Querying now-relative data

Author

Anselma, Luca, Stantic, Bela, Terenziani, Paolo, Sattar, Abdul

Published

2013

Journal Title

Journal of Intelligent Information Systems

DOI

10.1007/s10844-013-0245-8

Noname manuscript No.
(will be inserted by the editor)

Querying now-relative data

**Luca Anselma, Bela Stantic, Paolo Terenziani,
Abdul Sattar**

Received: 20/7/2012 / Accepted: 22 April 2013

Abstract *Now-relative* temporal data play an important role in most temporal applications, and their management has been proved to impact in a crucial way the efficiency of temporal databases. Though several *temporal relational* approaches have been developed to deal with now-relative data, none of them has provided a whole *temporal algebra* to query them. In this paper we overcome such a limitation, by proposing a general algebra which is parametrically adapted to cope with the relational approaches to now-relative data in the literature, i.e., *MIN*, *MAX*, *NULL* and *POINT* approaches. Besides being general enough to provide a query language for several approaches in the literature, our algebra has been designed in such a way to satisfy several theoretical and practical desiderata: *closure* with respect to representation languages, *correctness* with respect to the "consensus" *BCDM* semantics, *reducibility* to the standard non-temporal algebra (which involves *interoperability* with non-temporal relational databases), *implementability* and *efficiency*. Indeed, the *experimental evaluation* we have drawn on our implementation has shown that only a slight overhead is added by our treatment of now-relative data (with respect to an approach in which such data are not present).

Keywords Temporal relational databases · Now-related data · Querying bitemporal data · Algebraic operators · Experimental evaluation

L. Anselma
Dipartimento di Informatica, Università di Torino, Italy
E-mail: anselma@di.unito.it

B. Stantic
Institute for Integrated and Intelligent Systems, Griffith University, Brisbane Australia
E-mail: b.stantic@griffith.edu.au

P. Terenziani
Dipartimento di Informatica, Università del Piemonte Orientale "Amedeo Avogadro", Alessandria, Italy
E-mail: paolo.terenziani@mfn.unipmn.it

A. Sattar
Institute for Integrated and Intelligent Systems, Griffith University, Brisbane Australia
National ICT Australia, Brisbane Australia
E-mail: a.sattar@griffith.edu.au

1 Introduction

Temporal data play an important role in many domains and applications. In such contexts, data must be paired with the time when they occur (*valid time* henceforth). Additionally, there might be other temporal dimensions of interest. For example transaction time captures the time when the data are inserted/deleted in the database (in some applications this must be maintained for legal purposes). Referential time and other notions of temporal dimension are less used since additional research is required to fully understand how to cope with them. As a consequence, starting from the 1980's, there is a long tradition of approaches coping with valid and/or transaction time in relational databases. As a matter of fact, more than twenty years of research in the area of relational databases have widely demonstrated that the treatment of time in the relational approach involves the solution of difficult problems, and the adoption of advanced dedicated techniques. Practical examples of the problems to be faced are presented in Chapter 1 of the TSQL2 book [24]. Given the pervasive character of time, great efforts in terms of research were made in order to provide once-and-for-all a general solution to such problems. In this spirit, many extensions to the standard relational model were devised, and more than 2000 papers on temporal databases (TDBs) were published over two decades (cf., the cumulative bibliography in [34], the section about TDBs in the Springer Encyclopedia of Databases [16], which includes over 90 entries about TDBs, the entry "Temporal Database" in [16], and the surveys in [15, 29, 21, 12]).

Despite such a wide range of approaches, some "consensus" has been found by the TDB community. TSQL2 [24] is a temporal relational approach coping with bi-temporal data (i.e., with both valid and transaction time) which has been defined by a significant number of international researchers in the area. *BCDM* (Bitemporal Conceptual Data Model) [13] provides a unifying semantics for TSQL2 and several other approaches in the literature. However, several open issues still have to be addressed. One of them is the treatment of *now-relative* data.

Among temporal data, now-relative information play an important role. A temporal piece of data (tuple) is now-relative if one of the following conditions (or both) holds:

- it is part of the current status of the database (i.e., it has been inserted in the past, and has not been deleted yet); in such a case, the ending point of its transaction time is usually set to *now*, to represent the fact that, in the current status, it is part of the database;
- it is currently valid (i.e., the fact it describes holds at the current time and its ending time is unknown); in such a case, the ending point of its valid time is set to *now*, to state that it is currently valid.

An efficient treatment of *now* in temporal databases is very important, since now-relative facts may be very frequent, and are likely to be accessed more frequently [5]. As a matter of fact, it has been shown that the choice of the physical value for *now* significantly influences the efficiency of accessing temporal data [31].

Many recent approaches have focused on indexing [32, 27, 22, 17, 20] or timestamping [32, 11] now relative data. Other approaches have provided coalescing [6] and split [1] operators for such data, or have focused on their representation [18] or semantics [23] (see also the surveys in [16, 5]).

In the recent Database Encyclopedia by Springer, Dyreson, Jensen and Snodgrass [16] have pointed out that there are three different uses of *now* in databases. The first use of *now* is as a function within queries, views, assertions. The second use is as a database variable used as a special timestamp value associated with tuples or attribute values in TDB instances. The

third use of *now* is as a database variable with a specified offset. In this paper, we focus on the second use.

There are two kinds of treatment for now-relative data in TDBs. In the first, variables (e.g., *now* , $until-changed$, $forever$, ∞, @, and $-$) are introduced, leading to Variable databases [3]. The semantics of *now* variables have been widely studied [3] [16]. For instance, in [3], the semantics of *now* variables has been formalized through the introduction of *extensionalization* functions, that map from a database containing variables into an extensional database level which is fully ground and constitutes its sematics, at a given *reference time*.

A significant amount of work has also been devoted to the problem of querying (see, e.g., [3]) and updating (consider, e.g., [33]) *variable databases*. However, Variable databases require a significant departure from the "consensus" relational model (since the domain of SQL1999 values [19] does not support such variables) and cannot be easily implemented on existing relational databases. Variable databases will not be considered henceforth.

In the second kind, which we follow here, the relational model is extended. The literature contains three earlier approaches to denote the value *now*: (1) using *NULL*, (2) using the smallest timestamp (the minimum-value approach *MIN*), (3) using the largest timestamp (*MAX* approach) [31]. The *MAX* approach outperforms the *NULL* and *MIN* approaches [31]. The recently proposed *POINT* approach [25] outperforms the *MAX* approach for range queries.

Most approaches have focused on queries selecting tuples holding in a certain period (range queries) or point (slice query) in time. However, it is important to provide a full extensive query language for now-relative data. Specifically, one needs a temporal algebra coping with (bitemporal) now-relative data. This, to the best of our knowledge, has not yet been done.

This paper proposes a parametric algebra which, properly instantiated, copes with the *MIN*, *MAX*, *NULL* and *POINT* approaches. Our parametric algebra has been designed to satisfy several theoretical and practical desiderata:

- **closure**: the algebraic operators must be closed with respect to the representation language;
- **correctness**: the algebraic operators are semantically correct considering the *BCDM* semantics;
- **reducibility**: our temporal relational algebraic operators behave exactly as non-temporal relational algebraic ones on relations with no temporal information. This property grants interoperability with standard non-temporal relational approaches;
- **implementability**: although theoretically grounded, our algebra must be implementable. We observe that the reducibility property is important also to this respect, since it grants that the temporal algebra can be implemented on top of current non-temporal approaches;
- **efficiency**: last, but not least, our approach must be efficient. Indeed, in the paper we also propose an extensive experimental evaluation showing that, both in the cases of the *POINT* and of the *MAX* approach, our temporal algebra does not add any significant overload with respect to the "ideal" (but unrealistic) approach in which now-relative data are not present (since one knows the future end time of all data).

The paper is organized as follows. Section 2 is a preliminary one, in which we briefly describe the *MIN*, *MAX*, *NULL* and *POINT* approaches, and we discuss the *BCDM* semantic model, which we assume as the reference model to deal with the semantics of our approach, and to prove its correctness. In Section 3 we provide the formal semantics of the *MIN*, *MAX*, *NULL* and *POINT* approaches. Section 4 describes the core contributions of

our work, namely the description of the parametric algebra for the *MIN*, *MAX*, *NULL* and *POINT* approaches, and of its properties. Finally, in the absence of a "competitor" in the literature, we introduce an optimal (but unfeasible) approach to now-relative data (called $NOT-NOW$ approach). Section 5 describes our extensive experimental evaluation, showing that our approach only adds a slight overhead to the optimal $NOT-NOW$ approach. Finally, Section 6 contains comparisons and conclusions.

2 Preliminaries

In this section, we set up the stage presenting the background of our approach.

2.1 *MIN*, *MAX*, *NULL* and *POINT* representations

The basic idea of the *MAX* representation [32] is very simple: the largest database timestamp (represented by the value *max-value* henceforth) is used to denote *now* along both (transaction and valid) temporal dimensions.

Example 1 Let us consider a relation *Employee*, recording employee identifiers (*Id* attribute) and departments (*Dept* attribute). The fact that *John* has been employed in the toy department from day 10 to day 12, and that such a piece of information has been inserted at transaction time 11, and that at the current time $c_{now} = 14$ is still present in the database (i.e., it has not been deleted) is represented in the *MAX* approach by the tuple $[(John, toy|11, max-value, 10, 13)]$, where 11, $max-value$, 10 and 13 are the timestamps denoting the start and end of the transaction and of the valid times, respectively.

For the sake of homogeneity with the *POINT* approach, we assume that also in the *MAX* approach time periods are closed to the left and open to the right.

The *MIN* and *NULL* representations are similar to the *MAX* representation: the smallest database timestamp (represented by the value *min-value* henceforth) and the *NULL* value are respectively used to denote *now* along the temporal dimensions.

In the *POINT* approach [27] "degenerated" periods in which the starting point is equal to the ending point are used to represent *now* along the transaction-time and/or valid time dimensions.

Example 2 Example 1 above can be represented in the *POINT* approach by the tuple $[(John, toy|11, 11, 10, 13)]$ where the transaction-time period $[11, 11)$ is the representation used to deal with the period $[11, now]$.

In all the representations, data may be now-relative in both temporal dimensions. Additionally, in line with many approaches in the literature (see [7]), no approach allows *now* to be the value of the starting point of the transaction or the valid time (i.e., a period $[now, 100)$, with 100 greater than the current system time is not allowed either along the transaction- or along the valid time dimensions).

2.2 The *BCDM* semantic model

BCDM (Bitemporal Conceptual Data Model) [13, 24] is a unifying *semantic* data model, which has been developed in order to isolate the *core* notions underlying many temporal relational approaches, including the "consensus" TSQL2 one [24]. *BCDM* does not face issues such as data representation and storage optimization, aiming at a purely "semantic" approach. In *BCDM*, tuples are associated with valid time and transaction time. For both domains, a limited precision is assumed (the chronon is the basic time unit). Both time domains are totally ordered and isomorphic to the subsets of the domain of natural numbers. The domain of valid times D_{VT} is given as a set $D_{VT} = \{t_1, t_2, \ldots, t_k\}$ of chronons, and the domain of transaction times as $D_{TT} = \{t'_1, t'_2, \ldots, t'_j\} \cup \{UC\}$ (where UC -Until Changed- is a distinguished value). In general, the schema of a bitemporal conceptual relation $R = (A_1, \ldots, A_n|T)$ consists of an arbitrary number of non-timestamp attributes $A_1, \ldots, A_n$, encoding some fact, and of a timestamp attribute T, with domain $D_{TT} \times D_{VT}$. Thus, a tuple $x = (a_1, \ldots, a_n \mid t_b)$ in a bitemporal relation $r(R)$ on the schema R consists of a number of attribute values associated with a set of bitemporal chronons $t_{bi} = (c_{ti}, c_{vi})$, with $c_{ti} \in D_{TT}$ and $c_{vi} \in D_{VT}$. The intended meaning of a bitemporal *BCDM* tuple is that the recorded fact is true in the modeled reality during each valid-time chronon in the set, and is current in the relation during each transaction-time chronon in the set. Valid time, transaction-time and atemporal tuples are special cases, in which either the transaction time, or the valid time, or none of them are present.

The *BCDM* model explicitly requires that no two tuples with the same data part (i.e., *value-equivalent* tuples [24]) are allowed in the same relation. As a consequence, in *BCDM* the full history of a fact is recorded in a single tuple. The special value UC is used to model *now* along the transaction-time dimension. A special routine makes explicit the semantics of the special value UC: as time passes, at each clock tick for each bitemporal chronon (UC, c_v), a new bitemporal chronon (c_t, c_v) is added to the set of chronons, where c_t is the new transaction-time value.

Example 3 Example 1 can be represented in *BCDM* as follows:
$(John, toy \mid \{(11, 10), (11, 11), (11, 12), (12, 10), (12, 11), (12, 12), (13, 10), (13, 11), (13, 12), (14, 10), (14, 11), (14, 12), (UC, 10), (UC, 11), (UC, 12)\}$

An extension of standard relational algebraic operators has been provided to operate on the *BCDM* model, and proper insertion and deletion manipulation operations have been also defined.

BCDM is *reducible* and *equivalent* to the standard relational model [24]. These properties conjunctively grant that *BCDM* is a consistent extension of the standard (non-temporal) relational model, and that it operates in the same way as the standard model when time is disregarded.

We recall that, in the *BCDM* model, *now* is only coped with along the transaction-time dimension, through the introduction of the UC special symbol. Moreover, *BCDM* treatment of *now* is not feasible in practical implementations: of course, no approach can update the temporal component of all *current* tuples (i.e., tuples such that the end of the transaction time is set to UC) at each tick of the system's clock!

3 Semantics of the *MIN*, *MAX*, *NULL* and *POINT* approaches

In this section we introduce a set of functions which relate the *MIN*, *MAX*, *NULL* and *POINT* representations to the reference *BCDM* one. Such functions make the underlying semantics of the *MIN*, *MAX*, *NULL* and *POINT* representations explicit, and will be used in the following sections to prove the correctness of our approach.

We aim at providing a parametric temporal relational algebra which, in particular, can be properly instantiated to cope with the *MIN*, *MAX*, *NULL* and *POINT* approaches. To obtain such a goal, we introduce the *isNowRelative* and *setNow* functions to abstract from the specific way that the different approaches use to *represent* now-relative data. Specifically, the former function takes in input a temporal interval (represented by its starting and ending timestamps) and returns true if the period is now-relative; the latter takes in input a (now-relative) period (represented by its starting and ending timestamps) and returns the value used in order represent the value *now* in the specific approach.

Function 1 $isNowRelative(Start, End)$

case of: *POINT* approach:	**if** $Start = End$ **then return true else return false**
case of: *MAX* approach:	**if** $End = max - value$ **then return true else return false**
case of: *MIN* approach:	**if** $End = min - value$ **then return true else return false**
case of: *NULL* approach:	**if** $End = NULL$ **then return true else return false**

Function 2 $setNow(Start)$

case of: *POINT* approach:	**return** $Start$
case of: *MAX* approach:	**return** $max - value$
case of: *MIN* approach:	**return** $min - value$
case of: *NULL* approach:	**return** $NULL$

In the rest of this section, we focus only on *POINT* and *MAX* approaches since the treatment of *MIN* and *NULL* is analogous. We define the function $to\_BCDM^{c_{NOW}}(r)$, that converts a relation from either the *MAX* or the *POINT* representation to a *BCDM* representation; it assumes the value c_{NOW} for *now* (i.e., c_{NOW} is the timestamp that represents the current time). We adopt the following notation.

Notation 1 *Given a tuple x defined on the schema* $R = (A_1, \ldots, A_n \mid T)$*, we denote with A the set of attributes* $(A_1, \ldots, A_n)$*. Then* $x[A]$ *denotes the values in x of the attributes in A,* $x[T]$ *denotes the values of the temporal attributes of x. In particular,* $x[TT_S]$, $x[TT_E]$, $x[VT_S]$, $x[VT_E]$ *denote the starting and ending points of the transaction and valid times of tuples, respectively.*

The $to\_BCDM^{c_{NOW}}(r)$ function is very similar to *snap_to_conceptual* in TSQL2 [24] and basically associates with each tuple in the (*POINT* or *MAX*) representation the set of all the bi-temporal chronons corresponding to the rectangle represented by its temporal attributes (i.e., start and end of transaction and valid times). To achieve such a result, it adopts the $EXT^{c_{NOW}}$ function (where *EXT* stands for "temporal extension"), which only

operates on the temporal component of a tuple. In turn, $EXT^{c_{NOW}}$ is defined on the basis of two functions: $EXT\_TT^{c_{NOW}}(Start,\ End)$ and $EXT\_VT^{c_{NOW}}(Start,\ End)$.

$EXT\_TT^{c_{NOW}}(Start,\ End)$ operates on transaction times only. It takes in input the starting and the ending timestamp of a transaction-time period and it returns the set of transaction-time chronons included in the period. We observe that, since we assume a representation closed to the left and open to the right, the ending timestamp is not included. If the period is now-relative, the transaction-time chronons also include the chronon c_{NOW}. Moreover, since *BCDM* supports now-relative transaction time, the proper chronon UC is added to the set of transaction-time chronons. $EXT\_VT^{c_{NOW}}(Start,\ End)$, operating on valid times, is analogous to $EXT\_TT^{c_{NOW}}(Start,\ End)$, and it is not reported. Since *BCDM* does not support now-relative valid times, $EXT\_VT^{c_{NOW}}(Start,\ End)$ simply "instantiates" *now* at c_{NOW}.

Function 3 $EXT\_TT^{c_{NOW}}(Start,\ End)$

chronons $\leftarrow \emptyset$
if *isNowRelative*$(Start, End)$ **then** *endChronon* $\leftarrow c_{NOW} + 1$ **else** *endChronon* $\leftarrow End$
for all chronons *chron* such that $Start \leq chron < endChronon$
    chronons $\leftarrow$ *chronons* $\cup \{chron\}$
if *isNowRelative*$(Start, End)$ **then** *chronons* $\leftarrow$ *chronons* $\cup \{UC\}$
return *chronons*

The function $EXT^{c_{NOW}}(TT_S,\ TT_E,\ VT_S,\ VT_E)$ takes in input the starting and ending timestamps of a bitemporal period, and returns the corresponding set of bitemporal chronons, which is obtained by performing the cartesian product of the chronons representing the transaction and the valid times (as obtained through the application of the $EXT\_TT^{c_{NOW}}$ and $EXT\_VT^{c_{NOW}}$ functions).

Function 4 $EXT^{c_{NOW}}(TT_S,\ TT_E,\ VT_S,\ VT_E)$

return $EXT\_TT^{c_{NOW}}(TT_S,\ TT_E)\ \times\ EXT\_VT^{c_{NOW}}(VT_S,\ VT_E)$

Finally, $to\_BCDM^{c_{NOW}}(r)$ iterates the $EXT^{c_{NOW}}$ function on (the temporal components of) all the tuples in the input relation r. It is worth noting that, since *BCDM* does not admit value-equivalent tuples, the $to\_BCDM^{c_{NOW}}$ function must merge together (i.e., coalesce [2]) all value-equivalent tuples in the *MAX* or *POINT* representations, in order to produce one *BCDM* tuple from each set of value-equivalent representational tuples.

Example 4 Applying the *to_BCDM* function to Example 1 gives as a result the tuple in Example 3.

4 Algebra

In this section, we introduce a family of temporal query languages operating on (possibly now-relative) bitemporal data. For the sake of compactness, generality and clarity, we operate at the algebraic level.

Function 5 $to\_BCDM^{c_{NOW}}(r)$

```
s ← ∅
for all z ∈ r do
  x[A] ← z[A]
  x[T] ← EXT^cNOW(z[TT_S], z[TT_E], z[VT_S], z[VT_E])
  for all y ∈ r do
    if z[A] = y[A] then
      r ← r − {y} /* Remove y from r so that it will not be considered in the next
      iterations. */
      x[T] ← x[T] ∪ EXT^cNOW(y[TT_S], y[TT_E], y[VT_S], y[VT_E])
    end if
  end for
  s ← s ∪ {x}
end for
return s
```

Codd designated as complete any query language that was as expressive as his set of five relational algebraic operators, relational union ($\cup$), relational difference ($-$), selection (σ_P), projection (π), and Cartesian product ($\times$) [4]. In this section we propose an extension of Codd's algebraic operators in order to query the data models introduced in Section 2.1.

Several temporal extensions have been provided to Codd's operators in the TDB literature [24, 15]. In many cases, such extensions behave as standard non-temporal operators on the non-temporal attributes, and involve the application of set operators on the temporal parts. This approach ensures that the temporal algebrae are a consistent extensions of Codd's operators and are reducible to them when the temporal dimension is removed. For instance, in *BCDM* and in *TSQL2*, temporal Cartesian product involves pairwise concatenation of the values for non-temporal attributes of tuples and pairwise intersection of their temporal values. We ground our approach on such a "consensus" background, extending it in order to cope with the *MIN*, *MAX*, *NULL* and *POINT* representations of now-relative data.

4.1 Temporal algebrae

Instead of proposing four different algebrae (for *MIN*, *MAX*, *NULL* and *POINT* approaches), we capture the generalities between the four and "hide" the differences through the adoption of the *isNowRelative* and *setNow* functions described above. The specific temporal algebra for the *MIN*, *MAX*, *NULL* and *POINT* approaches can be simply obtained by properly instantiating *isNowRelative* and *setNow*.

Our definition of temporal Cartesian product is reported in the following. In the definitions below, we denote by $t[X_1, \ldots, X_k]$ the value of the attributes $X_1, \ldots, X_k$ in the tuple t and by D_{X_i} the domain of the attribute $X_i, i = 1, \ldots, k$. We present our operators following the style and the notation used in *TSQL2*. However, to distinguish the alternative possibilities involved by Cartesian product, in the definition we also use the "if-then-else" construct with the standard interpretation.

Definition 1 (Temporal Cartesian product $\times^N$) Given two bitemporal relations r and s in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the schemas $R_1^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$ and $R_2^N = (B_1, \ldots, B_k \mid TT_S, TT_E, VT_S, VT_E)$ respectively,

the temporal Cartesian product $r \times^N s$ is a temporal relation q defined over the schema $R_3^N = (A_1, \ldots, A_n, B_1, \ldots, B_k \mid TT_S, TT_E, VT_S, VT_E)$ and is defined as follows:
$r \times^N s = \{t \in D_{A_1} \times \ldots \times D_{A_n} \times D_{B_1} \times \ldots \times D_{B_k} \times D_{TT} \times D_{TT} \times D_{VT} \times D_{VT} \mid$
$\exists\, t_r \in r, \exists\, t_s \in s, t[A_1, \ldots, A_n] = t_r[A_1, \ldots, A_n] \wedge t[B_1, \ldots, B_k] = t_s[B_1, \ldots, B_k] \wedge$
if $(isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge isNowRelative(t_s[TT_S], t_s[TT_E]))$ **then**
$t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = setNow(t[TT_S])$
else if $(isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge \neg\, isNowRelative(t_s[TT_S], t_s[TT_E]))$ **then**
$t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = t_s[TT_E] \wedge t[TT_S] < t[TT_E]$
else if $(\neg\, isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge isNowRelative(t_s[TT_S], t_s[TT_E]))$ **then**
$t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = t_r[TT_E] \wedge t[TT_S] < t[TT_E]$
else if $(\neg\, isNowRelative(t_r[TT_S], t_r[TT_E]) \wedge \neg isNowRelative(t_s[TT_S], t_s[TT_E]))$ **then**
$t[TT_S] = max(t_r[TT_S], t_s[TT_S]) \wedge t[TT_E] = min(t_r[TT_E], t_s[TT_E]) \wedge$
$t[TT_S] < t[TT_E]$
if $(isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge isNowRelative(t_s[VT_S], t_s[VT_E]))$ **then**
$t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = setNow(t[VT_S])$
else if $(isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge \neg\, isNowRelative(t_s[VT_S], t_s[VT_E]))$ **then**
$t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(c_{NOW} + 1, t_s[VT_E]) \wedge$
$t[VT_S] < t[VT_E]$
else if $(\neg\, isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge isNowRelative(t_s[VT_S], t_s[VT_E]))$ **then**
$t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(c_{NOW} + 1, t_r[VT_E]) \wedge$
$t[VT_S] < t[VT_E]$
else if $(\neg\, isNowRelative(t_r[VT_S], t_r[VT_E]) \wedge \neg\, isNowRelative(t_s[VT_S], t_s[VT_E]))$ **then**
$t[VT_S] = max(t_r[VT_S], t_s[VT_S]) \wedge t[VT_E] = min(t_r[VT_E], t_s[VT_E]) \wedge$
$t[VT_S] < t[VT_E]\}$.

The result of the Cartesian product is a relation whose schema contains both the explicit attributes of r and of s. The timestamps of tuples in q correspond to the intersection of timestamps of the corresponding tuples in r and s, possibly now-relative.

In the definition, the transaction and the valid times of the output tuples are independently defined by cases. For instance, the first case states that, in case the transaction time of both t_r and t_s is now-relative, then the output transaction time is still now-relative along the transaction-time dimension. It is worth stressing that, in all the cases in which the tests $t[TT_S] < t[TT_E]$ or $t[VT_S] < t[VT_E]$ fail, the corresponding tuple is not part of the output. For instance, the second case in the definition states that if the transaction time of t_r is now-relative and the transaction time of t_s is not now-relative, then the resulting transaction time is not now-relative (along the transaction-time dimension) and can be determined as follows. Its starting point is the maximum between the two starting times and its ending point is the minimum between the two ending times. Specifically, since the transaction time of t_r is now-relative, for the semantics of transaction time, certainly the transaction time of t_s ends before *now* and it is the minimum. However, if the result of such an evaluation is a degenerated period whose ending point is not after the starting point (which happens just in case there is no intersection between the two transaction times), then the tuple is not part of the output.

Temporal relational union takes in input the tuples of two bitemporal relations r and s, and gives them in output unchanged both in the non-temporal and temporal part.

Definition 2 (Temporal union $\cup^N$) Given two bitemporal relations r and s in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the same schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$, the temporal union $r \cup^N s$ is a bitemporal relation q defined over the schema R^N, and is defined as follows:

$$
\begin{aligned}
r \cup^N s = \{t \in\ & D_{A_1} \times \ldots \times D_{A_n} \times D_{TT} \times D_{TT} \times D_{VT} \times D_{VT} \mid \\
& \exists\, t_r \in r,\ t[A_1, \ldots, A_n] = t_r[A_1, \ldots, A_n] \wedge t[TT_S] = t_r[TT_S] \wedge \\
& t[TT_E] = t_r[TT_E] \wedge t[VT_S] = t_r[VT_S] \wedge t[VT_E] = t_r[VT_E] \vee \\
& \exists\, t_s \in s,\ t[A_1, \ldots, A_n] = t_s[A_1, \ldots, A_n] \wedge t[TT_S] = t_s[TT_S] \wedge \\
& t[TT_E] = t_s[TT_E] \wedge t[VT_S] = t_s[VT_S] \wedge t[VT_E] = t_s[VT_E]\}
\end{aligned}
$$

Temporal relational projection simply operates on the non-temporal part of the input tuples, retaining only the values of the input attributes. The temporal component of the tuples is left unchanged.

Definition 3 (Temporal projection $\pi^N_{A_i, \ldots, A_j}$) Given a bitemporal relation r in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$, and given a subset $\{A_i, \ldots, A_j\}$ of the set $\{A_1, \ldots, A_n\}$, temporal projection $\pi^N_{A_i, \ldots, A_j}(r)$ is a bitemporal relation q defined over the schema $R'^N = (A_i, \ldots, A_j \mid TT_S, TT_E, VT_S, VT_E)$, and is defined as follows:

$$
\begin{aligned}
\pi^N_{A_i, \ldots, A_j}(r) = \{t \in\ & D_{A_1} \times \ldots \times D_{A_j} \times D_{TT} \times D_{TT} \times D_{VT} \times D_{VT} \mid \\
& \exists\, t' \in r,\ t[A_i, \ldots, A_j] = t'[A_i, \ldots, A_j] \wedge t[VT_S] = t'[VT_S] \wedge \\
& t[VT_E] = t'[VT_E] \wedge t[TT_S] = t'[TT_S] \wedge t[TT_E] = t'[TT_E]\}.
\end{aligned}
$$

The definition of selection on non-temporal attributes is trivial: only the input tuples whose non-temporal component satisfy the selection predicate ϕ are reported in output, unchanged (both in their temporal and nontemporal parts). We observe that ϕ is a predicate regarding non-temporal attributes only.

Definition 4 (Nontemporal selection σ^N_ϕ) Given a bitemporal relation r in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$, and a predicate ϕ regarding the non-temporal attributes only, $\sigma^N_\phi(r)$ is a bitemporal relation q defined over the schema R^N, and is defined as follows:

$$\sigma^N_\phi(r) = \{t \in r \mid \phi(t[A_1, \ldots, A_n])\}.$$

Intuitively speaking, in the temporal difference $r -^N s$, each tuple $t' \in r$ which has no value-equivalent tuple in s is simply reported unchanged in output. Otherwise, let $\{t_1, \ldots, t_k\}$ the set of all and only the tuples in s that are value-equivalent to t'. The time of the resulting tuple is obtained by removing from the time of t' the (union of the) times of $t_1, \ldots, t_k$. Of course, if the result of such a removal is empty, the tuple is not part of the output. Although the basic idea is simple, its realization is technically complex, since:

- in the *MIN*, *MAX*, *NULL* and *POINT* representations, times are represented through pairs of periods (start-end of the valid time, start-end of the transaction time), i.e., as rectangles in the bi-dimensional space [24], [13];
- additionally, in the *POINT* approach, points and lines in the bitemporal space are used to model now-relative data.

In the definition below, which, notably, is quite similar to the definition of difference in *BCDM*, the function $EXT^{c_{NOW}}$ (see Section 3) generates all the bitemporal chronons corresponding to the *MIN*, *MAX*, *NULL* or *POINT* representations. The (union of the) bitemporal chronons of $t_1, \ldots, t_k$ are subtracted from the bitemporal chronons of t'. Finally, the function *cover* reconverts the resulting set of bitemporal chronons into the *MIN*, *MAX*, *NULL*

or *POINT* representations. Specifically, given a set S of bitemporal chronons, $cover(S)$ provides in output a set of rectangles, in form of quadruples $(TT_S,\ TT_E,\ VT_S,\ VT_E)$, in the *MIN*, *MAX*, *NULL* or *POINT* representations, covering all and only the chronons in S. For each one of such rectangles, a tuple value-equivalent to t' (having such a rectangle as its temporal component) is reported in output. Obviously, if the difference of bitemporal chronons is empty, *cover* is applied to the empty set, and no tuple corresponding to t' is reported.

Definition 5 (Temporal difference $-^N$) Given two bitemporal relations r and s in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the same schema $R^N = (A_1,\ \ldots,\ A_n \mid TT_S,\ TT_E,\ VT_S,\ VT_E)$, and given any value c_{NOW} for *now*, the temporal difference $r -^N s$ is a bitemporal relation q defined over the schema R^N defined as follows:

$$
\begin{aligned}
r -^N s \ = \ \{t \in\ & D_{A_1} \times \ldots \times D_{A_n} \times D_{TT} \times D_{TT} \times D_{VT} \times D_{VT} \mid \\
& (\exists\, t' \in r,\ t[A_1,\ \ldots,\ A_n] = t'[A_1,\ \ldots,\ A_n] \wedge t[TT_S] = t'[TT_S] \wedge t[TT_E] = t'[TT_E] \wedge \\
& t[VT_S] = t'[VT_S] \wedge t[VT_E] = t'[VT_E] \wedge \neg\, \exists\, t' \in s,\ t[A_1,\ \ldots,\ A_n] = t'[A_1,\ \ldots,\ A_n]) \vee \\
& (\exists\, t' \in r,\ \ \exists\, t_1,\ \ldots,\ t_k \in s,\ \forall\, i\ 1 \leq i \leq k\ t_i[A_1,\ \ldots,\ A_n] = t'[A_1,\ \ldots,\ A_n] \wedge \\
& \forall\, u \in s\ u[A_1,\ \ldots,\ A_n] = t'[A_1,\ \ldots,\ A_n] \Rightarrow (u = t_1 \vee \ \ldots\ \vee\ u = t_k) \wedge \\
& t[A_1,\ \ldots,\ A_n] = t'[A_1,\ \ldots,\ A_n] = t_1[A_1,\ \ldots,\ A_n] = \ \ldots\ = t_k[A_1,\ \ldots,\ A_n] \wedge \\
& t[TT_S,\ TT_E,\ VT_S,\ VT_E] \in cover(EXT^{c_{NOW}}(t') - (EXT^{c_{NOW}}(t_1) \cup \ \ldots \\
& \cup\ EXT^{c_{NOW}}(t_k))))\}.
\end{aligned}
$$

Of course, many different implementations of the $cover(S)$ function are possible, depending on the policy used in order to generate the covering rectangles.

Actually, the above definition of difference has been provided to enhance clarity, and to stress our adherence to the reference *BCDM* model. On the other hand, for the sake of computational efficiency, the translation from *MIN*, *MAX*, *NULL* or *POINT* representations to bitemporal chronons and back, as well the application of unions and difference to bitemporal chronons, are not strictly necessary. As a matter of fact, one can more efficiently devise an algorithm that for each one of the *MIN*, *MAX*, *NULL* and *POINT* representations, taken in input a rectangle r and a set of rectangles $\{r_1,\ \ldots,\ r_k\}$, directly provides in output a set of rectangles covering the whole difference (or the empty set if the difference is empty). This is the strategy we adopt in this paper, proposing the function $difference$ as a sample implementation of the above computation.

4.1.1 A covering difference algorithm for the MIN, MAX, NULL and POINT approaches

The algorithm introduced in this section is an extension of the generic algorithm for computing the difference between sets of rectangles described in [9] to consider also rectangles bounded by *now*.

The input of the algorithm (see Function 6) is a rectangle represented by the quadruple (TT_S, TT_E, VT_S, VT_E) and a set of rectangles (each represented by a quadruple) to be subtracted from it. The variables r, s and *result* in the algorithm denote sets of rectangles. r is initialized to contain (TT_S, TT_E, VT_S, VT_E), s to the set of rectangles to be subtracted and *result* to the empty set. In each iteration of the outer loop, a rectangle of set r is chosen. Then, for each rectangle in s that intersects (identified by Function 7) with the chosen rectangle, we compute the difference (Function 6), determining the left, right, bottom and top rectangles resulting from the operation, if they exist (see Figure 1).

As for the Cartesian product, the operation is defined by cases. For instance, regarding the right rectangle, the first case (line 11) states that, if the transaction times of the element of r and of the element of s are not now-relative, the right rectangle exists if the transaction

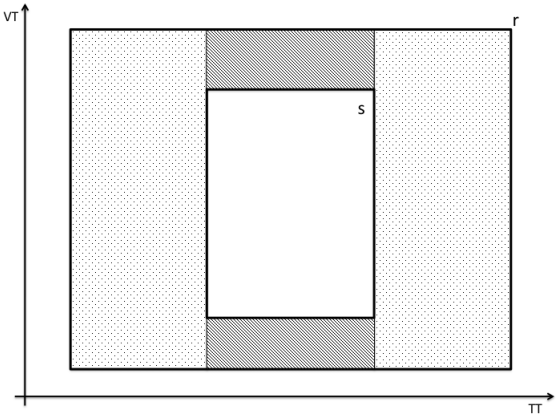

Fig. 1 Graphical representation of the difference operation between rectangles r (the biggest rectangle) and s (the smallest rectangle) with the four possible resulting rectangles (left, right, top, bottom). Some of such rectangles could be missing, depending on the relative location of r and s.

time of the minuend r ends after the transaction time of the subtrahend s (i.e., $TT_E^r > TT_E^s$). The transaction time of the right rectangle starts at TT_E^s and ends at TT_E^r and its valid time starts at VT_S^r and ends at VT_E^r.

On the other hand, if the transaction time of r is now-relative and the transaction time of s is not (line 12), because of the semantics of transaction time, the end of transaction time of r (i.e., *now*) is certainly after the end of the transaction time of s and the transaction time of the resulting rectangle ends at *now*.

The cases where r is not now-relative and s is now-relative and where both r and s are now-relative give no right rectangle because the end of the transaction time of s is certainly not before the end of the transaction time of r.

The bottom and top rectangles are computed in a similar way with a notable difference: the end of the valid time can be after *now*. This fact impacts line 16, where the valid time of r is not now-relative and the valid time of s is now-relative: differently from the analogous case in line 12, we must test whether the end of valid time of r is not after *now* (c_{NOW} is incremented because the valid time period is open to the right). Moreover, if the valid time of r is now-relative and the valid time of s is not, the top rectangle exists if the end of valid time of s is not after *now*.

It is worth noting that in the cases in lines 12 and 16 the result is also now-relative. However, no approach admits now-relative time periods where *now* is the starting point of the period. Therefore, in the case where we would obtain *now* as the starting point (see line 16), we rather set, as starting time, the current chronon, i.e., c_{NOW}.

The resulting rectangle is then added to r in such a way that in subsequent iterations it will be checked against the other elements of s.

A theoretical issue here concerns the minimality of the resulting covering set of rectangles. If the set is minimal, a minimal number of value-equivalent tuples is provided in the result of the difference operator. Such a minimality could be directly provided by the difference algorithm or could be obtained in a second step by applying transformations (like coalescing) that preserve snapshot equivalence [24]. However, from one side computing minimal covering might lead to the introduction of overlapping rectangles in the output (which amounts, roughly speaking, to introducing redundant information) and, from the

Function 6 $difference((TT_S, TT_E, VT_S, VT_E), \{(TT_S^1, TT_E^1, VT_S^1, VT_E^1), \ldots, (TT_S^k, TT_E^k, VT_S^k, VT_E^k)\})$

1: $r \leftarrow \{(TT_S, TT_E, VT_S, VT_E)\}$
2: $s \leftarrow \{(TT_S^1, TT_E^1, VT_S^1, VT_E^1), \ldots, (TT_S^k, TT_E^k, VT_S^k, VT_E^k)\}$
3: $result \leftarrow \emptyset$
4: **while** $r \neq \emptyset$ **do**
5: choose a rectangle $(TT_S^r, TT_E^r, VT_S^r, VT_E^r) \in r$
6: $changed \leftarrow$ **false**
7: **for all** $(TT_S^s, TT_E^s, VT_S^s, \mathrm{V}T_E^s) \in$ s **do**
8: **if** $intersects((TT_S^r, TT_E^r, VT_S^r, VT_E^r), (TT_S^s, TT_E^s, VT_S^s, VT_E^s))$ **then**
9: $r \leftarrow r - \{(TT_S^r, TT_E^r, VT_S^r, VT_E^r)\}$
10: **if** $TT_S^r < TT_S^s$ **then** $r \leftarrow r \cup \{(TT_S^r, TT_S^s, VT_S^r, VT_E^r)\}$ /* left rectangle */
11: **if** $\neg\, isNowRelative(TT_S^r, TT_E^r) \wedge \neg\, isNowRelative(TT_S^s, TT_E^s) \wedge TT_E^r > TT_E^s$ **then** $r \leftarrow r \cup \{(TT_E^s, TT_E^r, VT_S^r, VT_E^r)\}$ /* not now-relative right rectangle */
12: **if** $isNowRelative(TT_S^r, TT_E^r) \wedge \neg\, isNowRelative(TT_S^s, TT_E^s)$ **then** $r \leftarrow$ r $\cup$ $\{(TT_E^s, setNow(TT_E^s), VT_S^r, VT_E^r)\}$ /* now-relative right rectangle */
13: **if** $VT_S^r < VT_S^s$ **then** $r \leftarrow r \cup \{(TT_S^s, TT_E^s, VT_S^r, VT_S^s)\}$ /*bottom rectangle */
14: **if** $\neg\, isNowRelative(VT_S^r, VT_E^r) \wedge \neg\, isNowRelative(VT_S^s, VT_E^s) \wedge VT_E^r > VT_E^s$ **then** $r \leftarrow r \cup \{(TT_S^s, TT_E^s, VT_E^s, VT_E^r)\}$ /* not now-relative top rectangle */
15: **if** $isNowRelative(VT_S^r, VT_E^r) \wedge \neg\, isNowRelative(VT_S^s, VT_E^s) \wedge VT_E^s \leq c_{NOW}$ **then** $r \leftarrow r \cup \{(TT_S^s, TT_E^s, VT_E^s, setNow(VT_E^s))\}$ /*now-relative top rectangle*/
16: **if** $\neg\, isNowRelative(VT_S^r, VT_E^r) \wedge isNowRelative(VT_S^s, VT_E^s) \wedge VT_E^r > c_{NOW} + 1$ **then** $r \leftarrow r \cup \{(TT_S^s, TT_E^s, c_{NOW} + 1, VT_E^r)\}$ /*now-relative top rectangle*/
17: $changed \leftarrow$ **true**
18: **end if**
19: **end for**
20: **if** $changed$ = **false then**
$r \leftarrow r - \{(TT_S^r, TT_E^r, VT_S^r, VT_E^r)\}$
$result \leftarrow result \cup \{(TT_S^r, TT_E^r, VT_S^r, VT_E^r)\}$
21: **end if**
22: **end while**
23: **return** *result*

Function 7 $intersects((TT_S^r, TT_E^r, VT_S^r, VT_E^r), (TT_S^s, TT_E^s, VT_S^s, VT_E^s))$

1: **if** $isNowRelative(TT_S^r, TT_E^r)$ **then** $tt_E^r \leftarrow c_{NOW} + 1$ **else** $tt_E^r \leftarrow TT_E^r$
2: **if** $isNowRelative(TT_S^s, TT_E^s)$ **then** $tt_E^s \leftarrow c_{NOW} + 1$ **else** $tt_E^s \leftarrow TT_E^s$
3: **if** $isNowRelative(VT_S^r, VT_E^r)$ **then** $vt_E^r \leftarrow c_{NOW} + 1$ **else** $vt_E^r \leftarrow VT_E^r$
4: **if** $isNowRelative(VT_S^s, VT_E^s)$ **then** $vt_E^s \leftarrow c_{NOW} + 1$ **else** $vt_E^s \leftarrow VT_E^s$
5: **if** $TT_S^r \geq tt_E^s \vee TT_S^s \geq tt_E^r \vee VT_S^r \geq vt_E^s \vee VT_S^s \geq vt_E^r$ **return false else return true**

other side, it would require additional computation. For such a reason, we have chosen to prefer the efficiency to the minimality (for analogous reasons in TSQL2 coalescing is not performed by default in the algebraic operators, but only “on-demand”).

4.1.2 Temporal selection operators

Besides Codd's basic operators, many operators were provided in order to query the temporal component of temporal tuples. (consider, e.g., the TSQL2 "consensus" approach [24]). In this paper we just focus on *temporal selection*.

Definition 6 (Temporal selection $\sigma^N_{\phi t}$) Given a bitemporal relation r in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$ and a temporal predicate ϕ_t regarding the temporal attributes only, $\sigma^N_{\phi_t}(r)$ is a bitemporal relation q defined over the schema R^N, and is defined as follows:

$$\sigma^N_{\phi_t}(r) = \{t \in r \mid \phi_t(t[TT_S, TT_E, VT_S, VT_E])\}$$

In other words, all and only the tuples in r^N whose temporal component satisfy the selection predicate ϕ_t are reported in output. In particular, *range queries* have been proven to play a major role within TDBs [14]. Range queries have been already defined for the *POINT* approach [25].

4.1.3 Properties of the MIN, MAX, NULL and POINT algebrae

Both closure and correctness with respect to *BCDM* hold for the *MIN*, *MAX*, *NULL* and *POINT* algebrae.

Property 1 (Closure) The temporal relational algebra is closed, in the sense that any algebraic operator, taking in input relations in the *MIN*, *MAX*, *NULL* or *POINT* representation, give in output relations in the *MIN*, *MAX*, *NULL* and *POINT* representation respectively.

Property 2 (Correctness of the MIN, MAX, NULL and POINT algebrae) The *MIN*, *MAX*, *NULL* and *POINT* algebrae are correct, since they provide (in the *MIN*, *MAX*, *NULL* and *POINT* representations) all and only the results that would be obtained by the underlying *BCDM* semantic approach.

An important issue concerning a temporal algebra lies in its compatibility and reducibility to the standard (non-temporal) relational model, in order to grant that, if time is disregarded, the extended temporal model behaves like the standard model. These properties are important for temporal approaches, since they grant interoperability with pre-existent non-temporal approaches. For defining these properties, we introduce transaction- and valid timeslice operators. For instance, the transaction-timeslice operator takes in input a bitemporal relation r and a time t and gives as output a valid-time relation r', containing all and only the tuples in r whose transaction time contains t. Only the valid time of such tuples is retained in r'. The valid timeslice operator is analogous: if any tuple of the bitemporal relation has a valid time which includes the time value given in input, the relation given in output contains the explicit values of the tuple and its transaction time.

The transaction-timeslice operator for transaction-time relations and valid timeslice operator for valid time relations are straightforward special cases.

Definition 7 (Transaction timeslice) Let r a bitemporal relation in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$ and c_T an arbitrary time value not exceeding current time:

$$\rho^N_{c_T}(r) = \{t \in D_{A_1} \times \ldots \times D_{A_n} \times D_{VT} \times D_{VT} \mid \exists\, t' \in r,\ t[A] = t'[A] \wedge$$
$$t[VT_S] = t'[VT_S] \wedge t[VT_E] = t'[VT_E] \wedge c_t \in EXT^{c_{NOW}}(t'[TT_S], t'[TT_E])\}.$$

The result of transaction-timeslice operator is a valid-time relation, defined over the schema $R_V^N = (A_1, \ldots, A_n \mid VT_S, VT_E)$.

Definition 8 (Valid timeslice) Let r a bitemporal relation in the *MIN*, *MAX*, *NULL* or *POINT* approaches, defined over the schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$ and c_V an arbitrary time value:

$$\tau_{c_V}^N(r) = \{t \in D_{A_1} \times \ldots \times D_{A_n} \times D_{TT} \times D_{TT} \mid \exists\, t' \in r, t[A] = t'[A] \wedge t[TT_S] = t'[TT_S] \wedge t[TT_E] = t'[TT_E] \wedge c_v \in EXT^{c_{NOW}}(t'[VT_S], t'[VT_E])\}.$$

The result of valid timeslice operator is a transaction-time relation, defined over the schema $R_T^N = (A_1, \ldots, A_n \mid TT_S, TT_E)$.

We observe that the combined application to a bitemporal relation r of a transaction and a valid timeslice operators at times t_{TT} and t_{VT} respectively provides as output a standard (non-temporal) relation r', consisting of (the non-temporal part of) all and only those tuples in r which hold at the bi-temporal chronon (t_{TT}, t_{VT}). Informally, r' is the standard relation capturing the single *snapshot* (t_{TT}, t_{VT}) of the bitemporal relation r.

Property 3 (Reducibility of MIN, MAX, NULL and POINT algebrae to the standard (non-temporal) algebra) The *MIN*, *MAX*, *NULL* and *POINT* algebrae reduce to the standard algebra, i.e., the non-temporal relation obtained by applying extended temporal operators to temporal relations and then taking a snapshot is equivalent to the standard (non-temporal) relation obtained by first taking a snapshot of the temporal relations and then applying a standard (non-temporal) operator.

More formally, let r^N and s^N be bitemporal relations defined over a schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$, c_T an arbitrary time value not exceeding current time and c_V an arbitrary time value, $\rho_{c_T}(\tau_{c_V}(r^N\ Op^N\ s^N)) = \rho_{c_T}(\tau_{c_V}(r^N))\ Op^S\ \rho_{c_T}(\tau_{c_V}(s^N))$, where Op^N is a temporal operator and Op^S is a standard (non-temporal) operator. The statement is analogous for unary operators.

Definition 9 (Temporal transform) Let r be a standard (non-temporal) relation, defined over the schema $R_1 = (A_1, \ldots, A_n)$ and *tt_start*, *tt_end*, *vt_start*, *vt_end* timestamps, $transform(r, tt\_start, tt\_end, vt\_start, vt\_end) = \{t \in D_{A_1} \times \ldots \times D_{A_n} \times D_{TT} \times D_{TT} \times D_{VT} \times D_{VT} \mid \exists t' \in r \wedge t[T] = (tt\_start, tt\_end, vt\_start, vt\_end)\}$.

The result of temporal-transform operator is a bitemporal relation, defined over the schema $R_2 = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$.

The following property grants that those queries that are possible when time is not represented, are also possible when time is added.

Property 4 (Consistent extension) The temporal algebrae are consistent extensions of the standard (non-temporal) algebra, i.e., the standard relational operators have a counterpart in the temporal relational operators.

Finally, it is worth comparing the computational complexity of our algebraic operators with that of temporal operators already in the literature, e.g., with TSQL2's ones. Notably, our definitions of temporal union, temporal projection, nontemporal selection and temporal selection are equivalent to TSQL2's ones. As regards temporal difference, our operator is similar to TSQL2's operator since the difference between a bitemporal rectangle and a set of bitemporal rectangles is computed by both operators. The only notable difference is that,

in our approach, such a computation involves checking whether valid and transaction times are now-relative (see the calls to the *isNowRelative* function in Function 6). Analogously, while TSQL2's Cartesian product operator simply evaluates the intersection along the VT and TT dimensions, in our approach such a computation involves checking whether such values are now-relative and, in such a case, invoking the *setNow* function. As a consequence, only a constant time is added with respect to the TSQL2 operators, in the Cartesian product and difference operators. In the next Section, we will show experimentally that such an additional time is almost negligible. To do so, however, we first have to introduce a suitable comparison term, the $NOT-NOW$ approach.

4.2 $NOT-NOW$ approach

To the best of our knowledge, there are no temporal relational algebrae in the literature supporting the treatment of now-relative data without resorting to Variable databases. Therefore, to provide a "reference" comparison term for our MIN, MAX, $NULL$ and $POINT$ algebrae, in this section we provide a third approach, that we term "$NOT-NOW$". The $NOT-NOW$ approach is the simplest approach to cope with now-relative data. It assumes that the future is known, so that the ending times (of both valid and transaction times) of all tuples are known timestamps. Of course, in such a database there is no way to state, e.g., Example 1. One has to state, e.g., that the transaction time of the tuple starts at 11 and ends at a specific date. For the $NOT-NOW$ approach, a standard temporal algebra such as the one of TSQL2, can be used. $NOT-NOW$ is not feasible in practice because one cannot know the future. However, by comparing the MIN, MAX, $NULL$ and $POINT$ approaches to the $NOT-NOW$ one, we can show what the additional cost is of coping with now-relative data in such approaches.

5 Experimental evaluation

Since the MAX approach outperforms the $NULL$ and MIN approaches [31], in this section we empirically compare the performances of the MAX, the $POINT$ and the $NOT-NOW$ approaches. We generated the same data set for each approach and then on each relation we performed Temporal Cartesian Product, Temporal Difference, and Temporal Selection (e.g., range queries).

5.1 Data sets

We have randomly generated data sets with different data distributions to simulate a real-world scenario. For each approach we generated three tables differing in the percentage of now-relative data. As suggested in the literature [30], [27] for each approach we considered three datasets with 10%, 20% and 40% of now-relative data. The structures of all tables are identical: besides the ID and $Dept$ attributes, there are four temporal attributes VT_s, VT_e, TT_s, and TT_e. Each table contains one million tuples. However, for Temporal Cartesian Product and Temporal Difference, due to the answer size, we considered only one thousand rows in each table. For the *NOT-NOW* approach we have replaced now-relative valid time and/or transaction time ends with randomly generated timestamps in the future.

Approach	Now-relative data ratio	Disk accesses	CPU usage (10s of ms)	Resp.time (min:sec)	Logical reads	Answ.size (numb. of tuples)
MAX	10%	16	10,521	1:46.325	1,132,103	104,501
POINT	10%	16	10,338	1:44.393	1,128,668	104,501
$NOT-NOW$	10%	16	9,113	1:31.672	1,121,456	101,968
MAX	20%	16	12,679	2:13.714	1,240,591	228,466
POINT	20%	16	12,377	2:10.687	1,236,454	228,466
$NOT-NOW$	20%	16	11,352	2:01.404	1,228,571	226,481
MAX	40%	16	15,990	2:55.695	1,439,416	408,485
POINT	40%	16	15,547	2:51.133	1,429,377	408,485
$NOT-NOW$	20%	16	14,801	2:37.906	1,418,882	400,766

Table 1 Temporal Cartesian Product

The starting times of the periods were always uniformly distributed on the time domain, while duration and percentage of now-relative data were varied. While we have chosen a uniform distribution of period starting times, we adopted an exponential distribution of the durations because it reflects most real-world applications where short periods are more likely to occur than long periods [8].

5.2 Environment

Our implementation has been carried out on a 8 x UltraSparc III @ 900Hz with 8GB of RAM memory, running Oracle 11 RDBMS, with a database block size of 8K and size of SGA (System Global Area) of 1000MB. The SGA was locked into memory to ensure that paging does not affect results. To ensure that the logical read of data already in SGA does not influence the results we flushed the database buffer cache in SGA before every particular test. At the time of testing the database server did not have any other significant load. We used Oracle built-in methods for statistics collection, analytic SQL functions and the PL/SQL procedural runtime environment. For range queries we utilised the TD-tree indexing method [26], since it has been shown that the TD-tree has the best performance, considering the physical disk I/O and the query response time and at the same time can be employed within the commercial RDBMS [28].

5.3 Results and analysis

Both physical disk I/Os and CPU usage are taken into account in our analysis. We recall that, according to the analysis in [10], physical disk I/Os is the most important parameter. However, considering the relatively small number of records in tables used for evaluations of temporal Cartesian product and temporal difference and, since in such cases a full table scan needs to be performed, CPU usage and response time are useful indicators.

As we can see in Table 1, since Cartesian product requires a full table scan, disk accesses are the same in all approaches. Also as regards CPU usage and response time, the *POINT*, *MAX* and *NOT-NOW* approaches provide similar results. Actually, the *NOT-NOW* approach, which is the optimal one (since it indeed does not require any extension to cope with *now*), requires slightly less CPU usage for any percentage of now-relative data and

Approach	Now-relative data ratio	Disk accesses	CPU usage (10s of ms)	Resp.time (min:sec)	Logical reads	Answ.size (numb. of tuples)
MAX	10%	16	88	0:01.591	4,744	1,007
POINT	10%	16	87	0:01.523	4,742	1,007
NOT − NOW	10%	16	84	0:01.482	4,740	1,007
MAX	20%	16	87	0:01.972	4,736	1,007
POINT	20%	16	86	0:01.875	4,734	1,007
NOT − NOW	20%	16	85	0:01.775	4,730	1,007
MAX	40%	16	89	0:01.749	4,734	1,007
POINT	40%	16	87	0:01.716	4,734	1,007
NOT − NOW	20%	16	84	0:01.664	4,732	1,007

Table 2 Temporal Difference Product

therefore has the fastest response time. However, this is also due to the fact that, in the *NOT-NOW* approach, the answer size is slightly smaller than in the other approaches: since the now-relative data are represented with the random future timestamps, less tuples satisfy the query criteria (i.e., intersection of bitemporal times). Also, the *POINT* approach is slightly better than the *MAX* approach with regard to CPU usage and has a faster response time (due to the less expensive algorithm of temporal Cartesian product). This difference is increasing as the percentage of now-relative data increases, as a result of the increase of the answer size.

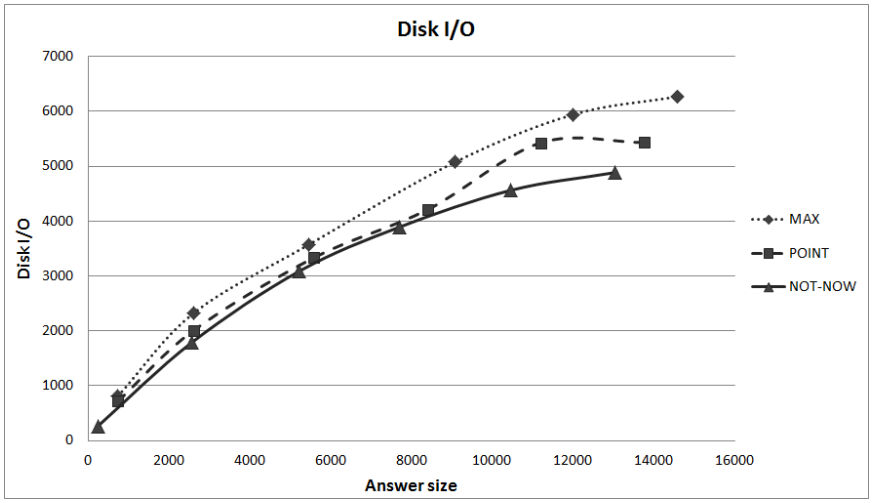

Fig. 2 Range Query - Physical Disk I/O

Table 2 shows the results concerning temporal difference. Notably, there are only slight differences in CPU usage and response time. Due to the nature of temporal difference (and the possible dimension of the answer size), we had to consider temporal difference between tables with only one thousand rows. Since difference requires a full table scan to consider all the records, physical disk I/O is constant. As it can be seen, both *MAX* and *POINT*

approaches are only slightly worse with regard to the CPU usage and response time than the optimal *NOT-NOW* approach.

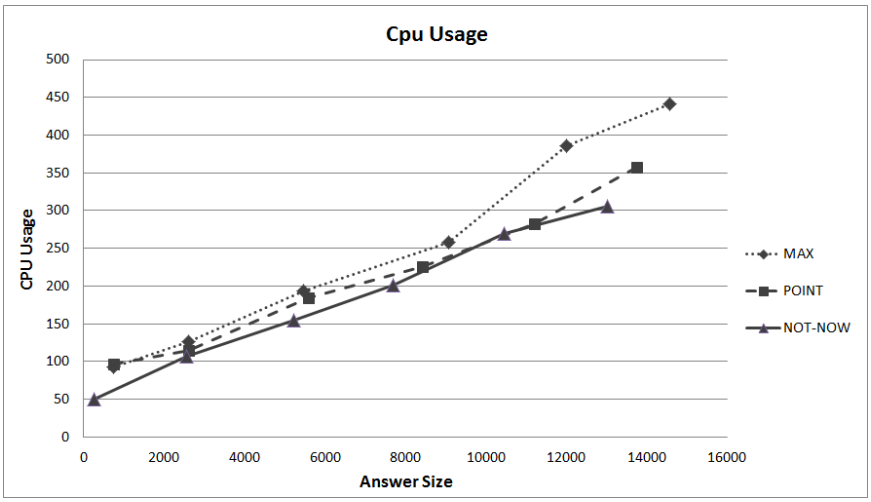

Fig. 3 Range Query - CPU Usage

In Figures 2, 3, and 4 we show the results for physical disk I/O, CPU usage, and query response time for range queries as a factor of the answer size. In these experiments, we have considered 20% of now-relative data. However, the results obtained with 10% and 40% now-relative data are similar. Once again, although the "ideal" *NOT-NOW* approach performs slightly better that the *POINT* and *MAX* approaches, differences are minimal.

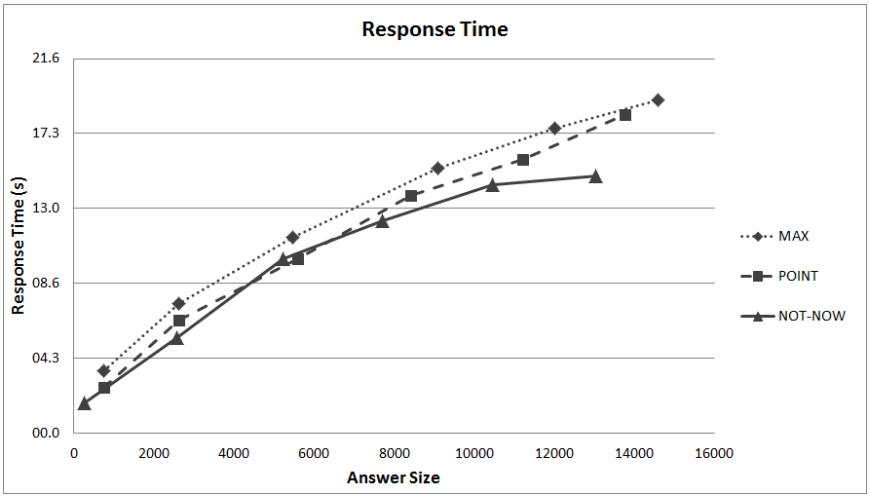

Fig. 4 Range Query - Response Time

6 Conclusions

Now-relative temporal data play an important role in most temporal applications. As a consequence, the treatment of such data has attracted a significant amount of attention in the (temporal) database literature.

Early approaches cope with now resorting to database variable [16]. Recently, the *MIN*, *MAX*, *NULL* and *POINT* approaches made it possibile to avoid the use of variables making easier a direct implementation on relational databases. Moreover, none of these approaches have provided a full algebraic language to query data.

In this paper we overcome such a limitation of the current literature. We focus on the purely relational approaches (thus neglecting variable databases), and provide a general temporal relational algebra which can act as the query language for the *MIN*, *MAX*, *NULL* and *POINT* approaches.

Besides generality, our algebra meets also several other theoretical and practical desiderata: *closure* with respect to representation languages, *correctness* with respect to the "consensus" *BCDM* semantics, *reducibility* to the standard non-temporal algebra, *implementability* and *efficiency*. Indeed, the experimental evaluation we have drawn on our implementation has shown that only a slight overhead is added by our treatment of now-relative data (with respect to an "ideal" but unfeasible approach in which such data are not included since future is known).

References

1. M. Agesen, M. Bohlen, L. Poulsen, and K. Torp. A split operator for now-relative bitemporal databases. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 41 –50, 2001.
2. M. H. Bohlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. *Proc. of the 22nd VLDB Conf*, pages 180–190, 1996.
3. J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the semantics of "Now" in databases. *ACM Transactions on Database Systems (TODS)*, 22(2):171–214, 1997.
4. E. F. Codd. Relational completeness of data base sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California*, 1972.
5. K. N. Creem. A comparison of approaches to modeling now in bitemporal databases. In *21 st Computer Science Seminar. Hartford. USA*, 2005.
6. C. E. Dyreson. Temporal coalescing with now granularity, and incomplete information. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 169–180, New York, NY, USA, 2003. ACM.
7. C. E. Dyreson, C. S. Jensen, and R. T. Snodgrass. Now in temporal databases. In L. LIU and M. ZSU, editors, *Encyclopedia of Database Systems*, pages 1920–1924. Springer US, 2009.
8. R. Fenk, V. Markl, and R. Bayer. Interval Processing with the UB-Tree. In *Proceedings of the 2002 International Symposium on Database Engineering and Applications*, pages 12–22, 2002.
9. D. S. Franzblau and G. Xenakis. An algorithm for the difference between set covers. *Discrete Appl. Math.*, 156(10):1623–1632, May 2008.

10. J. Hellerstein, E. Koutsupias, and C. Papadimitriou. On the Analysis of Indexing Schemes. *16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1997.
11. C. S. Jensen and D. B. Lomet. Transaction timestamping in (temporal) databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 441–450, 2001.
12. C. S. Jensen and R. Snodgrass. Temporal Data Management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):36–44, 1999.
13. C. S. Jensen and R. T. Snodgrass. Semantics of Time-Varying Information. *Information Systems*, 21(4):311–352, 1996.
14. H. Kriegel, M. Ptke, and T. Seidl. Managing intervals efficiently in object-relational databases. *Proceedings of the 26th International Conference on Very Large Databases*, pages 407–418, 2000.
15. J. L. Edwin McKenzie and R. T. Snodgrass. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys (CSUR)*, 23(4):501–543, 1991.
16. L. Liu and M. T. Özsu, editors. *Encyclopedia of Database Systems*. Springer US, 2009.
17. D. Lomet, M. Hong, R. Nehme, and R. Zhang. Transaction time indexing with version compression. *Proceedings of the VLDB Endowment*, 1(1):870–881, 2008.
18. C. Mao, H. Ma, Y. Tang, and L. Yao. Temporal data model and temporal database systems. In Y. Tang, X. Ye, and N. Tang, editors, *Temporal Information Processing Technology and Its Application*, pages 69–89. Springer Berlin Heidelberg, 2011.
19. J. Melton and A. R. Simon. *SQL:1999 - Understanding Relational Language Components*. Morgan Kaufman, 2002.
20. L.-V. Nguyen-Dinh, W. G. Aref, and M. F. Mokbel. Spatio-temporal access methods: Part 2 (2003 - 2010). *IEEE Data Eng. Bull.*, 33(2):46–55, 2010.
21. G. Ozsoyoglu and R. Snodgrass. Temporal and real-time databases: A survey. *IEEE Trans. On Knowledge and Data Engineering*, 7(4):513–532, 1995.
22. S. Šaltenis and C. S. Jensen. Indexing of now-relative spatio-bitemporal data. *The VLDB journal*, 11(1):1–16, 2002.
23. C. Z. Shichao Zhang and Z. Qin. Modeling temporal semantics of data. *Asian Journal of Information Technology*, 2(1):25–35, 2003.
24. R. T. Snodgrass. *The TSQL2 Temporal Query Language*. Kluwer Academic, 1995.
25. B. Stantic, A. Sattar, and P. Terenziani. The point approach to represent it now in bitemporal databases. *J. Intell. Inf. Syst.*, 32(3):297–323, 2009.
26. B. Stantic, J. Terry, R. W. Topor, and A. Sattar. Indexing Temporal Data with Virtual Structure. In *Advances in Databases and Information Systems - ADBIS*, pages 591–594, 2010.
27. B. Stantic, J. Thornton, and A. Sattar. A Novel Approach to Model NOW in Temporal Databases. *In Proceeding of the 10th International Symposium on Temporal Representation and Reasoning (TIME-ICTL 2003), Cairns*, pages 174–181, 2003.
28. B. Stantic, R. W. Topor, J. Terry, and A. Sattar. Advanced indexing technique for temporal data. *Comput. Sci. Inf. Syst.*, 7(4):679–703, 2010.
29. A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal databases: theory, design, and implementation*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1993.
30. K. Torp, C. S. Jensen, and M. Bohlen. Layered implementation of temporal DBMS concepts and techniques. *A TimeCenter Technical Report TR-2*, 1999.

31. K. Torp, C. S. Jensen, and M. H. Böhlen. Layered temporal dbms: Concepts and techniques. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 371–380. World Scientific Press, 1997.
32. K. Torp, C. S. Jensen, and R. T. Snodgrass. Effective Timestamping in Databases. *VLDB Journal: Very Large Data Bases*, 8(3–4):267–288, 2000.
33. K. Torp, C. S. Jensen, and R. T. Snodgrass. Modification semantics in now-relative databases. *Inf. Syst.*, 29(8):653–683, Dec. 2004.
34. V. Tsotras and A. Kumar. Temporal database bibliography update. *ACM Sigmod Record*, 25(1):41–51, 1996.

A Appendix

Property 2 (Correctness of the MIN, MAX, NULL and POINT algebrae) The *MIN*, *MAX*, *NULL* and *POINT* algebrae are correct, since they provide (in the *MIN*, *MAX*, *NULL* and *POINT* representations) all and only the results that would be obtained by the underlying *BCDM* semantic approach.

Proof The correctness of the *MIN*, *MAX*, *NULL* and *POINT* algebrae is proved by showing that, given the relations r^N and s^N and any binary operator Op^N in the *MIN*, *MAX*, *NULL* or *POINT* approaches (the proof is analogous when considering unary operators), the application of the operator in *MIN*, *MAX*, *NULL* or *POINT* algebrae gives a result equivalent to the application of the corresponding operator in the *BCDM* algebra (indicated by Op^B):

$$to\_BCDM^{c_{NOW}}(r^N\ Op^N\ s^N) = to\_BCDM^{c_{NOW}}(r^N)\ Op^B\ to\_BCDM^{c_{NOW}}(s^N)$$

We prove the result for the operation of Cartesian product by proving the two inclusions, that is we prove that:

$$to\_BCDM^{c_{NOW}}(r^N \times^N s^N) \subseteq to\_BCDM^{c_{NOW}}(r^N) \times^B to\_BCDM^{c_{NOW}}(s^N)$$

and

$$to\_BCDM^{c_{NOW}}(r^N \times^N s^N) \supseteq to\_BCDM^{c_{NOW}}(r^N) \times^B to\_BCDM^{c_{NOW}}(s^N)$$

Let x be a *BCDM* tuple in $to\_BCDM^{c_{NOW}}(r^N \times^N s^N)$ and let $R = (A_1, \ldots, A_a, B_1, \ldots, B_b \mid T)$ be the schema of $to\_BCDM^{c_{NOW}}(r^N \times^N s^N)$. We denote as A the attributes $A_1, \ldots, A_a$ and as B the attributes $B_1, \ldots, B_b$. Then, by the definition of the $to\_BCDM^{c_{NOW}}$ function, there exists a maximal set of (possibly now-relative) tuples $\{y_0, y_1, \ldots, y_k\}$ in $r^N \times^N s^N$ ($k \geq 0$) such that $x[AB] = y_0[AB] = \ldots = y_k[AB]$ and that, considering the rectangles resulting from the extension of their timestamps (with $now = c_{NOW}$), their union corresponds to $x[T]$, i.e., $x[T] = EXT^{c_{NOW}}(y_0[T]) \cup \ldots \cup EXT^{c_{NOW}}(y_k[T])$. If such tuples $\{y_0, y_1, \ldots, y_k\}$ belongs to $r^N \times s^N$, by the definition of $\times^N$, for each $y_i \in \{y_0, y_1, \ldots, y_k\}$ there must be a tuple $w'_{i'} \in r^N$ and a tuple $w''_{i''} \in s^N$ such that $x[A] = y_i[A] = w'_{i'}[A]$, $x[B] = y_i[B] = w''_{i''}[B]$ and the bitemporal chronons of y_i correspond to the intersection of the bitemporal chronons of $w'_{i'}$ and $w''_{i''}$, i.e.:

$$EXT^{c_{NOW}}(y_i[T]) = EXT^{c_{NOW}}(w'_{i'}[T]) \cap EXT^{c_{NOW}}(w''_{i''}[T])$$

In the following, let $\{w'_1, \ldots, w'_l\}$ and $\{w''_1, \ldots, w''_m\}$ the sets of all and only the tuples in r^N and in s^N respectively that generate the above tuples $\{y_0, y_1, \ldots, y_k\}$. Let us consider the tuples $w'_1, \ldots, w'_l$ in r^N. Applying the *to_BCDM* function, since these tuples have the same values for the explicit attributes, by definition of *to_BCDM*, *to_BCDM* returns one *BCDM* tuple $y \in to\_BCDM(r^N)$ such that:

$$y[A] = w'_1[A] = \ldots = w'_l[A]$$

and

$$y[T] = EXT(w'_1[T]) \cup \ldots \cup EXT(w'_l[T])$$

If we consider the tuples $w''_1, \ldots, w''_m$ in s^N, the function *to_BCDM* returns one *BCDM* tuple $z \in to\_BCDM(s^N)$ such that:

$$z[A] = w''_1[A] = \ldots = w''_m[A]$$

and

$$z[T] = EXT(w''_1[T]) \cup \ldots \cup EXT(w''_m[T])$$

Applying the $\times^B$ operator to y and z, by definition of the operator, we obtain a relation on schema $R(AB \mid T)$ containing a tuple $x' \in to\_BCDM^{c_{NOW}}(r^N) \times^B to\_BCDM^{c_{NOW}}(s^N)$ such that $x'[A] = y[A] = x[A]$ and $x'[B] = z[B] = x[B]$ and:

$$\text{x'[T]} = y[T] \cap z[T] = (EXT^{c_{NOW}}(w'_1[T]) \cup \ldots \cup EXT^{c_{NOW}}(w'_l[T])) \cap (EXT^{c_{NOW}}(w''_1[T]) \cup \ldots \cup EXT^{c_{NOW}}(w''_m[T]))$$

For the distributive property of intersection over union:

$$\begin{aligned}\text{x'[T]} = &(EXT^{c_{NOW}}(w'_1[T]) \cap EXT^{c_{NOW}}(w''_1[T])) \cup \ldots \cup \\ &(EXT^{c_{NOW}}(w'_1[T]) \cap EXT^{c_{NOW}}(w''_m[T])) \cup \ldots \cup \\ &(EXT^{c_{NOW}}(w'_l[T]) \cap EXT^{c_{NOW}}(w''_1[T])) \cup \\ &\ldots \cup (EXT^{c_{NOW}}(w'_l[T]) \cap EXT^{c_{NOW}}(w''_m[T]))\end{aligned}$$

Since by construction $\{w'_1, \ldots, w'_l\}$ and $\{w''_1, \ldots, w''_m\}$ are all and only the tuples that generates $\{y_0, y_1, \ldots, y_k\}$, i.e., such that for each i, $0 \leq i \leq k$, $x[A] = y_i[A] = w'_{i'}[A]$, $x[B] = y_i[B] = w''_{i''}[B]$ and $EXT^{c_{NOW}}(w'_{i'}) \cap EXT^{c_{NOW}}(w''_{i''}) = EXT^{c_{NOW}}(y_i[T])$, we have that: $x'[T] = EXT^{c_{NOW}}(y_1[T]) \cup \ldots \cup EXT^{c_{NOW}}(y_k[T]) = x[T]$. Therefore, $x = x'$ Now we prove the other direction of the inclusion.

Let x' be a *BCDM* tuple in $to\_BCDM^{c_{NOW}}(r^N) \times^B to\_BCDM^{c_{NOW}}(s^N)$. By definition of the $\times^B$ operator, there exist a tuple $y \in to\_BCDM^{c_{NOW}}(r^N)$ and a tuple $z \in to\_BCDM^{c_{NOW}}(s^N)$ such that $x'[A] = y[A]$, $x'[B] = z[B]$ and $x'[T] = y[T] \cap z[T] \neq \emptyset$.

By definition of *to_BCDM* function, if $y \in to\_BCDM^{c_{NOW}}(r^N)$ then there exists a (maximal) set of (possibly now-relative) tuples $\{w'_0, \ldots, w'_l\}$ that are in r^N such that $y[A] = w'_0[A] = \ldots = w'_l[A]$ and $y[T] = EXT^{c_{NOW}}(w'_0[T]) \cup \ldots \cup EXT^{c_{NOW}}(w'_l[T])$.

The same holds for z: there exists a (maximal) set of (possibly now-relative) tuples $\{w''_0, \ldots, w''_m\}$ that are in s^N such that $z[A] = w''_0[A] = \ldots = w''_m[A]$ and $z[T] = EXT^{c_{NOW}}(w''_0[T]) \cup \ldots \cup EXT^{c_{NOW}}(w''_m[T])$.

If we apply the $\times^N$ operator to r^N and s^N, because $\{w'_0, \ldots, w'_l\} \subseteq r^N$ and $\{w''_0, \ldots, w''_m\} \subseteq s^N$, we obtain, among the other tuples, a set of (possibly now-relative) tuples $\{y_1, \ldots, y_k\}$ such that $y_i[A] = w'_{i'}[A]$, $y_i[B] = w''_{i''}[B]$ and:

$$EXT^{c_{NOW}}(y_i[T]) = EXT^{c_{NOW}}(w'_{i'}[T]) \cap EXT^{c_{NOW}}(w''_{i''}[T])$$

Because $w'_{i'}[A] = y[A]$ and $w''_{i''}[B] = z[B]$, the *to_BCDM* function coalesces these tuples in one *BCDM* tuple x such that:

$$x[A] = y_0[A] = \ldots = y_k[A] = y[A], x[B] = y_0[B] = \ldots = y_k[B] = z[B]$$

and $x[T] = EXT^{c_{NOW}}(y_0[T]) \cup \ldots \cup EXT^{c_{NOW}}(y_k[T])$.

Since for each i, $0 \leq i \leq k$:

$$\begin{aligned}EXT^{c_{NOW}}(y_i[T]) = &EXT^{c_{NOW}}(w'_{i'}[T]) \cap EXT^{c_{NOW}}(w''_{i''}[T]), x[T] = EXT^{c_{NOW}}(y_0[T]) \cup \\ &\ldots \cup EXT^{c_{NOW}}(y_k[T]) = (EXT^{c_{NOW}}(w'_{i'_0}[T]) \cap EXT^{c_{NOW}}(w''_{i''_0}[T])) \\ &\cup \ldots \cup (EXT^{c_{NOW}}(w'_{i'_k}[T]) \cap EXT^{c_{NOW}}(w''_{i''_k}[T]))\end{aligned}$$

For the distributive property of union of sets over intersection of sets,

$$\begin{aligned}x[T] = &(EXT^{c_{NOW}}(w'_{i'_0}[T]) \cap EXT^{c_{NOW}}(w''_{i''_0}[T])) \cup \ldots \cup \\ &(EXT^{c_{NOW}}(w'_{i'_k}[T]) \cap EXT^{c_{NOW}}(w''_{i''_k}[T])) = (EXT^{c_{NOW}}(w'_1[T]) \cup \ldots \cup \\ &EXT^{c_{NOW}}(w'_l[T])) \cap (EXT^{c_{NOW}}(w''_1[T]) \cup \ldots \cup EXT^{c_{NOW}}(w''_m[T]))\end{aligned}$$

Since $y[T] = EXT^{c_{NOW}}(w'_0[T]) \cup \ldots \cup EXT^{c_{NOW}}(w'_l[T])$ and $z[T] = EXT^{c_{NOW}}(w''_0[T]) \cup \ldots \cup EXT^{c_{NOW}}(w''_m[T])$, we have that $x[T] = y[T] \cap z[T] = x'[T]$. Therefore, $x = x'$.

The proof for the other operators is similar.

Property 3 (Reducibility of MIN, MAX, NULL and POINT algebrae to the standard (non-temporal) algebra) The *MIN*, *MAX*, *NULL* and *POINT* algebrae reduce to the standard algebra, i.e., the non-temporal relation obtained by applying extended temporal operators to temporal relations and then taking a snapshot is equivalent to the standard (non-temporal) relation obtained by first taking a snapshot of the temporal relations and then applying a standard (non-temporal) operator.

More formally, let r^N and s^N be bitemporal relations defined over a schema $R^N = (A_1, \ldots, A_n \mid TT_S, TT_E, VT_S, VT_E)$, c_T an arbitrary time value not exceeding current time and c_V an arbitrary time value, $\rho_{c_T}(\tau_{c_V}(r^N\ Op^N\ s^N)) = \rho_{c_T}(\tau_{c_V}(r^N))\ Op^S\ \rho_{c_T}(\tau_{c_V}(s^N))$, where Op^N is a temporal operator and Op^S is a standard (non-temporal) operator. The statement is analogous for unary operators.

Proof We prove the property for the operator of Cartesian product. The proofs for the other operators are analogous.

The equivalence is proved by proving the two inclusions.

First we prove that $\rho_{c_T}(\tau_{c_V}(r^N \times^N s^N)) \subseteq \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$.

Let $R = (A_1, \ldots, A_a, B_1, \ldots, B_b \mid TT_S, TT_E, VT_S, VT_E)$ be the schema of $r^N \times^N s^N$. We denote as A the attributes $A_1, \ldots, A_a$ and as B the attributes $B_1, \ldots, B_b$. Let $x \in \rho_{c_T}(\tau_{c_V}(r^N \times^N s^N))$. Then, by definition of slice operators, there is a tuple $x' \in r^N \times s^N$ such that $x'[AB] = x[AB]$ and $(c_T, c_V) \in EXT^{c_{NOW}}((r^N \times^N s^N)[T])$.
Therefore, by definition of the $\times^N$, there exist a tuple $y \in r^N$ and a tuple $z \in s^N$ such that $y[A] = x'[A]$, $z[B] = x'[B]$, $(c_T, c_V) \in EXT^{c_{NOW}}(y[T])$ and $(c_T, c_V) \in EXT^{c_{NOW}}(z[T])$.
Therefore, by definition of slice operators, there exist a tuple $w' \in \rho_{c_T}(\tau_{c_V}(r^N))$ and a tuple $w'' \in \rho_{c_T}(\tau_{c_V}(s^N))$ such that $w' = y[A] = x'[A]$ and $w'' = z[B] = x'[B]$.
By definition of standard Cartesian product, there exists a tuple $x'' \in \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$ such that $x''[A] = w' = y[A] = x'[A]$ and $x''[B] = w'' = z[B] = x'[B]$.
Therefore, $x = x''$.

Now we prove that $\rho_{c_T}(\tau_{c_V}(r^N \times^N s^N)) \supseteq \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$.

Let us assume that $x'' \in \rho_{c_T}(\tau_{c_V}(r^N)) \times^S \rho_{c_T}(\tau_{c_V}(s^N))$. By definition of $\times^S$ operator, there exist a tuple $w' \in \rho_{c_T}(\tau_{c_V}(r^N))$ and a tuple $w'' \in \rho_{c_T}(\tau_{c_V}(s^N))$ such that $x''[A] = w'$ and $x''[B] = w''$. By definition of the slice operators, there exist a tuple $y \in r^N$ and a tuple $z \in s^N$ such that $y[A] = w'$, $z[B] = w''$, $(c_V, c_T) \in EXT^{c_{NOW}}(y[T])$ and $(c_V, c_T) \in EXT^{c_{NOW}}(z[T])$.

By definition of $\times^N$ operator, there exists a tuple $x' \in r^N \times^N s^N$ such that:

$$x'[A] = y[A], x'[B] = z[B] \text{ and } (c_V, c_T) \in EXT^{c_{NOW}}(x'[T])$$

By definition of slice operators, there exists a tuple $x \in \rho_{c_T}(\tau_{c_V}(r^N\ Op^N\ s^N))$ such that $x = x'[AB]$; therefore, since:

$$x[A] = x'[A] = y[A] = w' = x''[A] \text{ and } x[B] = x'[B] = z = w'' = x''[B]$$

we have that $x = x''$.

Property 4 (Consistent extension) The temporal algebrae are consistent extensions of the standard (non-temporal) algebra, i.e., the standard relational operators have a counterpart in the temporal relational operators.

Proof We have to prove that, let r^N and s^N be standard (non-temporal) relations defined over a schema: $R = (A_1, \ldots, A_n)$ and *tt_start*, *tt_end*, *vt_start*, *vt_end* timestamps,

$$transform(r\ Op\ s, tt\_start, tt\_end, vt\_start, vt\_end) =$$

$$transform(r, tt\_start, tt\_end, vt\_start, vt\_end)\ Op^N\ transform(s, tt\_start, tt\_end,$$

$$vt\_start, vt\_end)$$

where Op is a standard (non-temporal) operator and Op^N a temporal operator. The statement is analogous for unary operators.

We prove the two inclusions separately for the Cartesian product. The proofs for the other operators are analogous.

Let $x \in transform(r \times s, tt\_start, tt\_end, vt\_start, vt\_end)$. Then, by definition of *transform* function, $x[AB] \in r \times s$ and $x[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. By definition of $\times$, there exist a tuple $y \in r$ and a tuple $z \in s$ such that $x[A] = y$ and $x[B] = z$. Applying the *transform* function to y, we obtain a bitemporal tuple y' such that $y'[A] = y$ and $y'[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. The same holds for z: applying the *transform* function to z, we obtain a bitemporal tuple z' such that $z'[B] = z$ and

$z'[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. By definition of the $\times^N$ operator, there exists a bitemporal tuple x' such that $x'[A] = y'[A], x'[B] = z'[B]$ and:

$$x'[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$$

Then, $x = x'$.

Now we assume that $x' \in transform(r, tt\_start, tt\_end, vt\_start, vt\_end) \times^N transform(s, tt\_start, tt\_end, vt\_start, vt\_end)$. Then, by definition of $\times^N$, there exist a tuple $y' \in transform(r, tt\_start, tt\_end, vt\_start, vt\_end)$ and a tuple $z' \in transform(s, tt\_start, tt\_end, vt\_start, vt\_end)$ such that $x'[A] = y'[A]$ and $x'[B] = z'[B]$, and $z'[T] \cap y'[T] \neq \emptyset$. By definition of the $transform$ function, there exist a tuple $y \in r$ and a tuple $z \in s$ such that $y = y'[A]$ and $z = z'[A]$, and $y[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$ and $z[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$. By definition of the $\times$ operator, there exists a tuple $x'' \in r \times s$ such that $x''[A] = y, x''[B] = z$. By definition of the $transform$ function, there exists a tuple $x \in transform(r \times s, tt\_start, tt\_end, vt\_start, vt\_end)$ such that $x[A] = x''[A] = y'[A] = y = x'[A]$, $x[B] = x''[B] = z'[B] = z = x[B]$ and $x[T] = z'[T] = y'[T] = x[T] = (tt\_start, tt\_end, vt\_start, vt\_end)$.