Temporal Interoperability in Multi+Temporal Databases

Fabio Grandi

Dipartimento di Elettronica, Informatica e Sistemistica

Università degli Studi di Bologna, Italy

Fax: +39 (0)51 644.3540

EMail: fgrandi@deis.unibo.it

Abstract

A multidatabase system allows applications and users to access heterogeneous information through a common interface. Information is shared and exchanged in a transparent way within a federation of autonomous database systems. In a **multi+temporal database**, the connected systems may be standard (snapshot) databases, temporal databases (either valid-time, transaction-time or bitemporal) or even single multitemporal systems, in which data with different temporal formats may coexist.

In this paper we consider semantic interoperability problems arising from the interaction of temporally heterogeneous data in a multi+temporal environment. In particular, we focus on the evaluation of multitemporal algebraic expressions for the support of an SQL-like query interface. The basic requirement of a multitemporal algebra is that all the operations involved must be carefully designed in order to retrieve the largest amount of certain stored information and avoid the creation of spurious information during the interaction of heterogeneous data. The main difficulties underlying the definition of such operations are presented in the paper, together with solutions to overcome them. They include conversion of data from one temporal format into another, definition of lossless operations on temporally incomplete information, and semantic optimization of multitemporal algebraic expressions.

Keywords: *Multidatabase, Temporal Database, Relational Database, Semantic Interoperability*

1 Introduction

The interaction of different database systems [5, 17] is one of the requirements of today's advanced applications. The purpose of dealing with semantic heterogeneity is to overcome the problems connected with the management of possibly related data in a federation of systems. A multidatabase system allows applications and users to access heterogeneous information through a common interface, in a transparent way. In this work, we only consider *relational* databases; this assumption is not overly restrictive, since different database systems often provide a relational interface towards the external world [13]. Hence, the user/application interface provided by a multidatabase system can be based on the standard SQL relational data definition and manipulation language [22].

Another important issue recognized by database research in recent years is the representation of time and the capability of managing versioned data [12, 21, 23]. This requirement arises from many application areas, such as CAD/CAM/CIM, management of legal and medical records, financial information systems, and from the re-engineering of legacy applications. Two orthogonal time dimensions are usually considered in the current literature [10, 23]: *transaction time*, which indicates when an event is recorded in a database, and *valid time*, which represents when an event occurs, occurred or is expected to occur in the real world. According to this taxonomy, *snapshot* databases without time support, *transaction-* or *valid-time* databases supporting one time dimension (monotemporal) or *bitemporal* databases supporting both time dimensions can be defined [10].

The problem of temporal semantic interoperability dealt with in this paper consists of the sharing and exchange of information between relational databases of different temporal types or even between relations of different temporal types within the same database system. Other kinds of temporal interoperability (e.g. between different time representations [9]) are not taken into account for simplicity. However, different scenarios can be imagined in this context, as sketched in the following.

In a **temporal multidatabase**, different database sites or autonomous systems can hold data based on different temporal models. For instance, most of the existing databases are snapshot but should be made operative in a distributed temporal environment, as required in many application areas.

In a **multitemporal database**, a single system may allow the coexistence of relations with different temporal structures [18, 20]. For instance, *selective versioning* techniques can be used to partition attributes of a relation into groups with different update frequencies. Thus, storing them separately reduces the amount of storage space and can improve the overall database

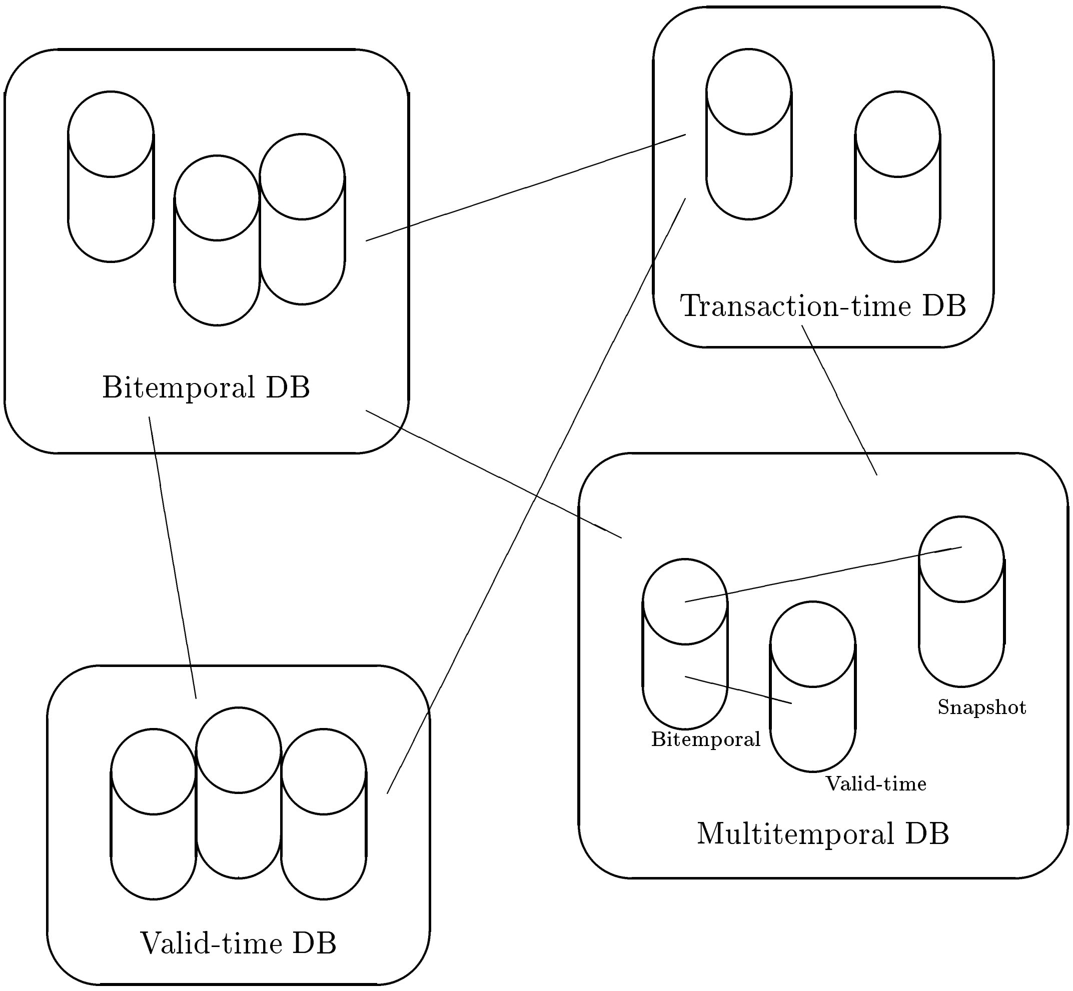

Figure 1: Internal and external interoperability in a multi+temporal database.

efficiency. Different groups can divide constant from time-varying attributes. Hence, owing to data semantics or application aims, one can design some relations as snapshot, some as mono- and some as bi-temporal. It should be noticed that, without selective versioning, constant attribute values would be duplicated in all the versions of a tuple with time-varying attributes. Therefore, selective versioning can be considered as a further normalization step which can be applied to relations during the logical design of a temporal database (e.g. enforcement of *Time Normal Form* [15]). However, the reconstruction of unnormalized data from the resulting relations via join operations requires a sort of *internal* interoperability. It can also be noticed that the TSQL2 query language [19, 20], designed as the standard temporal SQL extension, actually allows the definition of relations with different temporal formats within the same database, although the semantics of "multitemporal" queries has not been formally specified.

In the most general case, we can think of a **multi+temporal database** (see Fig. 1), in which a federation of independent temporal or multitemporal systems is connected to interoperate and provide the functionality of a single multitemporal multidatabase.

The common task of temporal interoperability problems is to provide correct interaction between relations containing information of a different temporal nature. To this purpose, an algebra working on relations of different temporal types must be defined, on which true multitemporal query languages can be based. The main problem connected with the definition of a multitemporal algebra is the transformation of data based on a given temporal format into another. Once all the relations involved in an algebraic expression, corresponding to a given query, are translated into a common temporal format, the expression can be evaluated by means of operators defined to work on data with that format. To this purpose, temporal conversion functions have been introduced in [2] and will be briefly surveyed in Sec. 2.

Since the bitemporal model is the most general representation of data evolving both in the real world and in the database, and the monotemporal and snapshot models can be seen as an *approximate* representation of a bitemporal reality, the common format into which data are to be translated before executing algebraic operations is the bitemporal one. In this way, the largest amount of information contained in the original relations is guaranteed to be preserved. If the result is required in a temporal format differing from the bitemporal one, a final conversion step must be executed.

Additional issues arising from the presence of temporal incompleteness, that is, lack of information concerning given time periods, were also discussed in [2] and will be reviewed in Sec. 3.

The evaluation of algebraic expressions in the bitemporal format, however, is neither convenient nor always necessary to obtain a correct result. This further issue leads to the semantic optimization of algebraic expressions, based on the choice of the most convenient temporal format in which sub-expressions can be evaluated without information loss. A solution of this important problem, which continues and completes the study initiated in [2] for the join operation, is the subject of Sec. 4. Equivalence rules for the temporal reduction of algebraic expressions will be introduced and also illustrated by means of a comprehensive example.

Conclusions can finally be found in Sec. 5.

2 Interoperability of Multitemporal Relational Data

In this section we briefly present the temporal conversion functions. Their formal definitions can be found in [2] for a relational model where tuples are time-stamped with intervals, and in [3] for the BCDM (Bitemporal Conceptual Data Model [11]) where tuples are time-stamped with *temporal elements*. Temporal elements are sets of disjoint maximal intervals in a monotemporal model, and sets of disjoint maximal rectangles in the bitemporal model. Only the basic principles underlying their definitions are reported here, along with some figures illustrating the action of the most complex functions.

The order relation "$\prec$" is defined between different temporal data models as follows:

$X \prec Y$ iff the data model Y includes all the time dimensions of X and not *vice versa*

Only the relationships $S \prec T \prec B$ and $S \prec V \prec B$ hold, where S, T, V, B stand for snapshot, transaction-time, valid-time and bitemporal, respectively. The order relation reflects inclusion of representation abilities provided by the different temporal models.

The notation $M_{XY} : X \to Y$ indicates the generic conversion function. For example, $M_{SV} : S \to V$ represents the function that translates snapshot relations into valid-time ones.

The functions M_{XY} for which $X \prec Y$ are temporal *extensions*, while the functions for which $Y \prec X$ are called *contractions*. The conversion functions which are neither extensions nor contractions (namely M_{TV} and M_{VT}) are called *orthogonal* as valid- and transaction-time are orthogonal dimensions.

The most critical definitions concern *extension* functions, since some kind of temporal information — not contained in the stored data — must be created. Contraction functions require the selection of a part of temporal information — contained in the stored data — to be discarded, but no new information is created. Orthogonal functions can be defined in a consistent way by means of extension and contraction functions.

2.1 Extension Functions

The extension functions are defined on the basis of the following general criteria:

E_1: By definition, snapshot (valid-time) relations only store current data, that is the current (historical) state of the data. Therefore, in an extension along transaction-time, such data can be assigned a $(t_{\text{now}}, t_\infty)$ transaction time-stamp, where t_{now} is the transaction time when the extension function is applied. No extension is however possible along transaction-time beyond t_{now}. As a matter of fact, transaction time is defined by the system and, in any case, the past history of the updates is unknown when an extension is made along transaction time. Any conjecture on past update history is definitely arbitrary and unsafe.

E_2: By definition, the value (last value) stored in a snapshot (transaction-time) relation is considered to be valid in the present but can also be considered valid in the future, since it cannot be forecasted if a modification will occur or not. Therefore, in an extension along valid time, such data can be assigned a $(t_{\text{now}}, t_\infty)$ valid time-stamp, where t_{now} equals the transaction time when the extension function is applied.

E_3: Transaction-time databases are normally used in applications to represent a sort of past history of time-variant data, even if such a history is not explicitly managed by the user (as happens in databases with valid time) and only summarizes the sequence of updates. However, following this natural interpretation, data versioned along transaction-time only, can be extended along valid-time for synchronous values of the two time axes. In other words, the data versioning managed by the system along transaction time can also be reflected on the valid-time axis, reconstructing the view a user had throughout the database life, that is the bitemporal data portion seen by a *diagonal user* [1].

The conversion functions acting on snapshot data (functions M_{ST}, M_{SV}, M_{SB}) are trivial and derive from the direct application of criteria E_1 and E_2.

The conversion from transaction to bitemporal (function M_{BT}, see Fig. 2) is more involved. A detailed discussion on the motivations leading to its definition and alternatives can also be found in [4]. In the figures, the horizontal (vertical) axis represents valid (transaction) time.

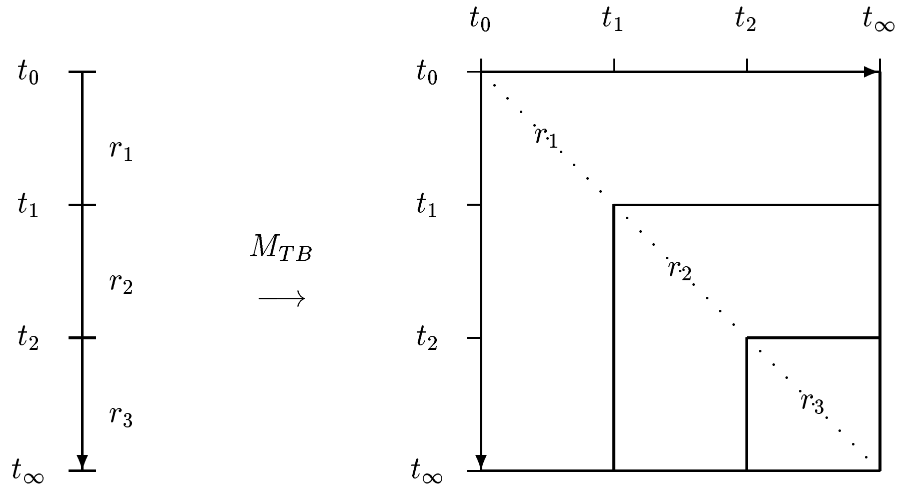

Figure 2: Transaction-time to Bitemporal M_{TB}

In the extension of a transaction-time relation along valid time, a tuple can be considered valid from the time it was inserted, that is from the beginning of its timestamp. As far as the end of validity is concerned, a distinction must be made between current and archived tuples.

A current tuple (e.g. r_3 in Fig. 2) can be considered indefinitely valid from its insertion time, since it cannot be forecasted if a modification transaction will ever arrive and archive such a tuple.

The tuples already archived (e.g. r_1 and r_2 in Fig. 2) can be considered as having their validity restricted by the execution time of the transaction that modified them, which limits their initial time pertinence to an "L-shaped" region symmetric with respect to the diagonal.

The effects of the conversion from valid-time to bitemporal (function M_{VB}) can be seen in Fig. 3.

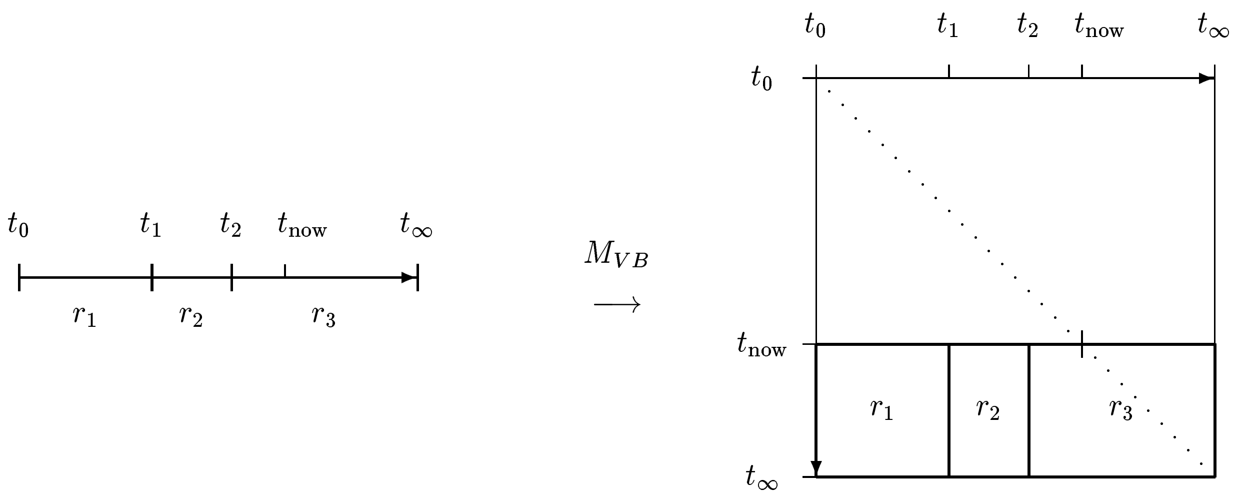

Figure 3: Valid-Time to Bitemporal M_{VB}

2.2 Contraction Functions

The contraction counterparts of the extension principles can be formulated as follows. Provided that the most general and comprehensive representation of data is bitemporal, and bitemporal data can be represented on a rectangular time-domain $(t_0, t_\infty) \times (t_0, t_\infty)$ obtained as the Cartesian product of the time semi-axes, we have:

C_1: to drop transaction-time means to extract from the rectangular domain the data lying on the straight line $t = t_\infty$ parallel to the valid-time axis;

C_2: to drop valid-time means to extract the data lying on the principal diagonal of the time-domain up to t_{now};

The contraction function definitions that follow from the above principles, are consistent with the extension functions: if the symbol "$\circ$" denotes the composition of operations and M_{XY}

(with $X \prec Y$) is a generic extension function, it can easily be shown that:

$$M_{XY} \circ M_{YX} = Id \ ,$$

where M_{YX} is the corresponding contraction function and Id is the identity function.

The conversion from bitemporal to transaction-time (function M_{BT}), defined according to (C_2), is exemplified in Fig. 4. Contraction functions imply a selection of only present/current

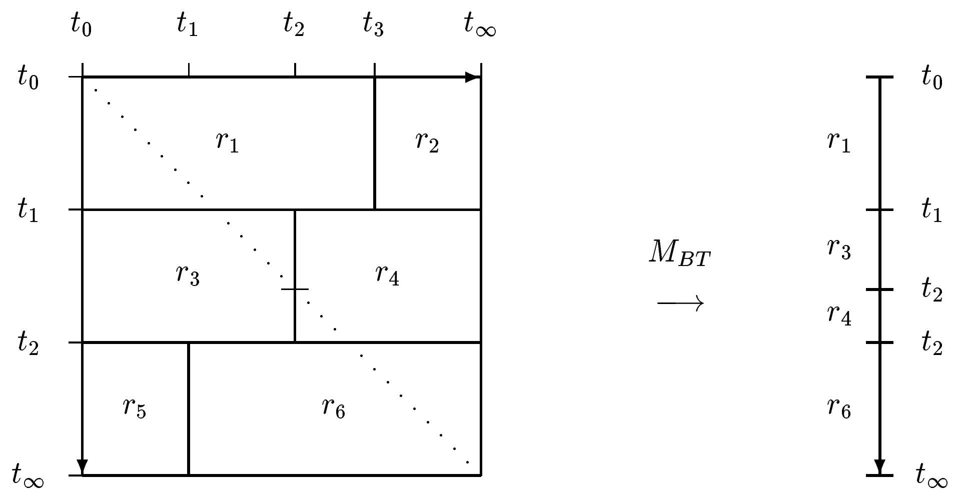

Figure 4: Bitemporal to Transaction-time M_{BT}

tuples in the converted relations and, thus, do not preserve the complete information originally contained in them.

2.3 Orthogonal Functions

The orthogonal conversion functions are not primitive since they can be defined in terms of the extension and contraction functions, according to the following principle:

O_1: The orthogonal functions can be defined as the composition of the extension from the source format to the bitemporal one, followed by the contraction to the target format, that is: $M_{VT} = M_{VB} \circ M_{BT}$ and $M_{TV} = M_{TB} \circ M_{BV}$.

Such compositions give rise to straightforward definitions. Both orthogonal functions, as they involve contractions in their definitions, are not information-preserving.

3 Interoperability with Temporally Incomplete Information

Let us define a *history* as a set of tuples with the same unique identifier in a relation [7]. Identifiers are assumed here to be time-invariant key values, or system-managed surrogates if key changes are allowed. A history represents the collection of all the versions — memorized as tuples — of an *object* [10]. We denote by object a single *entity* or *relationship* instance. Histories cannot contain different tuples overlapping in time since attribute values, at any given time, uniquely depend on the identifier. By the term *temporal incompleteness* of the information concerning an object, we mean that the tuples of the object history do not completely cover the time-universe (e.g. $(t_0, t_\infty) \times (t_0, t_\infty)$ in the bitemporal model).

Temporal incompleteness gives rise to difficulties when binary algebraic operators (e.g. union, difference, Cartesian product, join) are to be used. The reason is that a correct temporal extension of such operators must satisfy the *snapshot-reducibility* property [18, 14]. Snapshot-reducible operators ensure that the result obtained by applying a temporal operator to temporal relations is equivalent to the collection of results obtained by applying the corresponding snapshot operator to all the snapshot relations which can be extracted from the temporal relations. For a full temporal extension of relational algebra see [3].

For instance, the *temporal natural join* [10] must operate with synchronized time attributes (intersection semantics). Algorithms for temporal join, along one time dimension, can be found in the literature (e.g. Event-Join), where a unique surrogate (which always acts as a key) is considered to be a join attribute [16]. Tuples are joined if their surrogates match and their time-intervals overlap. The intersection of the time-intervals is the time-stamp assigned to the resulting join tuple. As far as incomplete relations are concerned, the algorithms proposed for the Event-Join provide the *run-time detection* of portions of histories not covered by matching histories in the other relation. For such regions, outerjoin tuples (with *null* values) are produced. Smart algorithms, exploiting the ordering of the relations and/or using additional data structures, have been proposed.

In the most general case of *bitemporal natural join*, there are two main differences with respect to Event-Join. First, a total ordering cannot be defined on bitemporal intervals or on temporal elements. Even if intervals are adopted, complex data structures (e.g. R-trees [8] instead of B-trees) should be used to efficiently represent covered/uncovered time regions. Second, the join attributes are not necessarily *candidate keys* of one or both of the participating relations, so that several tuples in each relation could have equal values of the join attributes.

Therefore, multiple coverings (between tuples in the former and tuples in the latter relation) must be considered to correctly generate "outerjoin" tuples, for each matching value of the join attributes. This operation may require a large main memory buffer to keep track of all the uncovered time-regions during the join execution. History completion in single relations (before the application of algebraic binary operators) avoids this problem, since only single coverings must be considered at a time.

If the relations to be "interoperated" on in an algebraic expression contain *complete* histories only, no "outerjoin" tuples must be generated and relations can be processed by examining each combination of tuples of the operand relations only once, and "standard" algorithms for the execution of algebraic operations can be used (e.g. "nested loops" for the join execution, if more efficient methods are not available).

If the relations do not contain only complete histories, the algebraic operations must be preceded by a *completion phase* where uncovered time regions of incomplete histories are filled up with *all-null* tuples. An all-null tuple has all the non-temporal attributes but the key set to *null*, whose meaning is "no information available". In a monotemporal model, the uncovered regions are intervals which can simply be used to time-stamp the all-null tuples to be added. In the bitemporal model with interval time-stamping, the uncovered regions must be decomposed into rectangles to time-stamp the all-null tuples to be added (see Fig. 5). In the BCDM, the uncovered regions can always be represented via a single temporal element, thus, exactly one all-null tuple must be added to each incomplete history. Therefore, with element time-stamping, history completion is not a demanding operation in terms of storage overhead, since it requires the addition of only one extra tuple per history. For example, a temporally incomplete relation containing 10,000 histories, composed on average of 100 versions each, contains 1,000,000 tuples before completion and 1,010,000 tuples after completion.

It should be noticed that, without elimination of temporal incompleteness, the temporal natural join with intersection-semantics would not be lossless, as properties:

$$\begin{aligned} R_1 &= \pi_{R_1}(R_1 \bowtie R_2) \\ R_2 &= \pi_{R_2}(R_1 \bowtie R_2) \end{aligned}$$

would no longer be satisfied. To ensure that such properties still hold, it would seem to be sufficient to complete the lifespans of operand relations instead of histories (the lifespan of a relation can be obtained as the union of the lifespans of the histories it contains). However, this would mean inserting tuples with key attributes set to null, which would violate the fundamental relational constraint of entity integrity.

Optimized algorithms for history completion in the presence of interval time-stamping can

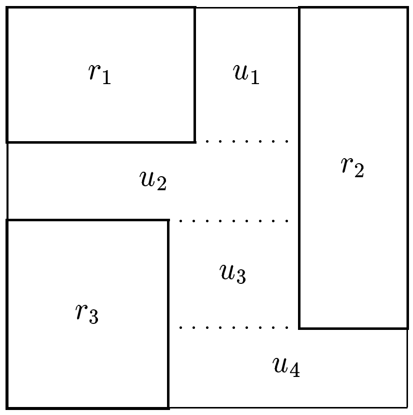

Figure 5: An incomplete bitemporal history (formed by r_1, r_2, r_3) with uncovered regions decomposed into rectangles (u_1, u_2, u_3, u_4).

be found in [2]. When temporal elements are used, history completion is trivial, since it is sufficient to construct the union of the timestamps of all the tuples forming a history, and use its complement to time-stamp the single "completion" tuple.

Finally, if Φ_X denotes the history completion function for the temporal format X, then for the multitemporal algebraic expression:

$$\mathcal{E}_F(R_1, \ldots, R_n)$$

which may contain any unary and binary algebraic operator and parenthesis level, and where Y is the temporal format required for the expression result and X_i is the temporal format of operand relation R_i ($i = 1, \ldots, n$), the only safe and correct evaluation (true semantics) is:

$$M_{BY}(\mathcal{E}'_B(\Phi_B(M_{X_1B}(R_1)), \ldots, \Phi_{X_n}(M_{X_nB}(R_n)))$$

The above formula must be read as follows: all operand relations are first converted to the bitemporal format, history completion functions are then applied to the resulting relations, then the expression $\mathcal{E}$ is evaluated by means of bitemporal algebraic operators ($\mathcal{E}'$ is the expression obtained from $\mathcal{E}$ by replacing the operators), and eventually the result is converted to the temporal format Y. Semantic optimization of such expressions, based on equivalence relations, is the subject of the next section.

4 Semantic Optimization of Multitemporal Expressions

In this section we consider optimization of multitemporal expressions, with respect to the order of execution of algebraic operations, temporal conversion and history completion functions.

Constraints are introduced to perform safe operations. We first assume that relations appearing in algebraic expressions only contain complete histories. This assumption will be relaxed later.

The addition of temporal dimensions yields relations with more complex timestamps, while the elimination of time dimensions yields relations with less tuples and with simpler timestamps, so that the evaluation of algebraic operators is usually more (less) expensive after adding (dropping) time dimensions. Moreover, the bitemporal format is not always necessary to obtain a correct result. For instance, if a snapshot and a valid-time relation must be joined and the result is desired in valid-time format, there is no need for nor convenience in adding the transaction-time dimension to data, evaluating a bitemporal join and eventually putting the result in valid-time format. It is sufficient to add valid time to the snapshot relation and evaluate a valid-time join. Moreover, if the result is desired in snapshot format, the most convenient way to evaluate the expression is to convert the valid-time data to snapshot and apply the snapshot join operator.

Let us consider first the behaviour of binary operators (namely, $\cup, -, \cap, \times, \bowtie$), which are snapshot-reducible. Some details on the multitemporal definition of such operators in the BCDM can be found in [3]. Consider the most important relational operation, the natural join. For example, if R_1 and R_2 are two relations with temporal format X_1 and X_2, respectively, their join can be effected as:

$$M_{JY}(M_{X_1J}(R_1) \bowtie_J M_{X_2J}(R_2)) \ ,$$

where J is the common temporal format adopted for the join execution and Y is the temporal format desired for the final result. The following constraints must be imposed on J in order to avoid loss of information:

$$X_1 \preceq J, X_2 \preceq J \quad \text{or} \quad Y \preceq J \ .$$

If the first two conditions are not met, we may have loss of information along a time dimension which was present in R_1 or R_2 and which is required in the final result. In this case some information, originally stored in R_1 or R_2, may be lost during the application of the functions M_{X_1J} and M_{X_2J} or during the join execution. However, if the condition $Y \preceq J$ is verified, this information can also be discarded since it is not required for the result. In other words, time dimensions required for the result, and already present in the operand relations, must never be eliminated.

For instance, if R_1 is snapshot (S), R_2 is valid-time (V), a natural join between them is needed and the join result needs to be bitemporal (B), then a correct choice is $J = V$ and a convenient operation sequence is therefore:

$$M_{VB}(M_{SV}(R_1) \bowtie_V R_2) \ .$$

If the result needs to be snapshot, a more convenient correct choice is $J = S$, yielding:

$$R_1 \bowtie_S M_{VS}(R_2) \ .$$

In the former case $X_1 \preceq J, X_2 \preceq J$ holds, whereas in the latter case $Y \preceq J$ holds.

Assuming that a snapshot operation is less expensive than a monotemporal one and that a monotemporal operation is less expensive than a bitemporal one, because of the different amount of data usually processed, the following procedure can be used as a heuristic approach to the semantic optimization of a correct binary temporal operator with conversion functions:

- Do the operand relations include time-dimensions not included in the temporal format Y of the result?
  - **Yes:** Drop them.
- Are the operand relations of the same temporal type?
  - **Yes:** Apply the algebraic operator and then convert the result to Y (if necessary).
  - **No:** Convert the relation(s) to Y and then execute the operation.

This procedure applies to all the snaphot-reducible binary operators.

As far as unary operators are concerned, the projection operation π and the selection operator σ with a *non-temporal* restriction condition are also snapshot-reducible, and their behaviour can be derived from the discussion above. Indeed, for an expression involving π:

$$M_{PY}(\pi_{T,P}(M_{XP}(R)),$$

where P is the temporal format adopted for the projection execution, T is the attribute list on which the projection is effected and Y is the temporal format desired for the result, the following constraints must be imposed on J in order to avoid loss of information:

$$X \preceq P \quad \text{or} \quad Y \preceq P \ .$$

If the first condition is not met, we may have loss of information along a time dimension which was present in R but, if the second condition is also verified, such information can be discarded since it is not required for the result. The corresponding optimization procedure is the following:

- Does the operand relation include time-dimensions not included in the temporal format Y of the result?
  - **Yes:** Drop them.

- Apply the algebraic operator and then convert the result to Y (if necessary).

This procedure applies to projection and non-temporal selection.

The only remaining operator is the selection σ with a *temporal* restriction condition, which is not snapshot-reducible and, thus, needs special treatment. In its lossless formulation, both the operand relation and the temporal selection condition must be put in bitemporal format. This means that if the selection only involves valid time, a conjunct `TRANSACTION() OVERLAPS 'NOW'` must be added to the selection condition (there would be no point in adding the companion conjunct `VALID() OVERLAPS (BEGINNING, FOREVER)`, which is always true, to a condition only involving transaction-time). Unfortunately, the cases in which the temporal σ operator can be evaluated in a temporal format less expensive than the bitemporal one are few, and can only be detected through individual case studies. Since the investigation of all the possible cases is tedious and straightforward, we omit it and only briefly report the results:

- If the selection result is needed in transaction-time format, the temporal selection condition only involves transaction time and the operand relation is not bitemporal, then the operand relation can be translated into transaction-time format and the selection evaluated in such a format.

- If the selection result is needed in valid-time format, the temporal selection condition only involves valid time and the operand relation is not transaction-time, then the operand relation can be translated into valid-time format and the selection evaluated in such a format.

- If the selection result is needed in valid-time format, the temporal selection condition only involves valid time and the operand relation is transaction-time, then the operand relation can be translated first into snapshot and then into valid-time format. The selection can finally be evaluated in valid-time format.

- In all the other cases, the selection must be executed in bitemporal format, after the conversion of the operand relation and the selection condition into bitemporal format when necessary. If the selection result is needed in a format different from bitemporal, a final conversion step is executed.

Moreover, if the histories of the relations appearing in the optimized algebraic expression are not complete, applications of the completion operators are needed. However, history completion is only needed for the operands of snapshot-reducible algebraic operators, and for the operand of a non-temporal selection involving the explicit condition `IS NULL`, which would be true for the all-null tuples appended by a history completion.

It could easily be verified that, if $X_2 \prec X_1$ and relation R has temporal format X_1, then we have:

$$M_{X_1X_2}(\Phi_{X_1}(R)) \quad = \quad \Phi_{X_2}(M_{X_1X_2}(R))$$

where Φ_X is the history completion function for the temporal format X. This means that contraction functions are completeness-preserving, that is they map complete histories into complete histories. In any case, it is also convenient to apply the contraction function $M_{X_1X_2}$ to R before completing its histories, for two reasons:

- History completion adds tuples to a relation, that is one per history with element time-stamping, usually much more with intervals. The application of a conversion function to a relation with more tuples is more expensive.
- History completion is less expensive the fewer dimensions the temporal format in which it is effected has: it has a null cost in the snapshot format, and monotemporal completion is less expensive than bitemporal, especially when intervals are used.

On the other hand, extension functions are not, in general, completeness-preserving: it is not lossless to apply an extension function $M_{X_1X_2}$ before the completion function. Anyway, an optimized conversion-completion function $\widehat{M}_{X_1X_2}$ can conveniently be used in this case[1].

We conclude this long discussion with a comprehensive optimization example. Consider the following TSQL2 multitemporal query:

```
SELECT SNAPSHOT E.SALARY
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DEPT=D.NAME
   AND VALID(E) OVERLAPS 'NOW'
   AND D.SITE='NY' AND VALID(D) PRECEDES '1991'
```

which retrieves the present salary of current employees working in a department located in New York before 1991. Assuming the relations `EMPLOYEE` and `DEPARTMENT` to be bitemporal and transaction-time, respectively, the above query is equivalent to the safe (evaluated in bitemporal format) multitemporal algebraic expression:

$$M_{BS}(\pi_{\text{salary},B}(\sigma_{\text{valid overlaps now},B}(E) \bowtie_{\text{E.dept=D.name},B}$$
$$\sigma_{\text{site='NY'},B}(\sigma_{\text{valid precedes 1991},B}(M_{TB}(D)\,))\,))$$

[1]It should be noticed that non-optimized extension functions also introduce history incompleteness themselves, due to information missing along added time dimensions.

First of all, the final result is required to be snapshot and, thus, the final projection can be effected in the snapshot format with a snapshot operand and, thus, the join can also be effected in such a format. Therefore, the expression becomes:

$$\pi_{\text{salary},S}(\sigma_{\text{valid overlaps now},S}(E) \bowtie_{\text{E.dept=D.name},S}$$
$$\sigma_{\text{site='NY'},S}(\sigma_{\text{valid precedes 1991},B}(M_{TB}(D)\,))\,)$$

Let us consider now the join operands. The sub-expression:

$$\sigma_{\text{valid overlaps now},S}(E)$$

can be evaluated in valid-time format, becoming (we make use of the composition operator "$\circ$" in order to save parentheses and improve readability):

$$M_{VS} \circ \sigma_{\text{valid overlaps now},V} \circ M_{BV}(E)$$

The sigma operand needs to be complete, so the application of a completion function is required (and is convenient after the contraction function):

$$M_{VS} \circ \sigma_{\text{valid overlaps now},V} \circ \Phi_V \circ M_{BV}(E)$$

As far as the second join operand is concerned:

$$\sigma_{\text{site='NY'},S}(\sigma_{\text{valid precedes 1991},B}(M_{TB}(D)\,))$$

since the non-temporal selection can be effected in snapshot format, it is equivalent to:

$$\sigma_{\text{site='NY'},S} \circ \sigma_{\text{valid precedes 1991},S} \circ M_{TB}(D)$$

The inner temporal selection (i.e. $\sigma_{\text{valid precedes 1991},S} \circ M_{TB}(D)$) can safely be evaluated as:

$$M_{VS} \circ \sigma_{\text{valid precedes 1991},V} \circ M_{SV} \circ M_{TS}(D)$$

The σ operand needs to be complete, so an optimized conversion-completion function $\widehat{M}_{SV}$ from snapshot to valid can be used (notice that snapshot histories are trivially always complete):

$$M_{VS} \circ \sigma_{\text{valid precedes 1991},V} \circ \widehat{M}_{SV} \circ M_{TS}(D)$$

After the execution of the selections, all the expression is in snapshot format and, therefore, no other history completions are needed. The optimized form of our expression is eventually:

$$\pi_{\text{salary},S}(M_{VS} \circ \sigma_{\text{valid overlaps now},V} \circ \Phi_V \circ M_{BV}(E) \bowtie_{\text{E.dept=D.name},S}$$
$$\sigma_{\text{site='NY'},S} \circ M_{VS} \circ \sigma_{\text{valid precedes 1991},V} \circ \widehat{M}_{SV} \circ M_{TS}(D))$$

5 Conclusions

The cooperative use of heterogeneous temporal data in a multi+temporal environment offers a new and attractive perspective for temporal applications and provides several challenging issues in the field of multidatabase semantic interoperability.

In this paper three main problems have been covered, completing the work started in [2, 3]. The first concerns the translation of relational data from one temporal format into another. This problem has been solved by means of the introduction of temporal conversion functions. The functions proposed can be used as a general interface for data exchange between different temporal relational models, and provide the preliminary step for interoperating relations with different temporal formats. The second problem concerns the extension of the temporal algebra in order to work in the presence of incomplete information. Solutions have been introduced for the elimination of temporal incompleteness before the evaluation of algebraic expressions. After the completion, algebraic operations can be executed by means of standard algorithms. The third problem concerns semantic optimization of multitemporal algebraic expressions, that is the study of the constraints to be imposed on the execution order of conversion, algebraic, and completion operations to obtain correct results without loss of information. Among correct operation sequences, directions for semantic optimization of multitemporal algebraic expressions have been proposed in this paper.

Although the general problem of query processing and optimization in a multi+temporal heterogeneous environment is far from being solved, future work may take advantage of the solutions outlined in this work.

References

[1] Bhargava G., Gadia S.K., "Relational Database Systems with Zero Information Loss," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, No. 1, Feb. 1993.

[2] De Castro C., Grandi F., Scalas M.R., "Semantic Interoperability of Multitemporal Relational Databases," in *Entity-Relationship Approach - ER '93*, Lecture Notes in Computer Science, Vol. 823, Springer Verlag, 1994.

[3] De Castro C., Grandi F., Scalas M.R., "Meaning of Relational Operations in a Temporal Environment," *Proc. Basque International Workshop on Information Technology*, San Sebastian, 1995.

[4] De Castro C., Scalas M.R., "Inferring Validity from Transaction Time," *Proc. Intl. Workshop on Temporal Representation and Reasoning (TIME'95)*, Florida, 1995.

[5] Elmagarmid A.K., Pu C. (eds.), Special Issue on Heterogeneous Databases of *ACM Computing Surveys*, Vol. 22, No. 3, Sept. 1990.

[6] Elmasri R., El-Assal I., Kouramajian V., "Semantics of Temporal Data in an Extended ER Model," *Proc. Intl. Conf. on E-R Approach*, Lausanne, 1990.

[7] Grandi F., Scalas M.R., Tiberio P., "A History Oriented Data View and Operation Semantics for Temporal Relational Databases," C.I.O.C.-C.N.R. Tech. Rep. No. 76, Bologna, Jan. 1991.

[8] Guttman A., "The R-trees: A Dynamic Index Structure for Spatial Searching," *Proc. of ACM SIGMOD Conf.*, Boston, 1984.

[9] Jajodia S., Wang X.S., "Temporal Mediators: Supporting Uniform Access to Heterogeneous Temporal Databases," *Proc. Workshop on Interoperability of Database Systems and Database Applications*, Fribourg, 1993.

[10] Jensen C.S., Clifford J., Elmasri R., Gadia S.K., Hayes P., Jajodia S. (eds.), Dyreson C., Grandi F., Käfer W., Kline N., Lorentzos N, Mitsopoulos Y., Montanari A., Nonen D., Peressi E., Pernici B., Roddick J.F., Sarda N.L., Scalas M.R., Segev A., Snodgrass R.T., Soo M.D., Tansel A., Tiberio P., Wiederhold G., "A Consensus Glossary of Temporal Database Concepts," *ACM SIGMOD Record*, Vol. 23, No. 1, March 1994.

[11] Jensen C.S., Soo, M.D., Snodgrass R.T., "Unifying Temporal Data Models via a Conceptual Model," *Information Systems*, Vol. 19, No. 7, 1994.

[12] Kline N., "An Update of the Temporal Database Bibliography," *ACM SIGMOD Record*, Vol. 22, No. 4, Dec. 1993.

[13] Litwin W., Abdellativ A., "Multidatabase Interoperability," *IEEE Computer*, Dec. 1986.

[14] McKenzie L.E., Snodgrass R.T., "Evaluation of Relational Algebras Incorporating the Time Dimension in Databases," *ACM Computing Surveys*, Vol. 23, No. 4, Dec. 1991.

[15] Navathe S.B., Ahmed R., "A Temporal Relational Model and a Query Language," *Information Sciences*, Vol. 49, 1989.

[16] Segev A., Gunadhi H., “Event-Join Optimization in Temporal Relational Databases,” *Proc. of Intl. VLDB Conf.*, Amsterdam, 1989.

[17] Sheth A.P. (ed.), Special Issue on Semantic Issues in Multidatabase Systems, *ACM SIGMOD Record*, Vol. 20, No. 4, Dec. 1991.

[18] Snodgrass R.T., “The Temporal Query Language TQuel,” *ACM Trans. on Database Systems*, Vol. 12, No. 2, June 1987.

[19] Snodgrass R.T. (ed.), Ilsoo Ahn, Gadi Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, T.Y. Cliff Leung, Nikos Lorentzos, Raghu Ramakrishnan, John F. Roddick, Arie Segev, Michael D. Soo, Suryanarayana M. Sripada, “TSQL2 Language Specification,” *ACM SIGMOD Record*, Vol. 23, No. 1, 1994.

[20] Snodgrass R.T. (ed.), Ilsoo Ahn, Gadi Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käefer, Nick Kline, Krishna Kulkarni, T.Y. Cliff Leung, Nikos Lorentzos, Raghu Ramakrishnan, John F. Roddick, Arie Segev, Michael D. Soo, Suryanarayana M. Sripada, *The TSQL2 Temporal Query Language*, Kluwer Academic Publishers, 1995.

[21] Soo M., “Bibliography on Temporal Databases,” *ACM SIGMOD Record*, Vol. 20, No. 1, Mar. 1991.

[22] SQL-92, *Information Technology - Database Languages - SQL*, ISO/IEC 9075:1992 (E) International Standard, 1992.

[23] Tansel A., Snodgrass R.T., Clifford J., Gadia S.K., Jajodia S., Segev A. (eds.), *Temporal Databases, Theory, Design and Implementation*, Benjamin/Cummings, 1993.