Unifying Temporal Data Models via a Conceptual Model

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

TR 93-31

September 20, 1993

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 be incapable 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 five representation schemes that support both valid and transaction time; these representations include both 1NF and non-1NF models. We use snapshot equivalence to relate the representational data models with the bitemporal conceptual data model.

We then consider querying within the two-level framework. To do so, we define an algebra at the conceptual level. We then map this algebra to the representation level in such a way that new operators compute equivalent results for different representations of the same bitemporal conceptual 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 improve 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`

Unifying Temporal Data Models via a Conceptual Model

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 [Sno92, JS92]. 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 [JCG*92].

While these data models differ in many dimensions, perhaps the most often stated distinction is that 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 model specific database design approaches and implementation strategies.

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 exploiting equivalence mappings between the conceptual model and the *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 logical query optimization [JSS92].

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, the conceptual model is defined. We then examine five representational data models that have been previously proposed. These representational models can be classified as either tuple timestamping (e.g., [BZ82, NA89, Sad87, Sar90, Sno87, Sno93]), backlog-based (e.g., [Kim78, JMRS92]), or 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 space efficiency for 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 because 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 bitemporal conceptual model as well as one of the tuple-timestamped representational models. 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.

2 Bitemporal Conceptual Relations

The primary reason for the success of the relational model is its simplicity. A bitemporal relation is necessarily more complex than a conventional relation. 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, our aim is to extend the conventional relational model as small an extent as necessary to capture this additional information.

2.1 Definition

Tuples in a bitemporal conceptual 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 portion of reality being modeled, and transaction time models the update activity associated with 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 time domains are isomorphic to the domain of natural numbers.

In general, the schema of a bitemporal conceptual 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, $x = (a_1, a_2, \ldots, a_n | \, t_b)$, in a bitemporal conceptual 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. Thus, 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 predictively 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 *until_changed* (*UC*). On June 10th, the personnel department discovers an error. Jake had really been hired 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

[1] This term is a generalization of *temporal element*, used to denotes a set of single dimensional chronons [Gad88]. Alternative, equally desirable terms include *time period set* [BZ82] and *bitemporal lifespan* [CC87].

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 result of three updates to the relation, all of which take place on June 20th. While 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.

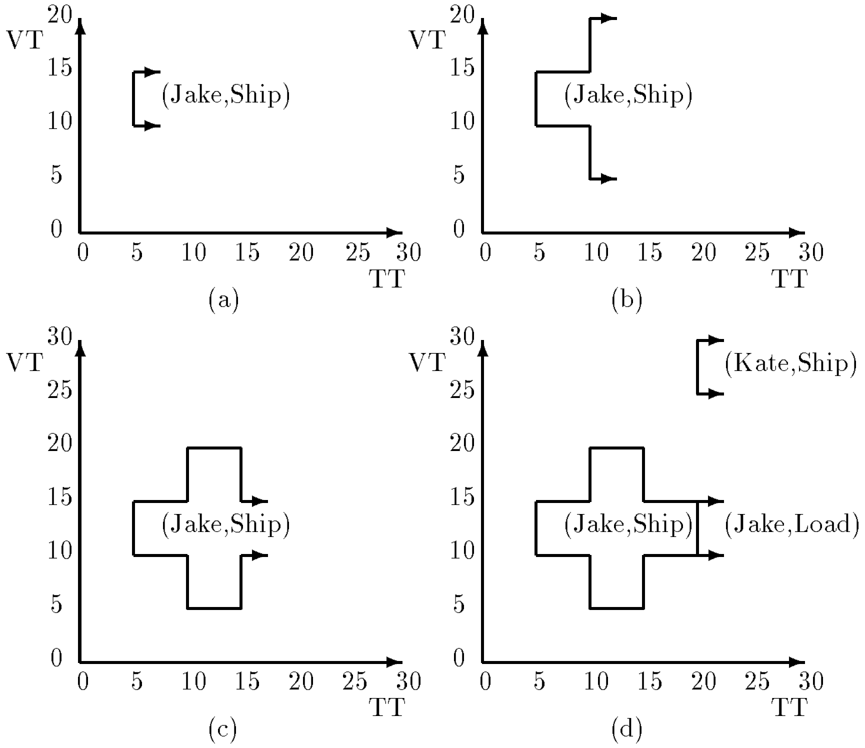

Figure 1: Bitemporal Elements

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 UC. 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	$\{(UC,10),\ldots,(UC,15)\}$
Kate	Ship	$\{(UC,25),\ldots,(UC,30)\}$

□

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, until its explicit attribute values are changed, the fact is current in the relation. This is indicated with a special transaction time value, UC.

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. The `Insert` routine 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.)

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

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

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. 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 (UC,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 (UC, 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 is current.

$$\texttt{delete}(r, (a_1, \ldots, a_n)) =$$
$$\begin{cases} r - \{(a_1, \ldots, a_n | \, t_b)\} \cup \{(a_1, \ldots, a_n | \, t_b - \texttt{uc\_ts}(t_b))\} & \text{if } \exists \, t_b \; ((a_1, \ldots, a_n | \, t_b) \in r) \\ r & \text{otherwise} \end{cases}$$

where $\texttt{uc\_ts}(t_b) = \{(UC, c_v) \mid (UC, c_v) \in t_b\}$.

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

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

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 or transaction time, respectively. Thus a 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 bitemporal conceptual 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 five representations of bitemporal relations that have been previously proposed. These representations constitute all relational bitemporal data models proposed to date. 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 bitemporal conceptual relations may be mapped into updates on relations in the representation.

3.1 Snodgrass' 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 [Sno87].

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. Then $\mathcal{R}$ is represented by a snapshot relation schema R as follows.

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

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

There is a multitude of possible ways of covering a bitemporal element. We require that any function that covers a bitemporal element $x[\mathrm{T}]$ of a bitemporal tuple x satisfy two properties.

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. We will revisit the topic of covering functions in Section 3.6.

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	*UC*	6/10	6/15
Kate	Ship	6/20	*UC*	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 $x[\mathrm{V}]$ 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 bitemporal conceptual 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.

```
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;
```

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 UC if encountered as a first component; otherwise, it returns a maximum first component. The function *max_2* selects a maximum second component. 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 generated tuple 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.

Note that the functions are the inverse of each other, i.e., for any conceptual relation instance r',

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

For the update routines, the most convenient covering functions partition on either valid or transaction time and do not permit overlaps. The current transaction time is c_t.

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

```
delete(r,(a_1,...,a_n)):
    for each x ∈ r
        if x[A] = (a_1,...,a_n) and x[T_e] = UC
            x[T_e] ← c_t;
    return r
```

The function $cover_v$ in the `insert` routine returns a set of valid-time intervals (each a set of contiguous valid-time chronons). The routine first reduces the valid time elements, produced by the covering function, to avoid overlap with the valid times of existing tuples that have a transaction time extending to UC and that are value equivalent to the one to be inserted. Then, one tuple is inserted for each of the remaining valid-time intervals. 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 `delete` and `insert`.

3.2 Jensen's 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_e 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 the 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 UC 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) ≠ UC
             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 ← z_1[V_s]; b ← z_1[V_e];
          c ← z_1[T]; d ← c_t + 1;
          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 ≠ c_t + 1
             r ← r − {z_3};
          x_1[T] ← bi_chr([c, d], [a, b]);
          if d = c_t + 1
             x_1[T] ← (x_1[T] ∪ {UC}) × {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 information recorded by the insertion request is still current, indicated by the ending transaction-time value, c_t+1. 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 the bitemporal conceptual 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 Gadia's 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 [Gad92].

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.

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

A tuple is composed of n sets. Each set element a is a triple of a transaction-time interval $[\mathrm{T}_s, \mathrm{T}_e]$, a valid-time interval $[\mathrm{V}_s, \mathrm{V}_e]$, 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, respectively.

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.

Emp		Dept	
$[20,UC] \times [25,30]$	Kate	$[20,UC] \times [25,30]$	Ship
$[5,9] \times [10,15]$	Jake	$[5,9] \times [10,15]$	Ship
$[10,14] \times [5,20]$	Jake	$[10,15] \times [5,20]$	Ship
$[15,19] \times [10,15]$	Jake	$[15,19] \times [10,15]$	Ship
$[20,UC] \times [10,15]$	Jake	$[20,UC] \times [10,15]$	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	
$[20,UC] \times [10,15]$	Jake	$[20,UC] \times [10,15]$	Load
$[5,9] \times [10,15]$	Jake	$[5,9] \times [10,15]$	Ship
$[10,14] \times [5,20]$	Jake	$[10,14] \times [5,20]$	Ship
$[15,19] \times [10,15]$	Jake	$[15,19] \times [10,15]$	Ship
$[20,UC] \times [25,30]$	Kate	$[20,UC] \times [25,30]$	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. Argument r' and *cover* are as described for the other representation schemes. Argument *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
        z ← (∅,...,∅);
        for each x ∈ g
            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] ← ∅;
            for each y ∈ z[A_i]
                t ← bi_chr(y.T, y.V);
                z[A_i] ← z[A_i] − {y};
                for each y' ∈ z[A_i]
                    if y.val = y'.val
                        t ← t ∪ bi_chr(y'.T, y'.V);
                        z[A_i] ← z[A_i] − {y'};
                g[i] ← g[i] ∪ {(y.val, t)};
        for each (a_1, a_2, ..., a_n) ∈ facts(g)
            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 *facts* function computes, for an array of attribute value/rectangle sets, all combinations of facts that can be constructed from those attribute values.

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

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

Insertion of a fact into an attribute-value timestamped relation can result in either of two actions. Either the new information is merged into an existing tuple $x \in r$ or no such x exists and the creation of an entirely new tuple is required.

The former case occurs when r is grouped so that x matches the explicit attribute values in exactly the grouping attributes, G. Placing the new information into x preserves the grouped structuring of relation. For any given attribute value $x[A_i]$, some or all of the information being inserted may already be present in $x[A_i]$. A triple y containing such information must match the information being inserted in the explicit attribute value a_i, be current in the database, and overlap in valid-time. We remove all such overlapping valid-times chronons, perform a covering of the remaining chronons, and insert triples into $x[A_i]$ for each element of the covering.

In the latter case, no tuple with matching grouping attributes is found. The new information cannot be merged into an existing tuple without violating the grouped structure of the relation. Therefore, a new tuple containing only the added information is created.

```
insert(r, (a_1, ..., a_n), t_v, cover_v):
    found ← FALSE;
    for each x ∈ r
        if x[G] = (a_1, ..., a_n)[G];
            found ← TRUE;
            for i ← 1 to n
                t' ← t_v;
                for each y ∈ x[A_i]
                    if y.val = a_i and y.T[e] = UC
                        t' ← t' − {y.V};
                for each t ∈ cover_v(t')
                    x[A_i] ← x[A_i] ∪ {([c_t, UC] '×' [min(t), max(t)] a_i)};
    if found = FALSE
        for each t ∈ cover_v(t_v)
            r ← r ∪ {{([c_t, UC] '×' [min(t), max(t)] a_1)} ... {([c_t, UC] '×' [min(t), max(t)] a_n)}};
    return r;
```

Deletion is more complicated. Removing a fact $(a_1, \ldots, a_n)$ from an attribute-valued timestamped relation r involves locating the tuple x containing the fact, if such an x exists, and altering x to reflect that the fact is no longer current. As we are interested only in current information, i.e., when $(a_1, \ldots, a_n)$ is current in the database, the triples in the attribute values of x that can participate in producing the fact must all have an ending transaction time of UC. The function *current* produces tuples from x representing the current information contained in x. It selects triples from each $x[A_i]$, $1 \leq i \leq n$, with an ending transaction time of UC and performs a Cartesian product, resulting in a relation whose tuples have attribute values each containing a single triple.

$$current(x) = \{((t_1 v_1 a_1), (t_2 v_2 a_2), \ldots, (t_n v_n a_n)) \mid \forall i \; 1 \leq i \leq n((t_i v_i a_i) \in x[A_i] \wedge UC \in t_i)\}$$

Each tuple y potentially has information that must be deleted from the current database state. This is the case if the explicit-attribute values of y match $(a_1, \ldots, a_n)$, and y contains a rectangle in bitemporal space where each of the triples $(t_i v_i a_i)$, $1 \leq i \leq n$, overlap. For each such y, we insert triples indicating that the fact has been deleted from the current database state, and, with the help of a covering function, reinsert unaffected information back into the relation.

```
delete(r, (a_1, ..., a_n), cover_v):
    for each x ∈ r
        z[A_i] ← ∅;  ...  z[A_n] ← ∅;
        for each y ∈ current(x)
            if y[A_1].val = a_1 and ... and y[A_n].val = a_n
                t_1 ← bi_chr(y[A_1].T, y[A_1].V);  ...  t_n ← bi_chr(y[A_n].T, y[A_n].V);
                t ← t_1 ∩ ... ∩ t_n;
                if t ≠ ∅
                    for i ← 1 to n
                        x[A_i] ← x[A_i] − {y[A_i]};
                        x[A_i] ← x[A_i] ∪ {([min_1(t), c_t − 1] '×' [min_2(t), max_2(t)] y[A_i].val)};
                        for each t' ∈ cover_v(t_i − t)
                            x[A_i] ← x[A_i] ∪ {([min_1(t'), max_1(t')] '×' [min_2(t), max_2(t)] y[A_i].val)};
    return r;
```

As before, `modify` is simply a combination of `insert` and `delete`.

3.4 McKenzie's Attribute Value Timestamped Representation Scheme

Like the representation of the previous section, McKenzie's data model uses non-1NF attribute-value timestamping [McK88, MS91].

In McKenzie's model, a bitemporal relation is a sequence of valid-time states indexed by transaction time. Tuples within a valid-time state are attribute-value timestamped. The timestamps associated with each attribute value are sets of chronons, i.e., valid-time elements. In addition, the model does not assume homogeneity—attributes within the same tuple may have different timestamps.

A bitemporal relation schema $\mathcal{R} = (A_1, \ldots, A_n \mid \mathrm{T})$ is represented by an attribute valued timestamped relation schema R as follows.

$$R \;=\; (\mathrm{T}, \mathrm{VR})$$

where VR is a valid-time relation, and T is the transaction time when VR became current in the database. Stepwise-constant semantics are assumed.

The schema of the valid-time state VR is as follows.

$$\mathrm{VR} \;=\; (A_1\mathrm{V}_1, \ldots, A_n\mathrm{V}_n)$$

Here A_1, …, A_n are explicit attribute values. Associated with each A_i, $1 \leq i \leq n$, is a valid-time element V_i denoting when A_i was true in the modeled reality.

Example: The sequence of valid-time states indexed by transaction time corresponding to the conceptual relation in Figure 1(d) is shown below.

T	VR
0	$\emptyset$
5	{(Jake {10,…,15}, Ship {10,…,15})}
10	{(Jake {5,…,20}, Ship {5,…,20})}
15	{(Jake {10,…,15}, Ship {10,…,15})}
20	{(Jake {10,…,15}, Load {10,…,15}), (Kate {25,…,30}, Ship {25,…,30})}

Notice that for each tuple in each valid-time state, the timestamps associated with the attribute values in a tuple are identical, i.e., the timestamps are homogeneous. As mentioned above, this is not required by the model, but in our example the values of the attributes Emp and Dept change synchronously, hence the timestamps associated with each are identical. □

Next, we consider the conversion between a bitemporal relation and its representation as a sequence of valid-time states in McKenzie's data model. As before, we exhibit two functions. The first maps conceptual instances into representational instances, and the second performs the inverse transformation.

```
conceptual_to_att2(r'):
    r ← ∅;
    uc_present ← FALSE;
    for each x ∈ r'
        for each (t, v) ∈ reduce(x[T]);
            if t = UC
                uc_present ← TRUE;
            else
                for i ← 1 to n
                    z[A_i] ← x[A_i];
                    z[T_i] ← v;
                r ← r ∪ {(t, {z})};
    if not uc_present
        r ← r ∪ {(c_t, ∅)};
    r ← r ∪ {(0, ∅)};
    r ← group(r);
    return r;
```

```
att2_to_conceptual(r):
    for each (t, vr) ∈ r
        vr ← homogenize(vr);
    reverse_sort(r);
    r' ← ∅;
    (t, vr) ← next(r);
    for each y ∈ vr
        z[A] ← y[A];
        z[T] ← bi_chr({t..c_t − 1, UC}, y[V]);
        r' ← r' ∪ {z};
    t_last ← t;  (t, vr) ← next(r);
    while (t, vr) ≠⊥
        for each y ∈ vr
            found ← FALSE;
            for each z' ∈ r'
                if z'[A] = y[A]
                    z'[T] ← z'[T] ∪ bi_chr({t..t_last − 1}, y[V]);
                    found ← TRUE;
            if not found
                z[A] ← y[A]
                z[T] ← bi_chr({t..t_last − 1}, y[V]);
                r' ← r' ∪ {z};
        t_last ← t;  (t, vr) ← next(r);
    return r';
```

The first function, `conceptual_to_att2`, takes a conceptual relation as its first argument and returns a sequence of valid-time relations, indexed by transaction time, in McKenzie's data model. A conceptual tuple x can contribute possibly many tuples to the result, with the generated tuples residing in possibly many different valid-time states. For example, the first tuple in the conceptual relation of Section 2.1 would contribute three tuples, (Jake $\{10..15\}$, Ship $\{10..15\}$), (Jake $\{5..20\}$, Ship $\{5..20\}$), (Jake $\{10..15\}$, Ship $\{10..15\}$), in the valid-time states associated with transaction times 5, 10 and 15, respectively. Value-equivalent tuples with identical valid-timestamps but at intermediate transaction times, e.g., (Jake $\{10..15\}$, Ship $\{10..15\}$) at transaction time 6, are not generated.

We accomplish this by deriving for each conceptual tuple x a set of stepwise constant states from its bitemporal element $x[\mathrm{T}]$. The result is a set of pairs (t,v), the first element being a transaction time and the second being a valid-time element. Effectively, each (t,v) denotes the state of $x[A]$ as being valid during the set v at the transaction time t. Intermediate states are not included in the computed set of pairs, effectively preserving the stepwise constant assumption.

The set of stepwise constant states is computed by the function `reduce` shown below. For the above example, `reduce` returns the set $\{(5,\{10..15\}), (10,\{5..20\}), (15,\{10..15\})\}$. The function `next_state` is called by `reduce`; it examines each bitemporal chronon in the timestamp and derives a state (t,v) where t is the earliest transaction time present in the timestamp, and v is the set containing exactly those valid-time chronons associated with t.

```
reduce(T):
    T' ← ∅;
    while T ≠ ∅
        (t, v) ← next_state(T);
        T' ← T' ∪ {(t, v)};
        T ← T − bi_chr({t}, v);
        t' ← t + 1;
        while (t', v) = next_state(T)
                T ← T − bi_chr({t'}, v);
                t' ← t' + 1;
    return T';
```

```
next_state(T):
    v ← ∅;
    t ← UC;
    for each b ∈ T
        if b.T < t
            v ← {b.V};
            t ← b.T;
        else
            if b.T = t
                v ← v ∪ {b.V};
    return (t, v);
```

For a given pair (t,v), a tuple is generated and placed in a valid-time state indexed by the transaction time t. The end result is a set of pairs of single tuple valid-time states indexed at the given by a transaction time.

Finally, the function `group` collapses all pairs with identical transaction-time components into a single valid-time state, indexed at the given transaction time.

```
group(r):
    S ← ∅;
    for each (t, vr) ∈ r;
        found ← FALSE;
        for each (t', vr') ∈ S
            if t = t'
                S ← S − (t', vr');
                S ← S ∪ {(t', vr' ∪ vr)};
                found ← TRUE;
        if not found
            S ← S ∪ {(t, vr)};
    return S;
```

The second function, `att2_to_conceptual`, performs the inverse transformation. It takes a sequence of valid-time states r, indexed by transaction time, and produces the equivalent conceptual relation.

As the valid-time states of r may contain tuples with non-homogeneous timestamps, we first transform each input valid-time state into an equivalent tuple-timestamped relation. This is the purpose of function `homogenize` shown below. For each tuple $x \in vr$, `homogenize` generates possibly many result tuples, one for each valid-time chronon present in a timestamp associated with an attribute value of x. The function determines the maximal set of attribute values simultaneously valid during that chronon, and generates a result tuple, whose tuple-timestamp contains the single chronon.

```
homogenize(vr):
    vr_h ← ∅;
    for each x ∈ vr
        for i ← 1 to n
            for each v ∈ x[V_i]
                z[A_1] ←⊥; ...  z[A_n] ←⊥;
                z[A_i] ← x[A_i];
                z[V] ← v;
                for j ← 1 to n
                    if j ≠ i and v ∈ x[V_j]
                        z[A_j] ← x[A_j]
                vr_h ← vr_h ∪ {z};
    return coalesce(vr_h);
```

```
coalesce(vr):
    vr' ← ∅;
    for each x ∈ vr
        vr ← vr − {x};
        for each y ∈ vr
            if x[A] = y[A]
                x[V] ← x[V] ∪ y[V];
                vr ← vr − {y};
        vr' ← vr' ∪ {x};
    return vr';
```

As many value-equivalent tuples may be produced, function `coalesce` is used to collapse such tuples into a single tuple. The timestamps of matching tuples are unioned into a single result tuple.

The valid-time states of r are then processed from latest to earliest in transaction time order; the pairs $(t, vr) \in r$ are sorted into descending order of t, and a function `next` returns the next (t, vr) in the sorted order. The current valid-time state is treated specially to accommodate the stepwise constant semantics between the time the state was stored, the current transaction time, and UC.

The remaining valid-time states are converted as follows. For a tuple $x \in vr$, its bitemporal timestamp is generated using the appropriate range of transaction-time and valid-time element associated with the tuple. However, since value-equivalent tuples may be present in different valid-time states, we must consolidate the information in such tuples within one resulting conceptual tuple. If a value-equivalent tuple z' is already present in the result, we augment its timestamp with the generated bitemporal element. Otherwise, a new tuple is inserted.

We now show how the semantics of bitemporal update are supported within this representation. Insertion of a fact into the database involves the creation of a new current state containing the fact and the time that it was valid. This state is constructed in one of two ways. If the valid-time state current at the time of the insertion contains a value-equivalent tuple, the timestamps of that tuple are augmented to reflect the new information. Otherwise a new tuple is inserted. In both cases, the updated valid-time state is inserted into r indexed by the current transaction time, c_t. We assume the function `rollback` returns the valid time state current during the argument transaction time.

```
insert(r, (a_1, ..., a_n), t_v):
    vr ← rollback(r, c_t);
    found ← FALSE;
    for each x ∈ vr
        if x[A] = (a_1, ..., a_n)
            for i ← 1 to n
                x[T_i] ← x[T_i] ∪ t_v;
            found ← TRUE;
    if not found
        vr ← vr ∪ (a_1 t_v, ..., a_n t_v);
    r ← r ∪ {(c_t, vr)};
    return r;
```

```
delete(r, (a_1, ..., a_n)):
    vr ← rollback(r, c_t);
    for each x ∈ vr
        if x[A] = (a_1, ..., a_n)
            t ← x[T_1] ∩ ... ∩ x[T_n];
            if t ≠ ∅
                for i ← 1 to n
                    x[T_i] ← x[T_i] − t;
                if x[T_1] = ∅ and ... and x[T_n] = ∅
                    vr ← vr − {x};
                r ← r ∪ {(c_t, vr)};
    return r;
```

Deletion of a fact involves the removal of the fact from the current valid-time state if it exists, and no action otherwise. A fact to be deleted is present in a tuple x, if the explicit attribute values of x match $(a_1, \ldots, a_n)$ and the intersection of the valid-time elements associated with the attribute values of x is non-empty. We delete from each timestamp the computed intersection, and remove the entire tuple if all resulting timestamps are empty.

3.5 Ben-Zvi's Tuple Timestamped Representation Scheme

Like the representational model in Section 3.1, Ben-Zvi's data model is a 1NF tuple-timestamping model. Appended to each tuple are five timestamp attributes [BZ82].

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 $\mathcal{R}$ is represented by a relation schema R in Ben-Zvi's data model as follows.

$$R \quad = \quad (A_1, \ldots, A_n, \mathrm{T}_{es}, \mathrm{T}_{rs}, \mathrm{T}_{ee}, \mathrm{T}_{re}, \mathrm{T}_d)$$

In a tuple, the value of attribute T_{es} (*effective start*) is the time when the explicit attribute values of the tuple start being true. The value for T_{rs} indicates when the T_{es} value was stored. Similarly, the value for T_{ee} (*effective end*) indicates when the information recorded by the tuple ceased to be true, and T_{re} contains the time when the T_{ee} value was recorded. The last implicit attribute T_d indicates the time when the information in the tuple was logically deleted from the database.

It is not necessary that T_{ee} be recorded when the T_{es} value is recorded (i.e., when a tuple is inserted). The symbol '–' indicates an unrecorded T_{ee} value (and T_{re} value). Similarly, the symbol '–', when used in the T_d field, indicates that a tuple contains current information.

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

Emp	*Dept*	T_{es}	T_{rs}	T_{ee}	T_{re}	T_d
Jake	Ship	6/10	6/5	6/15	6/5	6/10
Jake	Ship	6/5	6/10	6/20	6/10	6/15
Jake	Ship	6/10	6/15	6/15	6/15	6/20
Jake	Load	6/10	6/20	6/15	6/20	–
Kate	Ship	6/25	6/20	6/30	6/20	–

In the example, the timestamps T_{es} and T_{ee} are stored simultaneously, hence the registration timestamps associated with the effective timestamps are identical within each tuple. As facts are corrected, the deletion timestamp T_d is set to the current transaction time, effectively outdating the given fact, and a new tuple without a deletion time is inserted. As only two facts are current when all updates have been performed on the database, only two tuples with no deletion times remain. □

In the conversion functions presented next, 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 symbol '–' if *UC* is encountered as a first component; otherwise, it returns a maximum first component. The function *max_2* selects a maximum second component. The function *bi_chr* may accept the symbol '–' as a transaction-time end value, in which case the symbol is treated as the current time. Bitemporal chronons with *UC* as first component are then generated. When '–' is encountered as a valid-time end, it is treated as the maximum valid-time value, c^{∞}_{vt}. Analogously, when '–' is encountered as a transaction-time value, it is treated as the current transaction time, c_t, as well as the value *UC*.

The first conversion function is very similar to the corresponding function in Section 3.1. The routine `conceptual_to_snap2` constructs an output tuple for each rectangle in a covering of a bitemporal element. The effective-start and effective-end timestamps are set to the minimum and maximum valid-time chronons in the rectangle, respectively. We set the times when the valid timestamps were stored to the minimal transaction time chronon in the rectangle. The deletion time of the tuple is set to the maximal transaction time of the rectangle (possibly *UC*), thereby denoting when the fact was last current in the relation.

```
conceptual_to_snap2(r', cover):
    s ← ∅;
    for each x ∈ r'
        z[A] ← x[A];
        for each t ∈ cover(x[T])
            z[T_rs] ← min_1(t);
            z[T_re] ← z[T_rs];
            z[T_d] ← max_1(t);
            z[T_es] ←min_2(t);
            z[T_ee] ← max_2(t);
            s ← s ∪ {z};
    return s;
```

```
snap2_to_conceptual(r):
    s ← ∅;
    for each z ∈ r
        r ← r − {z};
        x[A] ← z[A];
        x[T] ← make_ts(z[T_es], z[T_rs], z[T_ee], z[T_re], z[T_d]);
        s ← s ∪ {x};
    return coalesce(s);
```

The function `snap2_to_conceptual` performs the inverse transformation. It constructs one conceptual tuple for each set of value-equivalent tuples in the representation. Initially, each representational tuple is examined, and a conceptual tuple corresponding to that representational tuple is generated.

The function `make_ts` constructs a bitemporal element from the five timestamps in the representational tuple. There are three cases to consider. In each case, we construct a bitemporal element representing a rectangle or union of rectangles bounded by the argument time values.

First, if the effective-time start and effective-time end values were stored simultaneously, the associated element corresponds to a rectangular region bounded in valid time and possibly unbounded in transaction time. Similarly, if the values were not stored simultaneously, it may be the case that the effective-end time was never stored. This corresponds to a rectangular region that is unbounded in valid time and possibly bounded in transaction time, depending on if the tuple has been deleted.

Otherwise, both the effective-time start and the effective-time end values have been stored, and are unequal. The resulting region is unbounded in valid time between the times when the effective-time start and effective-time end were stored, and possibly bounded in transaction time, depending on if the tuple has been deleted.

Finally, function `coalesce` collapses each set of value-equivalent tuples in the result into a single tuple.

```
make_ts(t_es, t_rs, t_ee, t_re, t_d):
    if t_rs = t_re
        t ← bi_chr({t_rs..t_d}, {t_es..t_ee});
    else
        if t_re = '–'
            t ← bi_chr({t_rs..t_d}, {t_es..c^∞_vt});
        else
            t ← bi_chr({t_rs..t_re}, {t_es..c^∞_vt}) ∪
                        bi_chr({t_re..t_d}, {t_es..t_ee});
    return t;
```

```
coalesce(r):
    r' ← ∅;
    for each x ∈ r
        r ← r − {x};
        for each y ∈ r
            if x[A] = y[A]
                x[T] ← x[T] ∪ y[T];
                r ← r − {y};
        r' ← r' ∪ {x};
    return r';
```

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

```
insert(r, (a_1, ..., a_n), t_v, cover_v):
    for each t ∈ cover_v(t_v)
        for each x ∈ r
            if x[A] = (a_1, ..., a_n) and x[T_d] = '–' and
                    x[T_es, T_ee] ∩ t ≠ ∅
                r ← r − {x};
                x[T_d] ← c_t;
                z[A] ← x[A];
                z[T_es] ← min(x[T_es] ∪ t);
                z[T_ee] ← min(x[T_ee] ∪ t);
                z[T_d] ← '–';
                r ← r ∪ {x, z};
    return r;
```

```
delete(r, (a_1, ..., a_n)):
    for each x ∈ r
        if x[A] = (a_1, ..., a_n) and
                x[T_d] = '–'
            x[T_d] ← c_t;
    return r;
```

3.6 Covering Functions

In Sections 3.1 to 3.5, we used covering functions when using sets of rectangles to represent bitemporal elements of conceptual tuples. 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 examples 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.

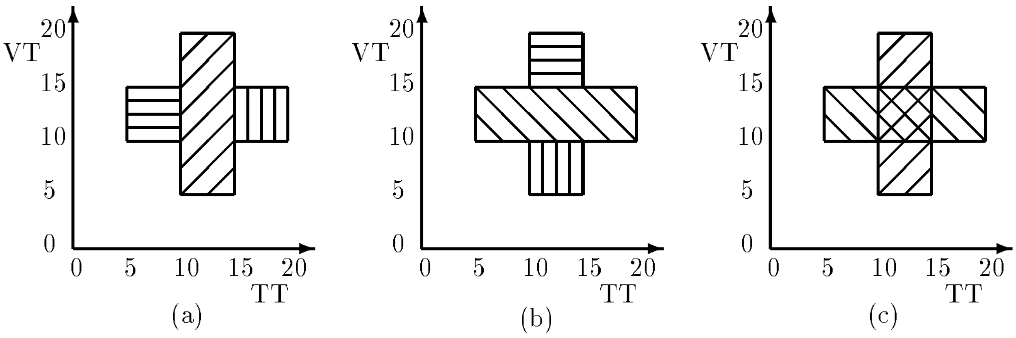

Figure 2: Example Coverings of a Bitemporal Element

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 last 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, as many more tuples may now be affected by a single update.

3.7 Summary

We introduced five representations of bitemporal relations and showed how instances in the bitemporal conceptual data model (BCDM) can be mapped to instances in each of these representations. In addition, we devised reverse mapping functions, from representational instances to BCDM instances. The established correspondence between representations and the conceptual model is central to our work—the BCDM forms a unifying link between disparate bitemporal models. The mapping functions assign semantics to instances in the five representations and allows us to meaningfully compare instances of diverse models.

In the next section, we discuss in more detail the role of the BCDM with respect to data model unification. Subsequent sections provide a detailed examination of the concept of equivalence among the data models.

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 nature of both the enterprise being modeled and the database, and hence incorporated support for both valid and transaction time. They attempted to retain the simplicity of the relational model; the two tuple-timestamping models were perhaps most successful in this regard. They attempted to present all the information concerning an object in one tuple; the attribute-value timestamped models were perhaps best at that. And they attempted to ensure ease of implementation and query evaluation efficiency; the backlog representation may have advantages here.

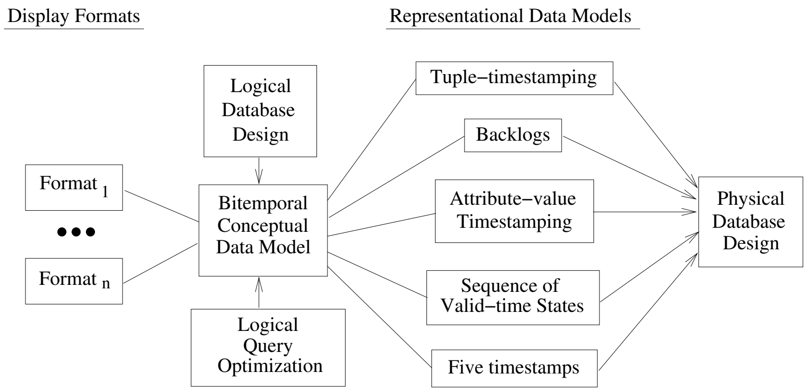

Figure 3: Interaction of Conceptual and Representational Data Models

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 presentation and implementation considerations. We feel that the bitemporal conceptual data model proposed in this paper is a more appropriate basis for expressing this semantics. This data model is notable 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 other 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 places the bitemporal conceptual data model with respect to the tasks of logical and physical database design, storage representation, query optimization, and display. It indicates that logical database design produces the conceptual relation schemas, which are then refined into relation schemas in some representational data model(s). The query language itself would be based on the conceptual data model. 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 five different representations of the example conceptual relation introduced in Section 2.1. Each of these may be an appropriate presentation under some circumstances, independent of how the relation is stored. For example, 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 semantically equivalent representations of the same conceptual relation may co-exist. In this and the next section, we explore the nature of this relationship between the conceptual data model and the representational data models. We focus next on the equivalence among the objects in the models; a following section will examine equivalence when operations on these objects is also considered.

5.1 Snapshot Equivalence

We use snapshot equivalence to formalize the notion of relation instances having the same information content.

Snapshot equivalence makes use of transaction and valid timeslice operators. We initially define these operators for conceptual relations, then for relations in each of the representational models.

The *transaction-timeslice* 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 ρ^{B}, we describe its operation on a bitemporal conceptual 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, along with each of the valid-time chronons paired to a matching transaction time, become a tuple in the result. The transaction-timeslice operator may also be applied to a transaction-time relation, in which case the result is a snapshot relation.

The *valid 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 valid time and produces a transaction-time relation. The valid-timeslice operator may also be applied to a valid-time relation, in which case the result is a snapshot relation.

Definition: Define a relation schema $R = (A_1, \ldots, A_n | \mathrm{T})$, and let r be an instance of this schema. Let t_2 denote an arbitrary time value and let t_1 denote a time not exceeding the current time. Then the transaction-timeslice and valid-timeslice operators may be defined as follows for the conceptual data model.

$$\begin{aligned}
\rho^{\mathrm{B}}_{t_1}(r) &= \{z^{(n+1)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{T}_v] = \{t_2 \mid (t_1, t_2) \in x[\mathrm{T}]\} \wedge z[\mathrm{T}_v] \neq \emptyset)\} \\
\tau^{\mathrm{B}}_{t_2}(r) &= \{z^{(n+1)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{T}_t] = \{t_1 \mid (t_1, t_2) \in x[\mathrm{T}]\} \wedge z[\mathrm{T}_t] \neq \emptyset)\}
\end{aligned}$$

□

The transaction-timeslice operator for transaction-time relations (ρ^{T}) and the valid-timeslice operator for valid-time relations (τ^{V}) are straightforward special cases.

We can now formally define snapshot equivalence so that it applies to each representational data model for which the valid-timeslice and transaction-timeslice operators have been defined.

Definition: Two relation instances, r and s, are *snapshot equivalent*, $r \overset{\mathrm{S}}{\equiv} s$, if for all times t_1 not exceeding the current time and all times t_2,

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

□

The concept of snapshot equivalence is due to Gadia and was first defined for valid-time relations [Gad86] and was later generalized to multiple dimensions [GY88]. We have chosen to avoid the original term *weakly equivalent* to avoid confusion with the different notion of *weak* *equivalence* over algebraic expressions (e.g., [Ull82]). Disambiguating the original term by prefixing with "temporally" is awkward. In the next section, we will discuss how snapshot equivalence may also be applied to pairs of instances when the instances belong to diferent models.

The following theorem states that identity and snapshot equivalence coincide for the conceptual model. It is a major source of semantic clarity that two instances have the same information content exactly when they are identical.

THEOREM **1** Let r and s be conceptual relations over the same schema. Then $r \overset{S}{\equiv} s$ if and only if $r = s$.

PROOF: First assume that $r \overset{S}{\equiv} s$. We show that for each $x \in r$, $x = (a_1, \ldots, a_n \mid t_x)$ there exists a $y \in s$, $y = (a_1, \ldots, a_n \mid t_y)$, with $t_x = t_y$.

By the definition of snapshot equivalence there exist tuples y_i, $i = 1, \ldots, m$, in s so that for all t_1, t_2, where t_1 does not exceed the current time, $\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(\{x\})) = \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(\{y_1, \ldots, y_m\}))$. The definitions of the involved operators demand that each of the y_i must have $a_1, \ldots, a_n$ as explicit attribute values. Further, the operators demand that $t_x = \cup_i \, t_{y_i}$. By definition of the BCDM, no two tuples with the same explicit attribute values may exist in an instance. Thus, $i = 1$ and $y_1 = y$, proving the claim. As a result, each tuple in r has an exact match in s. By the symmetrical argument, each tuple in s has a match in r, and the two instances are consequently identical.

In the other direction, assuming that $r = s$, clearly $\forall t_1$, t_2 where t_1 does not exceed the current time, $\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r)) = \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(s))$. □

5.2 Rollback and Timeslice Operators

We now define the timeslice operators for each of the five representational models. These definitions extend the notion of snapshot equivalence to the corresponding representation. In the definitions, let t denote an arbitrary time value and let t' be a time value not exceeding the current time.

DEFINITION: (Snodgrass' 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.

$$\begin{aligned} \rho^{\mathrm{B}}_{t'}(r) &= \{z^{(n+2)} \mid \exists x \in r \; (z[A] = x[A] \wedge z[\mathrm{V}] = x[\mathrm{V}] \wedge t' \in x[\mathrm{T}])\} \\ \tau^{\mathrm{B}}_{t}(r) &= \{z^{(n+2)} \mid \exists x \in r \; (z[A] = x[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \wedge t \in x[\mathrm{V}])\} \end{aligned}$$

□

DEFINITION: (Jensen's 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.

$$\begin{aligned} \rho^{\mathrm{B}}_{t'}(r) = \{z^{(n+2)} \mid \exists x \in r \; (&z[A] = x[A] \wedge z[\mathrm{V}] = x[\mathrm{V}] \wedge x[\mathrm{T}] \le t' \wedge x[\mathrm{Op}] = I \wedge \\ &(\neg \exists y \in r \; (y[A] = x[A] \wedge y[\mathrm{V}] = x[\mathrm{V}] \wedge y[\mathrm{Op}] = D \wedge x[\mathrm{T}] \le y[\mathrm{T}] \le t')))\} \\ \tau^{\mathrm{B}}_{t}(r) = \{z^{(n+2)} \mid \exists x \in r \; (&z[A] = x[A] \wedge z[\mathrm{T}] = x[\mathrm{T}] \wedge z[\mathrm{Op}] = x[\mathrm{Op}] \wedge t \in x[\mathrm{V}])\} \end{aligned}$$

In the definition of transaction timeslice, 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 underscores the emphasis accorded transaction time in this model. □

DEFINITION: (Gadia's Attribute Value Timestamped Data Model).

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

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

DEFINITION: (McKenzie's Attribute Value Timestamped Data Model). Define a relation schema $R = (\mathrm{T}, \mathrm{VR})$, with T being a transaction timestamp and $\mathrm{VR} = (A_1\mathrm{V}_1, \ldots, A_n\mathrm{V}_n)$, where the A_i, $1 \leq i \leq n$, are explicit attributes and the corresponding V_i are valid-time elements. An instance of this schema is a sequence of valid-time states indexed by transaction times as. Let r be such an instance.

$$\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n)} \mid \exists (t, vr) \in r\ (t' \leq t \wedge \neg\exists (t'', vr'') \in r\ (t' \leq t'' < t) \wedge z \in vr)\}$$
$$\tau^{\mathrm{B}}_{t}(r) = \{(t'', S) \mid \forall s \in S\ (\exists t''\ ((t'', vr) \in r \wedge \forall x \in vr\ (\forall i\, 1 \leq i \leq n\ ((t \in x[\mathrm{V}_i] \Rightarrow s[A_i] = x[A_i]) \wedge \\ (t \notin x[\mathrm{V}_i] \Rightarrow s[A_i] = \perp)) \wedge \exists i\, 1 \leq i \leq n\ (t \in x[\mathrm{V}_i])))))\}$$

The first operator extracts the valid time relation with the greatest transaction timestamp before t'. The second returns a rollback relation, a sequence of snapshot states such that each tuple in each snapshot state was valid at valid time t for all attributes. Some, but not all, attribute values in the tuples in the snapshot states may be null values. □

DEFINITION: (Ben-Zvi's Tuple Timestamped Data Model). Define a relation schema $R = (A_1, \ldots, A_n, \mathrm{T}_{es}, \mathrm{T}_{ee}, \mathrm{T}_{rs}, \mathrm{T}_{re}, \mathrm{T}_d)$, and let r be an instance of this schema.

$$\rho^{\mathrm{B}}_{t'}(r) = \{z^{(n+2)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{T}_{es}] = x[\mathrm{T}_{es}] \wedge x[\mathrm{T}_{rs}] \leq t' \wedge (x[\mathrm{T}_d] \neq \text{`}-\text{'} \Rightarrow t' \leq x[\mathrm{T}_d]) \wedge \\ ((x[\mathrm{T}_{re}] \neq \text{`}-\text{'} \Rightarrow t' \leq x[\mathrm{T}_{re}]) \Rightarrow z[\mathrm{T}_{ee}] = \text{`}-\text{'}) \wedge \\ ((x[\mathrm{T}_{ee}] \neq \text{`}-\text{'} \wedge x[\mathrm{T}_{re}] \leq t') \Rightarrow z[\mathrm{T}_{ee}] = x[\mathrm{T}_{ee}])\}$$
$$\tau^{\mathrm{B}}_{t}(r) = \{z^{(n+2)} \mid \exists x \in r\ (z[A] = x[A] \wedge z[\mathrm{T}_{rs}] = x[\mathrm{T}_{rs}] \wedge (\\ (((x[\mathrm{T}_{es}] \leq t) \wedge (x[\mathrm{T}_{ee}] \neq \text{`}-\text{'} \Rightarrow t \leq x[\mathrm{T}_{ee}])) \Rightarrow z[\mathrm{T}_{re}] = x[\mathrm{T}_d]) \vee \\ ((x[\mathrm{T}_{ee}] \neq \text{`}-\text{'} \wedge t \geq x[\mathrm{T}_{ee}] \wedge x[\mathrm{T}_{rs}] \neq x[\mathrm{T}_{re}]) \Rightarrow z[\mathrm{T}_{re}] = x[\mathrm{T}_{re}])))\}$$

In the first operator, the complexity arises in computing T_{ee} for the resulting tuples; the other implicit attribute, T_{es}, is trivial. Two possibilities for T_{ee} exist, '–' and $x[\mathrm{T}_{ee}]$, depending on the value of $x[\mathrm{T}_{re}]$. For the second operator, the complexity is in determining $z[\mathrm{T}_{re}]$, which can also assume two possible values, $x[\mathrm{T}_d]$ and $x[\mathrm{T}_{re}]$, depending primarily on the value of $x[\mathrm{T}_{ee}]$. □

For each of the five schemes, the transaction-timeslice operator for transaction-time relations (ρ^{T}) and the valid-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}.

The existence of the timeslice operators for the representational models has important implications, as we discuss in the following. Rather than providing theorems and proofs for each representational model, the theorems and proofs in the remainder of this section are limited to a single model only. Specifically, the model introduced in Section 3.1 is used due to its straightforward structure. Corresponding results hold for the remaining models; proofs may be similarly obtained.

There is no reason to apply ρ before τ in the definition of snapshot equivalence, as the following theorem states.

THEOREM **2** Let r be a temporal relation. Then for all times t_1 not exceeding the current time and for all times t_2,

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

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. □

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

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, $r \overset{\mathrm{S}}{\subseteq} s$, if for all times t_1 not exceeding UC and all times t_2,

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

More generally, a temporal query expression Q_1 is a *snapshot subset* of a temporal query expression Q_2, $Q_1 \overset{\mathrm{S}}{\subseteq} Q_2$, if all instantiations of Q_1 are snapshot subsets of the corresponding instantiations of Q_2. □

THEOREM **3** 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 $r_1 \overset{\mathrm{S}}{\equiv} r_2$.
2. If $s_1 \overset{\mathrm{S}}{\equiv} s_2$ 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 for all times t_1 and t_2 that if $x \in \tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_1))$ then also $x \in \tau^{\mathrm{V}}_{t_2}\rho^{\mathrm{B}}_{t_1}(r_2))$. Let tuple x be in $\tau^{\mathrm{V}}_{t_2}(\rho^{\mathrm{B}}_{t_1}(r_1))$. By the definitions of transaction and valid 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 $x'[\mathrm{T}]$. Independently of a particular covering function, 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_1 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. □

This theorem has important consequences. For each representation and for any covering function, snapshot equivalence partitions the relation instances into equivalence classes where each instance in an equivalence class maps to the same bitemporal conceptual relation instance. The semantics of the representational instance is thus identical to that of the corresponding conceptual instance. This correspondence provides a way of converting instances between representations: the conversion proceeds through a snapshot equivalent 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 3.5 are semantically equivalent to the bitemporal conceptual relation given in Section 2.1, and are thus semantically equivalent to each other.

6 Algebras and Equivalence

We now examine 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.

In Section 5.1, we defined two algebraic operators, the transaction-and valid-timeslice operators, on conceptual relations. We now define the remaining conceptual algebraic operators. Then the corresponding operations on the chosen tuple-timestamped representation (see Section 3.1) were defined. Each of the remaining four representations could have been used instead. 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 An Algebra for Bitemporal Conceptual Relations

Define a relation schema $R = (A_1, \ldots, A_n | \mathrm{T})$, and let r be an instance of this schema. Let t_2 denote an arbitrary time value and let t_1 denote a time not exceeding the current time.

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.

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

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.

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

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

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

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.

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

The last two lines compute the bitemporal element, depending on whether a value-equivalent tuple may be found in r_2.

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.

$$
\begin{aligned}
R &= (A_1, \ldots, A_n, B_1, \ldots, B_l \mid \mathrm{T}) \\
S &= (A_1, \ldots, A_n, C_1, \ldots, C_m \mid \mathrm{T})
\end{aligned}
$$

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 result is the intersection of the timestamps of the two tuples that produced it.

$$
\begin{aligned}
r \bowtie^{\mathrm{B}} s = \{z^{(n+l+m+1)} \mid \exists x \in r\ \exists y \in s\ (&x[A] = y[A] \wedge x[\mathrm{T}] \cap y[\mathrm{T}] \neq \emptyset \wedge \\
& z[A] = x[A] \wedge z[B] = x[B] \wedge z[C] = y[C] \wedge \\
& z[\mathrm{T}] = x[\mathrm{T}] \cap y[\mathrm{T}])\}
\end{aligned}
$$

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 An Algebra for Snodgrass' Tuple Timestamped Representation Scheme

For each of the algebraic operators defined in the previous section, we now define counterparts for the first of the five 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 $A_1, \ldots, A_n$, $\mathrm{T_s}$, $\mathrm{T_e}$, $\mathrm{V_s}$, and $\mathrm{V_e}$.

We define in turn projection, selection, union, difference, and natural join. The timeslice operators were defined in Section 5.2.

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

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

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

$$
\sigma_P^{\mathrm{B}}(r) = \{z^{(n+4)} \mid z \in r \wedge P(z[A]))\}
$$

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

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

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

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

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.

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

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.

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

As for the previous model, corresponding operators for valid-time and transaction-time relations may be defined as special cases of the operators already defined.

6.3 Equivalence Properties

We have seen that a bitemporal conceptual 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,

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

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 the framework presented here, only preservation of snapshot equivalence is required. 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.

We proceed by showing that 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 is correct, irrespective of covering.

Theorem 4 The algebraic operators preserve snapshot equivalence. Specifically, let $r \stackrel{\mathrm{S}}{\equiv} r'$ and $s \stackrel{\mathrm{S}}{\equiv} s'$. Then

$$
\begin{aligned}
r \bowtie^{\mathrm{V}} s &\stackrel{\mathrm{S}}{\equiv} r' \bowtie^{\mathrm{V}} s' \\
r \bowtie^{\mathrm{B}} s &\stackrel{\mathrm{S}}{\equiv} r' \bowtie^{\mathrm{B}} s'
\end{aligned}
$$

$$\begin{array}{rcl} \sigma_P^{\mathrm{B}}(r) & \stackrel{\mathrm{S}}{\equiv} & \sigma_P^{\mathrm{B}}(r') \\ \pi_D^{\mathrm{B}}(r) & \stackrel{\mathrm{S}}{\equiv} & \pi_D^{\mathrm{B}}(r') \\ r \cup^{\mathrm{B}} s & \stackrel{\mathrm{S}}{\equiv} & r' \cup^{\mathrm{B}} s' \\ r -^{\mathrm{B}} s & \stackrel{\mathrm{S}}{\equiv} & r' -^{\mathrm{B}} s'. \end{array}$$

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 $\stackrel{\mathrm{S}}{\equiv}$, for each $x_i \in r$ the exists a set of tuples $x_k^i \in r'$ with $x_k^i[AB] = x_i[AB]$ and so that $\cup_k x_k^i[\mathrm{V}] \supseteq x_i[\mathrm{V}]$. The set x_k^i covers x_i. For each j a similar set x_l^j exists that covers x_j. Applying $\bowtie^{\mathrm{V}}$ to the sets of tuples $x_k^i \in r'$ and $x_l^j \in s'$ yields a set of tuples x_m with $x_m[ABC] = x[ABC]$ 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. □

The next step is to combine the transformation functions with the representation level operators to create corresponding conceptual-level operators. Given a representation level operator, α, its corresponding conceptual-level operators, α^c, is defined as follows.

$$\alpha_X^c(r') = \texttt{snap\_to\_conceptual}(\alpha_X(\texttt{conceptual\_to\_snap}(r')))$$

Theorems 3 and 4 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, bitemporal conceptual 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 bitemporal conceptual 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. This is formalized in the following theorem.

THEOREM **5** The induced algebraic operators preserve snapshot equivalence.

PROOF: Let α_X^c be an induced conceptual operator, and suppose that conceptual relations r and s are snapshot equivalent. By Theorem 1, $r = s$, and therefore, $\texttt{conceptual\_to\_snap}(r) \stackrel{\mathrm{S}}{\equiv} \texttt{conceptual\_to\_snap}(s)$. By Theorem 4, $\alpha_X(\texttt{conceptual\_to\_snap}(r)) \stackrel{\mathrm{S}}{\equiv} \alpha_X(\texttt{conceptual\_to\_snap}(s))$. Finally, by Theorem 3, $\texttt{snap\_to\_conceptual}(\alpha_X(\texttt{conceptual\_to\_snap}(r))) \stackrel{\mathrm{S}}{\equiv} \texttt{snap\_to\_conceptual}(\alpha_X(\texttt{conceptual\_to\_snap}(s)))$. □

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.

Transaction-time relations   Snapshot relations

$$\begin{tikzcd}
r,\ r' \arrow[r, "\rho_t^{\mathrm{T}}"] \arrow[d, "\bowtie^{\mathrm{T}}"'] & \rho_t^{\mathrm{T}}(r),\ \rho_t^{\mathrm{T}}(r') \arrow[d, "\bowtie^{\mathrm{S}}"] \\
r \bowtie^{\mathrm{T}} r' \arrow[r, "\rho_t^{\mathrm{T}}"] & \rho_t^{\mathrm{T}}(r \bowtie^{\mathrm{T}} r') \stackrel{\mathrm{S}}{\equiv} \rho_t^{\mathrm{T}}(r) \bowtie^{\mathrm{S}} \rho_t^{\mathrm{T}}(r')
\end{tikzcd}$$

Figure 4: Reducibility of Transaction-time Natural Join to Snapshot Outer Natural Join.

THEOREM **6** Let t denote an arbitrary time that, when used with a rollback operator, does not exceed the current time. In each equivalence, let r and s be relation instances of the proper types for the given operators. Then the following hold.

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

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 $x_{12}'' \stackrel{\mathrm{S}}{\equiv} x''$ (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, $x_{12}'' \stackrel{\mathrm{S}}{\equiv} x''$. $\square$

6.4 Covering Transformations

When a bitemporal conceptual relation is mapped to a representation scheme, a covering function is employed to represent bitemporal elements by sets of rectangles. The mappings were used in Sections 3.1 to 3.5, and different types of covering functions were discussed in Section 3.6. 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 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. □

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 by $x = (a_1, \ldots, a_n, t)$ and $x' = (a'_1, \ldots, a'_n, t')$ where $\exists k_1 \ldots k_m \, (a_{k_1} = \perp \neq a'_{k_1} \wedge \ldots \wedge a_{k_m} = \perp \neq a'_{k_m})$ and $\forall i \notin \{k_1, \ldots, k_m\} (a_i = a'_i)$. The last elements, t and t', of the two tuples denote bitemporal elements. 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. □

Theorem 7 Coalescing preserves snapshot equivalence.

Proof: Let r be a relation instance containing x and x' as given 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. □

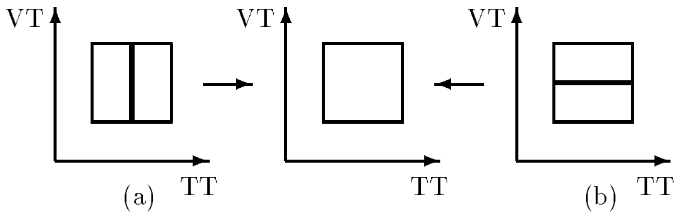

Figure 5: Coalescing

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, possibly 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.

$$\begin{array}{llll}
1 & s = & \{z^{(n+4)} \mid z[A] = x[A] \wedge ((z[\mathrm{T}] \in cover_t^{max}(x[\mathrm{T}] - x'[\mathrm{T}]) \wedge z[\mathrm{V}] = x[\mathrm{V}]) \vee \\
2 & & \qquad (z[\mathrm{T}] \in cover_t^{max}(x'[\mathrm{T}] - x[\mathrm{T}]) \wedge z[\mathrm{V}] = x'[\mathrm{V}]) \vee \\
3 & & \qquad (z[\mathrm{T}] = x[\mathrm{T}] \cap x'[\mathrm{T}] \wedge z[\mathrm{V}] = x[\mathrm{V}] \cup x'[\mathrm{V}]))\}
\end{array}$$

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

Theorem 8 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. □

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.

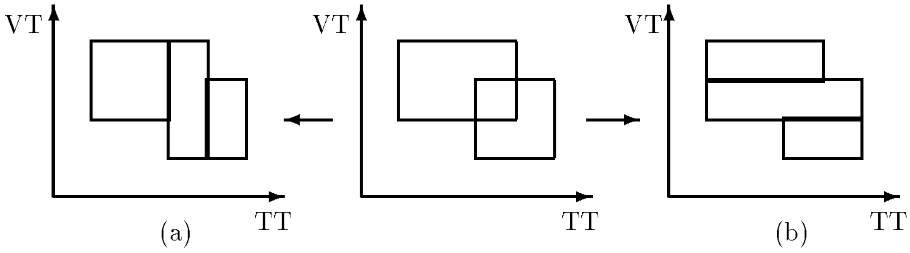

Figure 6: Eliminating Representational Repetition of Information

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 the *bitemporal conceptual data model* which 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 five existing bitemporal *representational data models*: a first normal form (1NF) tuple-timestamped data model, a data model based on 1NF timestamped change requests recorded in backlog relations, a non-1NF data model in which attribute values were stamped with rectangles in transaction-time/valid-time space, a non-1NF model where valid-time states are indexed by transaction time, and a 1NF model where each tuple is accorded five timestamp values. We also showed how extensions to the conventional relational algebraic operators could be defined in a representational data model and then be meaningfully mapped to analogous operators in the conceptual data model.

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 six models. It was demonstrated that equivalent algebras of snapshot perserving operators could be defined for different models. Further, the operators were shown to be 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 that alter coverings of bitemporal relation representations.

We advocate a separation of concerns. Each of 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. We are currently investigating this topic [JSS92]. Also, more work is needed in mapping existing temporal query language proposals into the conceptual data model.

The results in this paper employ snapshot equivalence to formalize the notion of relation instances having the same information content. Other equivalences, such as *strong equivalence* [CCT93], capture a different notion of identical information content. It would be illuminating to attempt to demonstrate mappings among the existing temporal models, to define equivalent algebras, and to define normal forms, using these other equivalences.

Acknowledgements

Part of this research was conducted while the first author visited the University of Arizona. Support was provided by the Danish Natural Science Research Council through grants 11–9675–1 SE and 11–0061, the National Science Foundation through grants IRI–8902707 and IRI–9302244, and the IBM Corporation through Contract #1124.

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.

[CCT93] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. Technical report STERN IS-93-8, New York University Stern School of Business, 1993. (to appear in *TODS*).

[Gad86] S. K. Gadia. Weak Temporal Relations. In *Proceedings of ACM PODS*, pages 70–77, 1986.

[GY88] S. K. Gadia and C. S. Yeung. A Generalized Model for a Relational Temporal Database. In *Proceedings of ACM SIGMOD*, pages 251–2259, 1988.

[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*, 2(1):75–111, 1992.

[JCG*92] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, and R. T. Snodgrass. A Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 21(3):35–43, September 1992.

[JS92] C. S. Jensen and R. T. Snodgrass. Proposal of a Data Model for the Temporal Structured Query Language. TempIS Technical Report 37, Department of Computer Science, University of Arizona, Tucson, AZ, July 1992.

[JSS92] C. S. Jensen, R. Snodgrass, and M. D. Soo. Extending Normal Forms to Temporal Relations. Technical Report TR-92-17, Department of Computer Science, University of Arizona, Tucson, AZ, July 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.

[McK88] E. McKenzie. *An Algebraic Language for Query and Update of Temporal Databases.* PhD thesis, Computer Science Department, University of North Carolina, 1988.

[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.

[Sno92] R. T. Snodgrass. Temporal Databases, in *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space,* Springer-Verlag, LNCS 639, pages 22–64, 1992.

[Sno93] R. Snodgrass. An Overview of TQuel, Chapter 6 of Temporal Databases: Theory, Design, and Implementation, Benjamin/Cummings Pub. Co., pages 141–182, 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.