION: *Elimination of repetition.* With x and x' as in the definition above, the information in tuple y, defined below, is contained in both x and x'.

$$y = (a_1, \ldots, a_n, \max(t_1, t_3), \min(t_2, t_4), \max(v_1, v_3), \min(v_2, v_4))$$

The repetition incurred by x and x' may be eliminated by replacing tuples x and x' by the set of tuples, s, defined below.

$$\begin{aligned} 1 \qquad s \;=\; \{z^{(n+4)} \mid z[A] = x[A] \wedge ((z[\mathrm{T}] \in \mathit{cover}_t^{max}(x[\mathrm{T}] - x'[\mathrm{T}]) \wedge z[\mathrm{V}] = x[\mathrm{V}]) \vee \\ 2 \qquad\qquad (z[\mathrm{T}] \in \mathit{cover}_t^{max}(x'[\mathrm{T}] - x[\mathrm{T}]) \wedge z[\mathrm{V}] = x'[\mathrm{V}]) \vee \\ 3 \qquad\qquad (z[\mathrm{T}] = x[\mathrm{T}] \cap x'[\mathrm{T}] \wedge z[\mathrm{V}] = x[\mathrm{V}] \cup x'[\mathrm{V}]))\} \end{aligned} \tag{24}$$

The function cover_t^{max} transforms an argument set of transaction time chronons into a set of maximal intervals of consecutive chronons. □

THEOREM **6** The elimination of repetition transformation has the following properties.

1. It eliminates repetition among two argument tuples.
2. The result, s, has at most three tuples.
3. It is equivalence preserving.
4. Repeated application produces a relation instance with no repetition of information. □

The transformation partitions the regions covered by the argument rectangles on transaction time. The symmetric transformation, which partitions on valid time, may also be included. These transformations are illustrated in parts (a) and (b), respectively, of Figure 6.

The elimination of repetition of information may increase the number of tuples in a representation. The transformation may still be desirable because subsequent coalescing may be possible and, more importantly, because certain updates are simplified.

7 Summary and Future Research

In this paper, we defined a *conceptual data model* that timestamps facts with bitemporal elements, which are sets of bitemporal chronons. We showed that it is a unifying model in that conceptual instances could be mapped into instances of three existing *representational data models*, a first

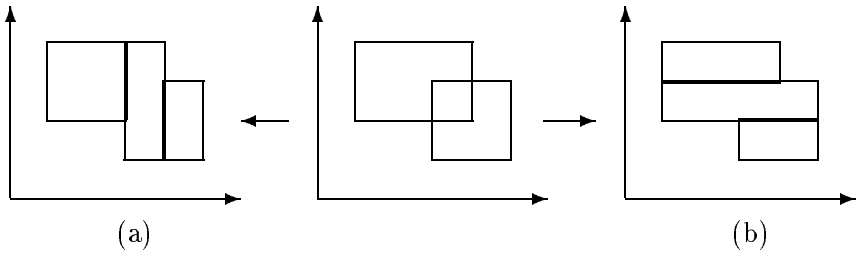

Figure 6: Eliminating Representational Repetition of Information

normal form (1NF) tuple timestamped data model, a data model based on 1NF timestamped change requests recorded in backlog relations, and a non-1NF data model in which attribute values were stamped with rectangles in transaction-time/valid-time space. We also showed how an extension to the conventional relational algebraic operators could be defined in the conceptual data model, and be mapped to analogous operators in the representational models.

An important property of the conceptual model shared with the conventional relational model, but not held by the representational models, is that relation instances are semantically unique, each models a different reality and thus has a distinct semantics. We employed *snapshot equivalence* to relate instances in these four models, and showed that the operators were equivalent, were snapshot preserving, and were a natural extension of the snapshot operators. Finally, we discussed covering functions at different points along the space-time tradeoff, and presented two types of transformations altering coverings of bitemporal relation representations.

We advocate a separation of concerns. Data presentation, storage representation, and time-varying semantics should be considered in isolation, utilizing different data models. Semantics, specifically as determined by logical database design, should be expressed in the conceptual model. Multiple presentation formats should be available, as different applications require different ways of viewing the data. The storage and processing of bitemporal relations should be done in a data model that emphasizes efficiency.

Additional research is needed in database design, utilizing the conceptual data model. It appears that normal forms may be more conveniently defined in this model than in the representational models. Also, more work is needed in mapping existing temporal query language proposals into the conceptual data model.

Acknowledgements

This research was conducted while the first author visited the University of Arizona. Support was provided by the Danish Natural Science Research Council through grant no. 11-9675-1 SE, the National Science Foundation through grant IRI-8902707, the IBM Corporation through Contract #1124, and by Christian and Otilia Brorsons Mindelegat.

References

[BADW82] A. Bolour, T.L. Anderson, L.J. Dekeyser, and H.K.T. Wong. The Role of Time in Information Processing: A Survey. *SigArt Newsletter*, 80:28–48, April 1982.

[BG89] G. Bhargava and S. Gadia. Achieving Zero Information Loss in a Classical Database Environment. In *Proceedings of the Conference on Very Large Data Bases*, pages 217–224, Amsterdam, August 1989.

[BZ82] J. Ben-Zvi. *The Time Relational Model*. PhD thesis, Computer Science Department, UCLA, 1982.

[CC87] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *Proceedings of the International Conference on Data Engineering*, pages 528–537, Los Angeles, CA, February 1987.

[Gad88] S.K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[Gad92] S.K. Gadia. A seamless generic extension of SQL for querying temporal data. Technical Report TR-92-02, Computer Science Department, Iowa State University, March 1992.

[JMRS92] C.S. Jensen, L. Mark, N. Roussopoulos, and T. Sellis. Using Caching, Cache Indexing, and Differential Techniques to Efficiently Support Transaction Time. *VLDB Journal*, to appear, 1992.

[Kim78] K.A. Kimball. The Data System. Master's thesis, University of Pennsylvania, 1978.

[LJ88] N. Lorentzos and R. Johnson. Extending Relational Algebra to Manipulate Temporal Data. *Information Systems*, 13(3):289–296, 1988.

[McK86] E. McKenzie. Bibliography: Temporal Databases. *ACM SIGMOD RECORD*, 15(4):40–52, December 1986.

[MS91] E. McKenzie and R. Snodgrass. Supporting Valid Time in an Historical Relational Algebra: Proofs and Extensions. Technical Report TR–91–15, Department of Computer Science, University of Arizona, Tucson, AZ, August 1991.

[NA89] S. B. Navathe and R. Ahmed. A Temporal Relational Model and a Query Language. *Information Sciences*, 49:147–175, 1989.

[SA85] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of ACM SIGMOD*, pages 236–246, 1985.

[Sad87] R. Sadeghi. *A Database Query Language for Operations on Historical Data*. PhD thesis, Dundee College of Technology, Dundee, Scotland, December 1987.

[Sar90] N. Sarda. Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):220–230, June 1990.

[Sno87] R. Snodgrass. The Temporal Query Language TQUEL. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Sno90] R. Snodgrass. Temporal Databases: Status and Research Directions. *ACM SIGMOD Record*, 19(4):83–89, December 1990.

[Sno93] R. Snodgrass. An Overview of TQuel, in Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings Pub. Co., to appear, 1993.

[Soo91] M. D. Soo. Bibliography on Temporal Databases. *ACM SIGMOD Record*, 20(1):14–23, March 1991.

[SS88] R. Stam and R. Snodgrass. A Bibliography on Temporal Databases. *Database Engineering*, 7(4):231–239, December 1988.

[Tan86] A.U. Tansel. Adding Time Dimension to Relational Model and Extending Relational Algebra. *Information Systems*, 11(4):343–355, 1986.

[Ull82] J.D. Ullman. *Principles of Database Systems, Second Edition.* Computer Science Press, Potomac, Maryland, 1982.

Appendix: Proofs of Theorems

Since the rollback and timeslice operations are defined separately for each representation, the proofs involving these operators must necessarily be representation specific. While we utilize only the tuple-timestamped representation, analogous proofs apply to the other representations.

THEOREM **1** Let r be a temporal relation. Then for all times t_1 not exceeding NOW and for all times t_2,

$$\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r)) \stackrel{\mathrm{S}}{\equiv} \rho^{\mathrm{T}}_{t_1}(\tau^{\mathrm{B}}_{t_2}(r)).$$

PROOF: Let $x \in \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r))$; then there is a tuple y in $\rho^{\mathrm{B}}_{t_1}(r)$ with $y[A] = x[A]$ and $t_2 \in y[\mathrm{V}]$. This implies the existence of a tuple z in r so that $z[A] = y[A]$, $z[\mathrm{V}] = y[\mathrm{V}]$, and $t_1 \in z[\mathrm{T}]$. As $t_2 \in z[\mathrm{V}]$, there is a tuple u in $\tau^{\mathrm{B}}_{t_2}(r)$ for which $u[A] = z[A]$ and $u[\mathrm{T}] = z[\mathrm{T}]$. As $t_1 \in u[\mathrm{T}]$, there is a tuple v in $\rho^{\mathrm{T}}_{t_1}(\tau^{\mathrm{B}}_{t_2}(r))$ with $v[A] = u[A]$. By construction, $v = x$. Thus, a tuple on the lhs (left hand side) is also on the rhs (right hand side). Proving the opposite inclusion is similar and omitted. Combining the inclusions proves the equivalence. □

In the proof of the following theorem, the notion of snapshot subset is utilized.

DEFINITION: A temporal relation instance, r, is a *snapshot subset* of a temporal relation instance, s, $r \stackrel{\mathrm{S}}{\subseteq} s$, if for all times t_1 not exceeding NOW and all times t_2,

$$\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r)) \subseteq \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(s)). \tag{25}$$

Further, a temporal query expression, Q_1, is a *snapshot subset* of a temporal query expression, Q_2, $Q_1 \stackrel{\mathrm{S}}{\subseteq} Q_2$, if all instantiations of Q_1 are snapshot subsets of all instantiations of Q_2. □

THEOREM **2** Snapshot equivalent temporal relations represent the same conceptual temporal relation:

1. If `conceptual_to_snap`$(r', cover_1) = r_1$ and `conceptual_to_snap`$(r', cover_2) = r_2$, then $r_1 \stackrel{\mathrm{S}}{\equiv} r_2$.
2. If $s_1 \stackrel{\mathrm{S}}{\equiv} s_2$ then `snap_to_conceptual`(s_1) = `snap_to_conceptual`(s_2).

PROOF: We prove the two implications in turn. To prove that r_1 and r_2 are snapshot equivalent, we prove that r_1 is a snapshot subset of r_2, and conversely. We need to show that $\forall t_1, t_2, x \in \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_1))$, $x \in \tau^{\mathrm{V}}_{t_2}\rho^{\mathrm{B}}_{t_1}(r_2))$. Let tuple x be in $\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_1))$. By the definition of rollback and timeslice, a set of tuples x_i exist in r_1 with $x_i[A] = x$ and $t_1 \in x_i[\mathrm{T}]$ and $t_2 \in x_i[\mathrm{V}]$. By the premise and the definition of `conceptual_to_snap`, a single tuple x' exists in r' with $x'[A] = x_i[A]$ and so that $x'[\mathrm{T}]$ contains exactly the bitemporal chronons covered by the x_i. Further, the bitemporal chronon (t_2, t_1) must be in $x'[\mathrm{T}]$. In general, an application of `conceptual_to_snap` to x' will then result in a set of tuples y_j, each with $y_j[A] = x'[A]$. For at least one of the y_j, it must be true that

$t_1 \in y_j[\mathrm{T}]$ and $t_2 \in y_j[\mathrm{V}]$ (the first requirement). Therefore, tuple $y = x'[A]$ must be in $\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_2))$. Since $y = x$, r_1 is a snapshot subset of r_2. Due to symmetry, proving the reverse is similar.

To prove the second implication, pick an arbitrary tuple x in some snapshot of s_1 and let (t_i, t_j) be the set of pairs of valid and transaction times so that x is in $\tau^{\mathrm{V}}_{t_i}(\rho^{\mathrm{B}}_{t_j}(s_1))$. (This is simply the bitemporal element in s_i corresponding to the fact x.) By the premise and the definition of snapshot equivalence, the set of pairs (t'_i, t'_j) such that x is in $\tau^{\mathrm{V}}_{t'_i}(\rho^{\mathrm{B}}_{t'_j}(s_2))$ must be identical to the set (t_i, t_j). In general, these sets of pairs are covered by different sets of rectangles in s_1 and s_2. However, the function `snap_to_conceptual` simply accumulates the covered pairs (corresponding to bitemporal chronons) in sets, rendering the particular covering by rectangles immaterial. □

THEOREM **3** The algebraic operators preserve snapshot equivalence. Specifically, let $r \overset{\mathrm{S}}{\equiv} r'$ and $s \overset{\mathrm{S}}{\equiv} s'$. Then

$$\begin{aligned}
r \bowtie^{\mathrm{V}} s &\overset{\mathrm{S}}{\equiv} r' \bowtie^{\mathrm{V}} s' \\
r \bowtie^{\mathrm{B}} s &\overset{\mathrm{S}}{\equiv} r' \bowtie^{\mathrm{B}} s' \\
\sigma^{\mathrm{B}}_P(r) &\overset{\mathrm{S}}{\equiv} \sigma^{\mathrm{B}}_P(r') \\
\pi^{\mathrm{B}}_D(r) &\overset{\mathrm{S}}{\equiv} \pi^{\mathrm{B}}_D(r') \\
r \cup^{\mathrm{B}} s &\overset{\mathrm{S}}{\equiv} r' \cup^{\mathrm{B}} s' \\
r -^{\mathrm{B}} s &\overset{\mathrm{S}}{\equiv} r' -^{\mathrm{B}} s'.
\end{aligned}$$

PROOF: As before, we proceed by demonstrating snapshot subsets. To prove the first equivalence, let tuple x be in the lhs. By the definition of $\bowtie^{\mathrm{V}}$ there exists a set of tuples $x_i \in r$ with $x_i[AB] = x[AB]$ and so that $\cup_i x_i[\mathrm{V}] \supseteq x[\mathrm{V}]$. Similarly, there exists a set of tuples $x_j \in s$ with $x_j[AC] = x[AC]$ and so that $\cup_j x_j[\mathrm{V}] \supseteq x[\mathrm{V}]$. Next, by the definition of $\overset{\mathrm{S}}{\equiv}$, for each $x_i \in r$ the exists a set of tuples $x^i_k \in r'$ with $x^i_k[AB] = x_i[AB]$ and so that $\cup_k x^i_k[\mathrm{V}] \supseteq x_i[\mathrm{V}]$. The set x^i_k covers x_i. For each j a similar set x^j_l exists that covers x_j. Applying $\bowtie^{\mathrm{V}}$ to the sets of tuples $x^i_k \in r'$ and $x^j_l \in s'$ yields a set of tuples x_m with $x_m[AB] = x[AB]$ and so that $\cup_m x_m[\mathrm{V}] \supseteq x[\mathrm{V}]$. This proves that any tuple in a snapshot made from the lhs will also be present in the same snapshot made from the rhs. By symmetry, the reverse is also true, and the equivalence follows.

The proofs of the other equivalences are similar. □

THEOREM **4** Let t denote an arbitrary time that, when used with a rollback operator, does not exceed NOW. In each equivalence, let r and s be relation instances of the proper types for the given operators. Then the following hold.

$$\begin{aligned}
\rho^{\mathrm{T}}_t(r \bowtie^{\mathrm{T}} s) &\overset{\mathrm{S}}{\equiv} \rho^{\mathrm{T}}_t(r) \bowtie^{\mathrm{S}} \rho^{\mathrm{T}}_t(s) \\
\tau^{\mathrm{V}}_t(r \bowtie^{\mathrm{V}} s) &\overset{\mathrm{S}}{\equiv} \tau^{\mathrm{V}}_t(r) \bowtie^{\mathrm{S}} \tau^{\mathrm{V}}_t(s) \\
\tau^{\mathrm{B}}_t(r \bowtie^{\mathrm{B}} s) &\overset{\mathrm{S}}{\equiv} \tau^{\mathrm{B}}_t(r) \bowtie^{\mathrm{T}} \tau^{\mathrm{B}}_t(s) \\
\rho^{\mathrm{B}}_t(r \bowtie^{\mathrm{B}} s) &\overset{\mathrm{S}}{\equiv} \rho^{\mathrm{B}}_t(r) \bowtie^{\mathrm{V}} \rho^{\mathrm{B}}_t(s)
\end{aligned}$$

PROOF: An equivalence is shown by proving its two inclusions separately. The non-timestamp attributes of r and s are AB and AC, respectively, where A, B, and C are sets of attributes and A denotes the join attribute(s).

We prove the fourth equivalence. The proofs of the remaining equivalences are similar and are omitted. Let $x'' \in$ lhs. Then there is a tuple $x' \in r \bowtie^{\mathrm{B}} s$ such that $x'[ABC] = x''$ and $t \in x'[\mathrm{T}]$.

By the definition of $\bowtie^{\mathrm{B}}$, there exists tuples $x_1 \in r$ and $x_2 \in s$ such that $x_1[A] = x_2[A] = x'[A]$, $x_1[B] = x'[B]$, $x_2[C] = x'[C]$, $x'[T] = x_1[T] \cap x_2[T]$, and $x'[V] = x_1[V] \cap x_2[V]$. By the definition of ρ_t^{B}, there exists a tuple $x_1' \in \rho_t^{\mathrm{B}}(r)$ such that $x_1' = x'[AB]$ and $x_1'[V] = x'[V]$ and a tuple $x_2' \in \rho_t^{\mathrm{B}}(s)$ such that $x_2' = x'[AC]$ and $x_2'[V] = x'[V]$. Then there exists $x_{12}'' \in$ rhs such that $x_{12}''[AB] = x_1'$, $x_{12}''[C] = x_2'[C]$, and $x_{12}''[V] = x_1'[V] \cap x_2'[V]$. By construction $x_{12}'' \overset{\mathrm{S}}{\equiv} x''$ (in fact, $x_{12}'' = x''$).

Now assume $x'' \in$ rhs. Then there exists tuples x_1' and x_2' in $\rho_t^{\mathrm{B}}(r)$ and $\rho_t^{\mathrm{B}}(s)$, respectively, such that $x_1' = x''[AB]$ and $x_2' = x''[AC]$ and $x''[V] = x_1'[V] \cap x_2'[V]$. This implies the existence of tuples $x_1 \in r$ and $x_2 ins$ and with $x_1[AB] = x_1'[AB]$, $x_1[V] = x_1[V]$, $t \in x_1[\mathrm{T}]$, $x_2[AC] = x_2'[AC]$, $x_2[V] = x_2'[V]$, and $t \in x_2[\mathrm{T}]$. There must exist a tuple $x' \in r \bowtie^{\mathrm{B}} s$ with $x'[AB] = x_1[AB]$, $x'[C] = x_2[C]$, $x'[V] = x_1[V] \cap x_2[V]$, and $t \in x'[\mathrm{T}]$. Consequently, there exists a tuple $x_{12}'' \in$ lhs such that $x_{12}'' = x'[ABC]$ and $x_{12}''[V] = x'[V]$. By construction, $x_{12}'' \overset{\mathrm{S}}{\equiv} x''$. □

The proof of the next theorem utilizes a subtle requirement on null values in bitemporal relations. Specifically, we require that null information not conflict with non-null information. If one tuple states that the value of an attribute is null then another, temporally concurrent tuple that contains non-null information for that attribute must not exist. More formally, we define this property as follows.

DEFINITION: *Consistency of null information.* Let two tuples x and x', both belonging to a relation instance r, be given as

$$\begin{aligned} x &= (a_1, \ldots, a_{i-1}, \perp, a_{i+1}, \ldots, a_n, t) \\ x' &= (a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n, t') \end{aligned}$$

where $a_i \neq \perp$ and t and t' are bitemporal element valued timestamps. If, for all such tuple pairs in r, it is the case that $t \cap t' = \emptyset$ then the null information in r is consistent. □

If a relation instance has consistent null information then a null value means that no other tuple has more definite information, and all relevant information is contained within a single tuple. When inconsistent null information is present, nothing may be inferred from encountering a null value, and the locality is jeopardized.

THEOREM **5** Coalescing preserves snapshot equivalence.
PROOF: Let r be a relation instance containing x and x' as given as in the definition of coalescing. In the first of the two cases, let relation s be identical to r, but with x and x' replaced by the tuple y as given in the definition. We must prove r and s snapshot equivalent. The tuples x and x' result in exactly the tuple $(a_1, \ldots, a_n)$ being present in all snapshots of r with a transaction time in $[t_1, t_2]$ and a valid time in $[\min(v_1, v_3), \max(v_2, v_4)]$. Similarly, the tuple y results in $(a_1, \ldots, a_n)$ being part of all snapshots of s with a transaction time in $[t_1, t_2]$ and a valid time in $[\min(v_1, v_3), \max(v_2, v_4)]$. The requirement that null information be genuine ensures this even in the case when there are nulls among the a_i. The proof for the second of the two cases is similar. □

THEOREM **6** The elimination of repetition transformation has the following properties.

1. It eliminates repetition among two argument tuples.
2. The result, s, has at most three tuples.
3. It is equivalence preserving.

4. Repeated application produces a relation instance with no repetition of information.

PROOF: There is no repetition of information between the resulting tuples as they do not overlap in transaction time.

Let x and x' be given as in the definition of elimination of repetition and define $T_x = cover_t^{max}(x[\mathrm{T}] - x'[\mathrm{T}])$ and $T'_x = cover_t^{max}(x'[\mathrm{T}] - x[\mathrm{T}])$. Tuples x and x' are replaced by at most three tuples. Line 3 results in one tuple. Lines 1 and 2 collectively result in two tuples, for the following reasons. The set T_x has two elements when $x'[\mathrm{T}]$ contains no endpoints of x[T]. In this case T'_x is empty. The sets T_x and T'_x have both one element when $x'[\mathrm{T}]$ contains exactly one of the endpoints of $x[\mathrm{T}]$. Lastly, T_x is empty when $x'[\mathrm{T}]$ contains both endpoints of x[T]. In this case T'_x has two elements.

Being similar to that for coalescing, the proof of equivalence preservation is omitted.

The process of eliminating repetition is terminating because the new tuples that result from one transformation step cover strictly smaller intervals in the transaction time dimension. In addition, two tuples that cover only a single transaction time and have repeated information may be coalesced into a single tuple that would not be further partitioned. □