Algebraic Query Languages on Temporal Databases with Multiple Time Granularities[*]

X. Sean Wang

Technical report ISSE-TR-94-107
Revised April 1995

Abstract

This paper investigates algebraic query languages on temporal databases. The data model used is a multidimensional extension of the temporal modules introduced in [WJS95]. In a multidimensional temporal module, every non-temporal fact has a timestamp that is a set of n-ary tuples of time points. A temporal module has a set of timestamped facts and has an associated temporal granularity (or temporal type), and a temporal database is a set of multidimensional temporal modules with possibly different temporal types. Temporal algebras are proposed on this database model. Example queries and results of the paper show that the algebras are rather expressive. The operations of the algebras are organized into two groups: snapshot-wise operations and timestamp operations. Snapshot-wise operations are extensions of the traditional relational algebra operations, while timestamp operations are extensions of first-order mappings from timestamps to timestamps. Multiple temporal types are only dealt with by these timestamp operations. Hierarchies of algebras are defined in terms of the dimensions of the temporal modules in the intermediate results. The symbol $\text{TALG}_k^{m,n}$ is used to denote all the algebra queries whose input, output and intermediate modules are of dimensions at most m, n and k, respectively. (Most temporal algebras proposed in the literature are in $\text{TALG}_1^{1,1}$.) Equivalent hierarchies $\text{TCALC}_k^{m,n}$ are defined in a calculus query language that is formulated by using a first-order logic with linear order. The addition of aggregation functions into the algebras is also studied.

[*]This work was partly supported by the NSF grant IRI-9409769 and also by an ARPA grant, administered by the Office of Naval Research under grant number N0014-92-J-4038.

1 Introduction

Temporal information plays an important role in various database applications, and because of this, many temporal data models and their query languages are proposed [TCG$^+$93]. These data models address many fundamental issues in temporal information modeling and manipulation. However, one important aspect that is missing from most of temporal database research in the literature is the data models and their query languages that deal with multiple time granularities (or temporal types).[1] In [WJS95], we introduced such a data model and its calculus query language. The purpose of this paper is to propose and investigate algebraic query languages that incorporate multiple temporal types.

Temporal modules defined in [WJS95] can be viewed as an abstract (or conceptual) temporal data model in which (a) each tuple is associated with a set of time points (i.e., a timestamp) and (b) each time point is associated with a set of tuples (i.e., the facts that hold at the time). In temporal module jargon, we model the aspect (a) into a tuple-windowing function τ, and aspect (b) time-windowing function φ. A tuple-windowing function accepts a tuple and returns the timestamp (i.e., a set of time points) of the tuple, and a time-windowing function accepts a time point and returns the facts (i.e., a set of tuples) that hold at the given time. These two windowing functions are the two views of the same information.

As our running example, we assume there is a group of robots who are performing certain tasks. The task that each robot is performing and the power consumptions at certain time are listed in the table of Figure 1. (The time points are in seconds that are measured from the first second that the robots were activated.) In this example, the tuple-windowing function returns the set $\{1, 130\}$ if the tuple

Robot	Task	PowerConsumption	Time (second)
Dan	Pick	10.2	1
Niel	Move	7	1
Wuyi	Flash	2	2
Bliss	Move	6.8	10
Dan	Move	7	10
Wuyi	Move	7.1	45
Wuyi	Pick	11	61
Dan	Flash	2.3	61
Bliss	Pick	10.8	130
Niel	Move	7	130

Figure 1: Robots and tasks.

$\langle Niel, Move, 7\rangle$ is given, i.e., $\tau(\langle Niel, Move, 7\rangle) = \{1, 130\}$, and the time-windowing function returns the set $\{\langle Dan, Pick, 10.2\rangle, \langle Niel, Move, 7\rangle\}$ if time 1 is given, i.e., $\varphi(1) = \{\langle Dan, Pick, 10.2\rangle, \langle Niel, Move, 7\rangle\}$.

Temporal modules abstract many of the data models proposed in the literature. Such an abstraction

[1]See Related Work section for details.

allows us to arrange our algebraic operations on temporal modules into two groups: The first group is what we call snapshot-wise operations and the second group timestamp operations. A snapshot of a temporal module is the set of tuples (i.e., a relation) that hold at a given time point. A snapshot-wise operation is simply an operation that operates on each snapshot of a temporal module. Clearly, any operation on (non-temporal) relations can be straightforwardly extended to a snapshot-wise operation. Here, we extend the traditional relational algebra operations into snapshot-wise operations. As an example, the selection operation that selects the tuples using the condition $Robot =' Bliss' \wedge Task =' Pick'$ will be performed on 6 snapshots: namely the 6 non-empty relations that are returned by the application of the time-windowing function on times 1, 2, 10, 45, 61 and 130, respectively. The result of this particular snapshot-wise operation gives empty set on the first 5 snapshots. At time 130, the selection returns a tuple $\langle Bliss, Pick, 10.8\rangle$. Thus, the result of this snapshot-wise operation returns a temporal module whose only non-empty snapshot is at time 130 and there is only one tuple $\langle Bliss, Pick, 10.8\rangle$ at that time.

The more interesting group of operations is the timestamp operations. This is where we differ from most proposals in the literature. A timestamp operation takes as input one or more timestamps (i.e., sets of time points) and returns one timestamp (i.e., one set of time points). Such a mapping is extended to temporal modules by applying the mapping on each non-empty result of the tuple-windowing function on every tuple. For example, let f be the mapping such that $f(I) = \{i | \exists j \in I(i < j)\}$, i.e., f returns all the time points that is smaller than some time point in the given set. Applying f to the temporal module corresponding to the table of Figure 1, among other things, we know that the new timestamp for $\langle Bliss, Pick, 10.8\rangle$ is $\{1, \ldots, 129\}$ since the timestamp for this tuple in the given module is $\{130\}$. Intuitively, a timestamp operation changes the time that facts hold for the purpose of user query. This is a powerful way of extracting temporal information. As an example, assume we want to find out if Dan ever moves *before* Bliss picks. We may do so by applying the above f mapping to the timestamp of each tuple, and then look at each snapshot of this new temporal module along with the corresponding snapshot of the original module: Obviously, iff there is a snapshot when Dan moves in the original module and Bliss picks in the new module, we know that Dan moves *before* Bliss picks. This last test on snapshots can be accomplished by a snapshot-wise natural join and snapshot-wise selection on the new temporal module and the original one. (The answer for this is "yes" based on the table in Figure 1 since in the new module, i.e., the temporal module after f is applied, contains a tuple $\langle Bliss, Pick, 10.8\rangle$ at time point 10, and the original modules contains tuple $\langle Dan, Move, 7\rangle$ at the same time point 10.)

This arrangement of timestamp operation also gives rise natural treatment of multiple temporal types. For example, assume that a user is interested in knowing that if Dan and Bliss ever perform the same task in one *minute*. This query can be easily accomplished by first changing the timestamp (by some timestamp operation) into minutes. For the table in Figure 1, the first 6 rows will be labeled as in minute 1, the next two rows minute 2 and the last two rows minute 3. Snapshot-wise operations can then be used to see if in any snapshot (now in terms of minutes), Dan and Bliss perform the same task. (The answer is "yes" since the rows 4 and 5 are now both labeled 1, i.e., Bliss and Dan both move in minute 1.) Such operations are similar to the `scale` operation of [DS94].

Algebraic operations on temporal modules should preserve the structure of temporal modules. Each algebraic operation should be a mapping from temporal modules to temporal modules, i.e., the input of the operation is one or more temporal modules and the output must also be a temporal module. The aforementioned operations all satisfy this property. The temporal modules as defined in [WJS95] are one dimensional, i.e., each fact is associated with a set of time points. In other words, only one kind of time (i.e., either valid time, or transaction time, or user time, etc.) is supported in temporal modules of [WJS95]. Such an arrangement may limit the expressiveness of algebraic query languages, for the information extracted by an operation must be encoded by such a one-dimensional temporal module. We conjecture in this paper that if we increase the dimensions of the *intermediate* temporal modules, the algebra will become more expressive. In this paper, we link this conjecture to the conjecture that calculus query languages based on first-order logic with linear order is *strictly* more expressive than those based first-order logic with temporal modalities **Since** and **Until** [CCT94, Cho94]. Specifically, we show that the hierarchy in the algebra that is defined by the dimensions of intermediate temporal modules is equivalent to a hierarchy, which is defined on the number of free time variables in certain subformulas, in a calculus query language based on first-order logic with linear order.

In light of the above discussion, we extend the temporal modules into multidimensional. That is, each timestamp is a set of n-ary tuples of time points, for some positive integer n. Such an extension can be intuitively viewed as to include valid time, transaction time, user time, reference time, etc [SA85, CI94]. However, our intension is that the multidimensionality is used more for the intermediate results rather than for the stored temporal modules. The snapshot-wise operations and the timestamp operations mentioned earlier are easily extended to multidimensional temporal modules.

When dealing with multiple temporal types, we not only need to convert timestamps in terms of one temporal type into that in terms of another temporal type, but often we need to change the facts accordingly. For example, one may ask the power consumptions of each robot in each minute, assuming the power consumption of a minute is the sum of the power consumption at the seconds within the minute. In this case, the aggregation function **sum** is needed. In order to perform this aggregation function, the tuples are grouped not only according to their attribute values, but also according to the timestamp: In this particular example, only if two seconds are within the same minute, the corresponding tuples can then be in a group. We introduce such aggregation operations into our algebra.

The rest of the paper is organized as follows: Related work is discussed in Section 2. In Section 3, temporal types and multidimensional temporal modules are defined. In Section 4, algebraic operations on temporal modules are given. Based on these operations, temporal algebras TALG, TALG$_k$ and TALG$_k^{m,n}$ are presented. Section 5 introduces corresponding calculus query languages TCALC, TCALC$_k$ and TCALC$_k^{m,n}$ which are to be used to compare with the algebras of Section 4. Section 6 proves that the algebras are equivalent in expressiveness to the corresponding, data-domain independent (a notion defined here) calculus. The addition of aggregation functions into temporal algebras is investigated in Section 7. Section 8 concludes the paper.

2 Related work

After a diversified, active research period, the temporal database area appears to have started to turn to unification. The design of TSQL2 [TSQ94] and the study of conceptual temporal models, which include the bi-temporal conceptual relations [JSS94] and the temporal module model [WJS95], are two developments within this general trend. The current paper continues the investigation of conceptual temporal data models, namely algebraic query languages we call TALG on temporal modules. We are not aware of any other algebraic query languages that incorporate multiple temporal types, which places TALG in a unique position. However, we find the work of TSQL2 [TSQ94], including the algebra for TSQL2 [SJS94], the bi-temporal data model [JSS94], and the work on temporal aggregations [SGM93, Tan87] are related to the current paper.

As mentioned earlier, most of the temporal data models and their query languages in the literature do not support multiple temporal types. One important exception is TSQL2. The TSQL2 language displays an impressive array of features that include the support for multiple calendars and granularities and aggregation. However, the algebra that is designed for TSQL2 [SJS94] does not consider the issue of multiple granularities. Therefore, the TALG algebra can be viewed as a complement to the algebra of [SJS94]. Some of the features that are in TSQL2, such as "sliding window" aggregation (e.g., three-month averages starting from each month), are not expressible in TALG and worth further investigation.

Although it does not deal with multiple granularities, the algebra for the bi-temporal relational model [JSS94] is also an algebraic query language on a conceptual model. However, one important difference between TALG and the algebra of [JSS94] is in the organization of operations. TALG operations are organized into two groups: snapshot-wise operations and timestamp operations, while the bi-temporal algebra operations are more integrated. We believe that the separation of the two groups in TALG makes the query language more intuitive, and gives rise natural treatment of multiple temporal types. Another important difference is that in TALG, we allow multidimensional temporal modules in the intermediate results, even if the input and output are restricted to one dimensional. We conjecture that this makes TALG more expressive than the bi-temporal algebra.

The addition of aggregation functions into TALG is different from that of TSQL2, [SGM93] and [Tan87]. In TSQL2, [SGM93] and [Tan87], aggregates are performed on a set of timestamped facts. In contrast, we believe that an aggregation function should not take the time into consideration. Any time related manipulation should be dealt with by other constructs. This separation follows the spirit of the separation of snapshot-wise operations and timestamp operations. Also, the aggregation in the current paper takes advantage of the multitude of temporal types. On the other hand, the dependence on the temporal types limits the ability to express certain intuitive aggregates that are expressible in TSQL2.

Another research area that is related to the current paper is the work on multiple calendars, e.g., [CR87, NS92, CSS94, SS92]. These work are more focused on the management or description of calendars but not on incorporating them into query languages.

3 Data model

This section introduces a data model that is an extension of the one presented in [WJS95].

3.1 Temporal types

We start with defining temporal types that model typical (and atypical) calendar units.[2] We assume there is an underlying notion of absolute time, represented by the set $\mathcal{N}$ of all positive integers.

Definition Let $\mathcal{I}_{\mathcal{N}}$ be the set of all intervals on $\mathcal{N}$, i.e., $\mathcal{I}_{\mathcal{N}} = \{[i,j] \mid i,j \in \mathcal{N} \text{ and } i \leq j\} \cup \{[i,\infty] \mid i \in \mathcal{N}\}$.[3] A *temporal type* is a mapping μ from the set of the positive integers (the *time ticks*) to the set $\mathcal{I}_{\mathcal{N}} \cup \{\emptyset\}$ (i.e., all intervals on $\mathcal{N}$ plus the empty set) such that for each positive integer i, all following conditions are satisfied:

(1) if $\mu(i) = [k,l]$ and $\mu(i+1) = [m,n]$, then $m = l+1$.

(2) $\mu(i) = \emptyset$ implies $\mu(i+1) = \emptyset$.

(3) there exists j such that $\mu(j) = [k,l]$ with $k \leq i \leq l$.

For each positive integer i and temporal type μ, $\mu(i)$ is called the i-th tick (or tick i) of μ. Condition (1) states that the ticks of a temporal type need to be *monotonic* and *contiguous*, i.e., the subsequent tick (if not empty) is the next contiguous interval. Condition (2) disallows a temporal type to have an empty tick unless all its subsequent ticks are empty. And condition (3) requires that each absolute time value must be included in a tick. One particular consequence of the above three conditions is that the last non-empty tick (if it exists) must be an interval of the form $[i,\infty]$.

Typical calendar units, e.g., `day`, `month`, `week` and `year`, can be defined as temporal types that follow the above definition, when the underlying absolute time is discrete.

An important relation regarding temporal types involves time ticks. For example, we would like to say that a particular month is within a particular year. For this purpose, we assume there is a binary (interpreted) predicate $\mathbf{IntSec}_{\mu,\nu}$ for each pair of temporal types μ and ν:

Definition For temporal types μ and ν, let $\mathbf{IntSec}_{\mu,\nu}$ be the binary predicate on positive integers such that $\mathbf{IntSec}_{\mu,\nu}(i,j)$ is **true** if $\mu(i) \cap \nu(j) \neq \emptyset$, and $\mathbf{IntSec}_{\mu,\nu}(i,j)$ is **false** otherwise.

In order words, $\mathbf{IntSec}_{\mu,\nu}(i,j)$ is **true** iff the intersection of the corresponding absolute time intervals of tick i of μ and tick j of ν is not empty. For instance, $\mathbf{IntSec}_{\mathtt{month},\mathtt{year}}(i,j)$ is true iff the month i falls within the year j.

[2]This subsection borrows heavily from [BWBJ95].

[3]An interval $[i,j]$ ($[i,\infty]$, resp.) is viewed as the set of all integers k such that $i \leq k \leq j$ ($k \geq i$, resp.).

3.2 Temporal module schemes and temporal modules

We assume there is a set of attributes and a set of values called the *data domain*. Each finite set R of attributes is called a *relation scheme*. A relation scheme $R = \{A_1, \ldots, A_n\}$ is usually written as $\langle A_1, \ldots, A_n \rangle$. For relation scheme R, let $\mathbf{Tup}(R)$ denote the set of all mappings, called *tuples*, from R to the data domain. A tuple t of relation scheme $\langle A_1, \ldots, A_n \rangle$ is usually written as $\langle a_1, \ldots, a_n \rangle$, where $a_i = t(i)$ for each $1 \le i \le n$.

Definition For each positive integer n, an n-dimensional temporal module scheme is a triple (R, μ, n) where R is a relation scheme and μ a temporal type. A *n-dimensional temporal module* on (R, μ, n) is a 5-tuple $(R, \mu, n, \varphi, \tau)$, where

1. φ is a mapping, called *time windowing function*, from $\mathcal{N} \times \cdots \times \mathcal{N}$ (n times) to $2^{\mathbf{Tup}(R)}$, and
2. τ is a mapping, called *tuple windowing function*, from $\mathbf{Tup}(R)$ to $2^{\mathcal{N} \times \cdots \times \mathcal{N}}$ ($\mathcal{N}$ appears n times),

such that (a) for positive integers $i_1, \ldots, i_n$, $\varphi(i_1, \ldots, i_n) = \emptyset$ if $\mu(i_j) = \emptyset$ for some $1 \le j \le n$, and (b) for all positive integers $i_1, \ldots, i_n$ and tuple t, $(i_1, \ldots, i_n) \in \tau(t)$ iff $t \in \varphi(i_1, \ldots, i_n)$.

Throughout the paper, we assume that the temporal modules are *finite*, i.e., $\bigcup_{i \ge 1} \varphi(i)$ is a finite set, where φ is the time windowing function of a temporal module. Note that this finiteness does not exclude those temporal modules that have an infinite number of i such that $\varphi(i) \neq \emptyset$. We do require, however, that the number of distinct tuples (regardless of time) is finite. In other words, we require that there are only a finite number of tuples t such that $\tau(t) \neq \emptyset$, where τ is the tuple windowing function of a temporal module.

Intuitively, the time windowing function φ in a temporal module $(R, \mu, n, \varphi, \tau)$ gives the tuples (facts) that hold at (the combination of) non-empty time ticks $i_1, \ldots, i_n$ of temporal type μ. This is a generalization of many temporal models in the literature. Here, the multidimensionality reflects the *valid time*, *transaction time*, *user time*, and so on [TCG$^+$93]. However, it will become clear later that we will be focusing on unary temporal modules when we consider the expressiveness of our query languages. Condition (b) above requires that tuple windowing function τ be the inverse of φ. Thus, when defining a temporal module, we only need to tell what φ (τ, resp.) is and τ (φ, resp.) will be "derived" from φ (τ, resp.).

Another viewpoint is that the time-windowing function of an n-dimensional temporal module gives, for positive integers $i_1, \ldots, i_n$, the snapshot of the temporal module at time $i_1, \ldots, i_n$, while tuple-windowing function gives, for each tuple t, the (n-dimensional) timestamps of t in the temporal module. These two views allow us to organize our algebraic operations (described later) into two categories: "snapshot-wise" operations and "timestamp" operations.

Example 1 The table in Figure 1 gives the temporal module $\texttt{Robots} = (R, \texttt{second}, 1, \varphi, \tau)$, where $R = \langle Robot, Task, PowerConsumption \rangle$ and φ is defined as follows:

$$\begin{aligned}
\varphi(1) &= \{\langle Dan, Pick, 10.2\rangle, \langle Niel, Move, 7\rangle\} \\
\varphi(2) &= \{\langle Wuyi, Flash, 2\rangle\} \\
\varphi(10) &= \{\langle Bliss, Move, 6.8\rangle, \langle Dan, Move, 7\rangle\} \\
\varphi(45) &= \{\langle Wuyi, Move, 7.1\rangle\} \\
\varphi(61) &= \{\langle Wuyi, Pick, 11\rangle, \langle Dan, Flash, 2.3\rangle\} \\
\varphi(130) &= \{\langle Bliss, Pick, 10.8\rangle, \langle Niel, Move, 7\rangle\}
\end{aligned}$$

and $\varphi(i) = \emptyset$ for all other times. The tuple-windowing function can be derived from the above time-windowing function. □

A *temporal database scheme* is a finite set of temporal module schemes, each of which is assigned a unique name. A *temporal database* is a finite set of temporal modules, each of which is associated with a scheme name and is a temporal module on the corresponding temporal scheme.

4 Temporal algebras

In this section, we first present our algebraic operations on temporal modules. By using these operations, we then define our temporal algebras.

The operations are of two kinds. The first kind is "snapshot-wise operations". Here we adopt the traditional relational algebra operations. These traditional operations will operate on each "snapshot" of temporal modules. A *snapshot* of an n-ary temporal module $(R, \mu, n, \varphi, \tau)$ at time $i_1, \ldots, i_n$ is the relation $\varphi(i_1, \ldots, i_n)$.

4.1 Snapshot-wise operations

We have the following operations that map from a single temporal module to a single temporal module. We assume $M = (R, \mu, n, \varphi, \tau)$. Note that π and σ in the following definitions are the traditional projection and selection operations, respectively, in the relational algebra.

Projection. If $\{A_1, \ldots, A_k\} \subseteq R$, then $\pi^m_{A_1,\ldots,A_k}(M) = (\langle A_1, \ldots, A_k\rangle, \mu, n, \varphi', \tau')$, where for all positive integers $i_1, \ldots, i_n$, $\varphi'(i_1, \ldots, i_n) = \pi_{A_1,\ldots,A_k}(\varphi(i_1, \ldots, i_n))$.

Selection. If P is a selection condition in the traditional relational algebra that involves only attributes in R, then $\sigma^m_P(M) = (R, \mu, n, \varphi', \tau')$, where for all positive integers $i_1, \ldots, i_n$, $\varphi'(i_1, \ldots, i_n) = \sigma_P(\varphi(i_1, \ldots, i_n))$.

The following are operations that map from two temporal modules to a single one. We assume $M_1 = (R_1, \mu, n, \varphi_1, \tau_1)$ and $M_2 = (R_2, \mu, n, \varphi_2, \tau_2)$. Note that M_1 and M_2 have the same temporal type and the same dimension, and $\bowtie$ in the following definition is the natural join in the traditional relational algebra.

Natural join. $\mathtt{M}_1 \bowtie^m \mathtt{M}_2 = (R_1 \cup R_2, \mu, n, \varphi', \tau')$, where for all positive integers $i_1, \ldots, i_n, \varphi'(i_1, \ldots, i_n) = \varphi_1(i_1, \ldots, i_n) \bowtie \varphi_2(i_1, \ldots, i_n)$.

We have the following extension of the standard set operations. In the following definitions, M_1 and M_2 are as given above, but with the condition that $R_1 = R_2$. Note $\cup$ and $-$ are the standard set union and difference, respectively.

Union. $M_1 \cup^m M_2 = (R_1, \mu, n, \varphi', \tau')$, where for all positive integers $i_1, \ldots, i_n$, $\varphi'(i_1, \ldots, i_n) = \varphi_1(i_1, \ldots, i_n) \cup \varphi_2(i_1, \ldots, i_n)$.

Difference. $M_1 -^m M_2 = (R_1, \mu, n, \varphi', \tau')$, where for all positive integers $i_1, \ldots, i_n$, $\varphi'(i_1, \ldots, i_n) = \varphi_1(i_1, \ldots, i_n) - \varphi_2(i_1, \ldots, i_n)$.

As usual, we may define our intersection $\cap^m$ by using $-^m$.

The notations for snapshot-wise operations all include a superscript ^{m}, signifying that the corresponding operations are applied to each snapshot of temporal *m*odules. *In the sequel, when no confusion is possible, we shall drop this superscript* ^{m}.

4.2 Timestamp operations

The other kind is "timestamp operations". The first such operation is derived from **IntSec**. We assume $M = (R, \mu, n, \varphi, \tau)$.

Temporal type conversion. If ν is a temporal type, then $\kappa_\nu(M) = (R, \nu, n, \varphi', \tau')$, where for each tuple $t \in \mathbf{Tup}(R)$,

$$\tau'(t) = \{(j_1, \ldots, j_n) | \exists (i_1, \ldots, i_n) \in \tau(t)(\textbf{IntSec}_{\mu,\nu}(i_p, j_p) = \mathtt{true} \text{ for each } 1 \le p \le n)\}.$$

In other words, a fact is taken to be valid at the tick combination $(j_1, \ldots, j_n)$ of ν if it is valid at tick combination $(i_1, \ldots, i_n)$ of μ such that tick each i_p of μ intersects with the corresponding tick j_p of ν. For example, $\kappa_{\mathtt{minute}}(\mathtt{Robots})$, where $\mathtt{Robots} = (R, \mathtt{second}, 1, \phi, \tau)$ is as in Example 1, gives the temporal module $(R, \mathtt{minute}, 1, \phi', \tau')$, where $\phi'(1) = \phi(1) \cup \phi(2) \cup \phi(10) \cup \phi(45)$, $\phi'(2) = \phi(61)$, $\phi'(3) = \phi(130)$, and $\phi'(i) = \emptyset$ for each $i \ge 4$.

It is easily seen that the operation here only changes the timestamps of the facts, but not the facts themselves. In Section 7, we will see how the facts themselves are manipulated via aggregation functions.

Another suite of time operations are from any given class of mappings on timestamp sets. A *k-ary timestamp operation* is a mapping from $2^{\mathcal{N}^{n_1}} \times \cdots \times 2^{\mathcal{N}^{n_k}}$ to $2^{\mathcal{N}^m}$, where $n_1, \ldots, n_k$ and m are positive integers. That is, it is a mapping whose input is k sets of n_1-ary tuples of time points, ..., n_k-ary tuples of time points, respectively, and whose output is a set of m-ary tuples of time points. And furthermore, the input sets and the output set of the mapping are of fixed temporal types. We call these arities and types as the *signature* of the mapping, denoted by $(n_1, \mu_1, \ldots, n_k, \mu_k) \to (m, \nu)$. Intuitively, we take the input sets as n_j-ary timestamps in terms of the temporal type μ_j $(1 \leq j \leq k)$ and the output set as m-ary timestamps in terms of ν. Thus, a timestamp operation of signature $(n_1, \mu_1, \ldots, n_k, \mu_k) \to (m, \nu)$ converts k sets of timestamps into one set of timestamps in terms of ν. A collection of such timestamp operations is called a *timestamp operation system.*

The timestamp operations in a timestamp operation system are extended to operations on temporal modules as follows. Let $M_1 = (R_1, \mu_1, n_1, \varphi_1, \tau_1), \ldots, M_k = (R_k, \mu_k, n_k, \varphi_k, \tau_k)$ be temporal modules.

General time operation. If f is a k-ary timestamp operation and $(n_1, \mu_1, \ldots, n_k, \mu_k) \to (m, \mu)$ its signature, then $[\![f]\!](M_1, \ldots, M_k) = (R, \mu, m, \varphi', \tau')$, where $R = R_1 \cup \cdots \cup R_k$ and for each tuple $t \in \mathbf{Tup}(R)$,

$$\tau'(t) = \{(j_1, \ldots, j_m) | \exists t_1, \ldots, t_k((j_1, \ldots, j_m) \in f(\tau_1(t_1), \ldots, \tau_k(t_k)) \text{ and } (\tau_i(t_i) \neq \emptyset \wedge t_i = t[R_i]) \text{ for each } 1 \leq i \leq k) \}$$

That is, for each combination of k tuples $t_1, \ldots, t_k$ that are from $M_1, \ldots, M_k$, respectively, and have common values for common attributes, we create one tuple t and the timestamp of t is a m-dimensional timestamp that is the result of applying f on the timestamps of $t_1, \ldots, t_k$. Another point of view is that we take the natural join of all the tuples from the k (if $k = 1$, no natural join is needed) temporal modules with no regard of their timestamps first. Then we assign timestamp for each tuple t by using the result of applying f function on the timestamps of the tuples that contribute to t. A tuple t_i contributes to t if $t[R_i] = t_i$ for each $i = 1, \ldots, k$.

As an example, assume that $M_1 = (\langle A, B \rangle, \mu_1, 1, \phi_1, \tau_1)$ and $M_2 = (\langle B, C \rangle, \mu_2, \phi_2, \tau_2)$, where τ_1 and τ_2 are given as follows: (i) $\tau_1(\langle a_1, b_1 \rangle) = \{1, 2, 3, 5\}, \tau_1(\langle a_2, b_2 \rangle) = \{2, 3\}$, and τ_1 on all other tuples gives the empty set, and (ii) $\tau_2(\langle b_1, c \rangle) = \{3, 4\}$ and τ_2 gives the empty set on all other tuples. Let f be the mapping of the signature $(1, \mu_1, 1, \mu_2) \to (1, \nu)$ defined by $\{i | i \in I_1 \wedge \exists j (j \geq i \wedge j \in I_2)\}$, i.e., i is in the output if i is in I_1 and j is in I_2 for some later time j. Now $[\![f]\!](M_1, M_2) = (\langle A, B, C \rangle, \nu, 1, \phi', \tau')$, where τ' is defined as follows: $\tau'(\langle a_1, b_1, c \rangle) = \{1, 2, 3\}$ and gives the empty set on all other tuples.

MFO$^{\leq}$

There are many natural timestamp operation systems. Here we use a particular system, namely MFO$^{\leq}$, adopted from formulas of many sorted first-order logic with linear order. The sorts are the temporal types that we use. Each constant and variable has a particular temporal sort. (We shall use temporal sort and temporal type interchangeably.) Each predicate has a temporal type associated for its arguments (all arguments of a predicate are of the same type). The atomic formulas are of the form (1) $p_j(i_1, \ldots, i_n)$, where p_j is an n-ary predicate of type μ and $i_1, \ldots, i_n$ are constants or variables of type μ, for some temporal type μ, (2) $i_1 \theta i_2$, where i_1 and i_2 are of the same temporal type and $\theta \in \{=, \neq, <, \leq, >, \geq\}$, and (3) $\mathbf{IntSec}_{\mu,\nu}(i_1, i_2)$, where i_1 and i_2 are type μ and ν, respectively. The formulas of this logic is defined recursively as usual by allowing Boolean connectors and quantifiers. Furthermore, if k distinct predicate appear in a formula, we shall use $p_1, \ldots, p_k$ as their names, and $\mu_1, \ldots, \mu_n$ as their temporal types. As a syntactic sugar, when a quantifier is used, we may write $\exists i{:}\mu$ to indicate the type of i is μ (similarly for $\forall$). A formula ψ is in MFO$^{\leq}$ iff it is a formula in the above logic and all its free variables are of the same type ν. Suppose $p_1, \ldots, p_k$ are all the predicates appearing in the MFO$^{\leq}$ formula ψ, whose arities and types are $n_1, \ldots, n_k$ and $\mu_1, \ldots, \mu_k$, respectively. Furthermore, suppose the number of free variables of ψ is m and all these m free variables of ψ are of type ν. Then we say ψ has the *signature* $(n_1, \mu_1, \ldots, n_k, \mu_k) \rightarrow (m, \nu)$.

Example 2 Let ψ be the following formula of signature $(1, \texttt{second}) \rightarrow (1, \texttt{minute})$:

$$p_1(i) \wedge \exists j_1{:}\texttt{minute}(\textbf{IntSec}_{\texttt{second},\texttt{minute}}(i, j_1) \wedge \forall i'{:}\texttt{second}((p_1(i') \wedge \textbf{IntSec}_{\texttt{second},\texttt{minute}}(i', j_1)) \rightarrow i' < i))$$

Thus, $\psi(i)$, where i is a second, is true if i is a second that makes p_1 true (i.e., $p_1(i)$), and we can find a minute j_1 such that second i is in minute j_1, and all seconds (i') that are in minute j_1 and that make p_1 true should be before second i. Therefore, $\psi(i)$ is true iff i is the last second in a minute that makes p_1 true. $\square$

Assume ψ is a MFO$^{\leq}$ formula of the signature $(n_1, \mu_1, \ldots, n_k, \mu_k) \rightarrow (m, \nu)$. Then this formula can be viewed as a k-ary timestamp operations of the signature $(n_1, \mu_1, \ldots, n_k, \mu_k) \rightarrow (m, \nu)$. Indeed, if a subset I_j of $\mathcal{N}^{n_j}$ is viewed as the interpretation of the n_j-ary predicate $p_j()$ such that $p_j(i_1, \ldots, i_{n_j})$ is true iff $(i_1, \ldots, i_{n_j}) \in I$, then ψ gives the following set: $\{i_1, \ldots, i_m | \psi(i_1, \ldots, i_m)$ is **true**$\}$. We call such a temporal operation system also as MFO$^{\leq}$ (when no confusion arises) and, unless specified otherwise, *all timestamp operations we use in the sequel are in MFO$^{\leq}$.*

Note that the type conversion operation (from a temporal module with a specific temporal type and dimension) can be expressed as a general time operation with the appropriate timestamp operations. Furthermore, the natural join operation on temporal modules with the specific temporal type μ and specific dimension n can be defined by a general time operation using the binary timestamp function f of signature $(n, \mu, n, \mu) \rightarrow (n, \mu)$ defined by the MFO$^{\leq}$ formula $p_1(i_1, \ldots, i_n) \cap p_2(i_1, \ldots, i_n)$.

Example 3 By using the general timestamp operation, we can achieve the following query on Example 1: "Find the last action of Bliss during the first minute." This query can be written as the following sequences of operations:

$$\pi_{Task}([\![f]\!](\sigma_{Robot='Bliss'}(\texttt{Robots}))),$$

where

$$f(i) = p_1(i) \wedge \textbf{IntSec}_{\texttt{second},\texttt{minute}}(i, 1) \wedge \forall i'{:}\texttt{second}((p_1(i') \wedge \textbf{IntSec}_{\texttt{second},\texttt{minute}}(i', 1)) \rightarrow i' < i))$$

and i is in $\texttt{second}$. $\square$

We note in passing here that all snapshot-wise operations are defined in terms of the time windowing functions φ since each application of the time windowing function gives a snapshot, while all time operations are defined in term of the tuple windowing functions τ since each application of the tuple windowing function gives the timestamp of the given tuple.

4.3 The temporal algebra TALG and algebras TALG$_k$

We now define our algebra TALG. For a given database scheme S, the following are TALG expressions on S. Each expression has a corresponding temporal module scheme.

Constant module. If a is a constant in the data domain, A an attribute, μ a temporal type, and $i_1, \ldots, i_k$ positive integers such that $\mu(i_j) \neq \emptyset$, $1 \leq j \leq k$, then $(A, a, \mu, i_1, \ldots, i_k)$ is a TALG expression on S.

Database module. If `M` is a scheme name in S, then `M` is a TALG expression on S.

Snapshot-wise operations. If e_1 and e_2 are TALG expressions on S, then $\pi_{A_1,\ldots,A_k}(e_1)$, $\sigma_P(e_1)$, $(e_1 \bowtie e_2)$, $(e_1 \cup e_2)$, $(e_1 - e_2)$ are all TALG expressions on S.

Time operations. If $e_1, \ldots, e_k$ are TALG expressions on S, ν a temporal type and f a k-ary timestamp operation, then $\kappa_\nu(e_1)$ and $[\![f]\!](e_1, \ldots, e_k)$ are both TALG expressions on S.

Rename. If A and B are two attributes and e is a TALG expression on S, then $\rho_{A \to B}(e)$ is a TALG expression on S.

In Figure 2, we summarize all the algebraic operations we have defined. Note that we dropped the superscript ^{m} from the notations for the snapshot-wise operations.

Group	Name	Notation
Snapshot-wise operations	Projection	$\pi_{A_1,\ldots,A_k}(e)$
	Selection	$\sigma_P(e)$
	Natural join	$e_1 \bowtie e_2$
	Union	$e_1 \cup e_2$
	Difference	$e_2 - e_2$
Time operations	Temporal type conversion	$\kappa_\nu(e)$
	General time operation	$[\![f]\!](e_1, \ldots, e_k)$
Misc. operations	Constant Module	$(A, a, \mu, i_1, \ldots, i_k)$
	Database Module	`M`
	Rename	$\rho_{A \to B}(e)$

Figure 2: All the operations in TALG.

Each TALG expression on S has a temporal scheme. The scheme of $(A, a, \mu, i_1, \ldots, i_k)$ is $(\langle A \rangle, \mu, k)$ and the schemes of the the rest of the expressions are given naturally. Since each operation is a partial mapping depending on the scheme(s) of the input expression(s) (see the definitions of the operations), certain expressions are not correctly typed, i.e., the operations are not defined on the input modules of the given schemes.

Definition For each temporal database scheme S, the collection of all the correctly typed TALG expressions on S is called the TALG *algebra on* S, or simply the TALG *algebra* when S is understood.

We may now define two hierarchies within TALG:

Definition For each positive integer k, a TALG expression e (on S) is said to in the TALG$_k$ *algebra (on* S*)* if the dimension of each subexpression of e is at most k. A TALG expression e is said to be in the TALG$_k^{m,n}$ *algebra (on* S*)* if (a) it is in TALG$_k$, (b) all the temporal module scheme names appearing in it is at most m-dimensional, and (c) the scheme for e is n-dimensional.

In other words, expressions in TALG$_k^{m,n}$ have three properties: (a) the dimensions of the input temporal modules are at most m, (b) the dimension of the output temporal module is at most n, and (c) for each intermediate temporal module (as the result of a subexpression), the dimension is at most k. Note here that $n \leq k$ and $m \leq k$ by definition since an expression is also a subexpression of itself, and each temporal module scheme name appearing in the expression is also a subexpression.

Note that most valid-time temporal algebras that appeared in the literature are equivalent to TALG$_1^{1,1}$.

Example 4 We now use TALG to express the following queries:

1. "Who moved before the first time Bliss picked?"

$$\pi_{Robot}[\![f_2]\!](\sigma_{Task='Move'}(\texttt{Robots}), \pi([\![f_1]\!](\sigma_{Robot='Bliss' \wedge Task='Pick'}(\texttt{Robots})))),$$

   where $f_1(i) = \exists i'(i < i' \wedge p_1(i') \wedge \neg\exists j(j < i' \wedge p_1(j)))$ and $f_2(i) = p_1(i) \wedge p_2(i)$. Note that all temporal variables are of type `second` and the project π gets rid of all data attributes.

2. "Did Dan and Bliss ever perform the same task during the same minute?"

$$\pi_{Task}([\![f_1]\!](\sigma_{Robot='Dan'}(\texttt{Robots})) \bowtie (\pi_{Task}[\![f_1]\!](\sigma_{Robot='Bliss'}(\texttt{Robots}))),$$

   where $f_1(i{:}\texttt{minute}) = \exists j{:}\texttt{second}(\textbf{IntSec}_{\texttt{second,minute}}(j, i) \wedge p_1(j))$. The answer is "yes" iff the above query returns a non-empty set.

3. "Find the time when the workshop state is first repeated." Here, a "workshop state" is a function from robots to tasks. That is, the tasks that robots are performing defines the workshop state. And "first repeated" means that the first time that workshop is in the same state as at a previous time. Let $\texttt{States} = \pi_{Robot,Task}(\texttt{Robots})$. Then the this query is expressed in TALG as follows:

$$[\![f_2]\!](\pi([\![f_1]\!](\texttt{States}, \texttt{States}))),$$

   where $f_1(i, j) = i < j \wedge \neg(p_1(i) \vee p_2(j))$ and $f_2(j) = \exists i(\neg p_1(i, j)) \wedge \forall j'(j' < j \rightarrow \neg\exists i' \neg p_1(i', j'))$. Intuitively, the timestamp operation done by f_1 gives a two dimensional temporal module such that if (i, j) is in the timestamp of $\langle r, t\rangle$, then the robot r *is not* doing the same task t at time i and j. The projection π without any subscript is, in effect, the union all these pairs (i,j) such that some robot r is not doing the same task at i and j. So the negation of these pairs are the times that each robot r is performing the same task (different robots may be performing different tasks). The timestamp operation given by f_2 is to retrieve the second time point this happens, i.e., the first time the workshop state is repeated. All the temporal types in the above expression are in `second`.

Note that the first two queries are in $\text{TALG}_1^{1,1}$ and the last one is in $\text{TALG}_2^{1,1}$. We conjecture that the last query cannot be expressed in $\text{TALG}_1^{1,1}$. $\square$

5 Temporal calculi

A temporal module scheme $\mathtt{M} = (\langle A_1,\ldots,A_k\rangle,\mu,n)$ can be viewed as a $(k+n)$-ary predicate with the first k position being of data sort and the last n positions being of a time sort (of type μ). A temporal module $M = (\langle A_1,\ldots,A_k\rangle,\mu,n,\varphi,\tau)$ can be viewed as an interpretation of the predicate for $\mathtt{M} = (\langle A_1,\ldots,A_k\rangle,\mu,n)$ defined as follows: $\mathtt{M}(x_1,\ldots,x_k,i_1,\ldots,i_n)$ is **true** (under the interpretation M) iff $\langle x_1,\ldots,x_k\rangle$ is in $\varphi(i_1,\ldots,i_n)$ (or equivalently, iff $(i_1,\ldots,i_n)\in\tau(\langle x_1,\ldots,x_k\rangle)$). Thus, it is natural to use a many-sorted first-order logic to construct a calculus query language on temporal modules.[4]

In our many sorted logic, we assume we have one data sort and many temporal sorts. Each temporal type we use gives rise to a distinct temporal sort. Each variable and constant has a fixed sort. We use x, possibly subscripted, to denote a variable or constant of the data sort, and i and j, possibly subscripted, to denote a variable or constant of a temporal sort. We shall use "temporal type" and "temporal sort" interchangeably if no confusion arises.

The range of each data variable is the data domain and the range of each temporal sort variable is $\mathcal{N}$. Atomic TCALC formulas are of the following four types: (a) $x_1 = x_2$ where x_1 and x_2 are data terms; (b) $\mathbf{IntSec}_{\mu,\nu}(i_1,i_2)$, where i_1 and i_2 are variables or constants respectively of sorts μ and ν; (c) $i_1\theta i_2$, where i_1 and i_2 are variables or constants of the same temporal sort and $\theta\in\{=,\neq,<,\leq,>,\geq\}$, and (d) $\mathtt{M}(x_1,\ldots,x_k,i_1,\ldots,i_n)$, where $\mathtt{M} = (\langle A_1,\ldots,A_k\rangle,\mu,n)$ is a scheme name of the database scheme with k being the arity of R, each x_i is a non-temporal variable name or constant and each i_j a temporal variable or constant of sort μ. A TCALC *formula* is formed by Boolean connectives and the existential and universal quantifiers in a usual way. As a syntactic sugar, we may change the quantification of temporal sorts to the form: $\exists t:\mu$ and $\forall t:\mu$, where t is of sort μ. This syntactic sugar allows us to tell the sorts of bounded variables from the formula itself. The predicates **IntSec** are interpreted earlier, $\leq$ etc are interpreted on the integer order, and $=$ is the standard equality.

Note that TCALC formulas are similar to $\text{MFO}^{\leq}$ formula, except that $\text{MFO}^{\leq}$ has no temporal module predicates $\mathtt{M}()$ and no data variables.

A TCALC *query* is of the form

$$\{x_1\ldots x_k,i_1,\ldots,i_n{:}\nu \mid \psi(x_1,\ldots,x_k,i_1,\ldots,i_n)\}$$

[4]Here again, we borrow heavily from [BWBJ95] for the definition of TCALC.

where $x_1 \ldots x_k$ are variable names or constants of the data sort, $i_1, \ldots, i_n$ are temporal variables or constants of (the same) type ν, and $\psi(x_1, \ldots, x_k, i_1, \ldots, i_n)$ is a TCALC formula whose only free variables are among $x_1, \ldots, x_k, i_1, \ldots, i_n$. The formula ψ can obviously contain temporal variables of sorts different from ν, provided that they are bounded.

The answer of a TCALC query Q is defined naturally by considering the whole data domain as the range of data variables and the positive integers as the range of time variables.

Definition The collection of all TCALC queries on a database scheme S is called the TCALC *calculus on* S, or simply TCALC *calculus* when S is understood.

Similar to TALG, we now define two hierarchies of TCALC as follows:

Definition For each positive integer k, a TCALC query $\{x_1, \ldots, x_p, i_1, \ldots, i_q{:}\nu|\psi\}$ (on S) is said to be in the TCALC$_k$ *calculus (on S)* if $q \leq k$ and for all subformula of ψ of the form $\exists x\psi'$ or $\forall x\psi'$, where x is a data variable, ψ' has at most k *free* time variables. Furthermore, a TCALC query $\{x_1, \ldots, x_p, i_1, \ldots, i_q{:}\nu|\psi\}$ is said to be in the TCALC$_k^{m,n}$ *calculus (on S)*, where $n \leq k$ and $m \leq k$, if (a) it is in TCALC$_k$, (b) the dimensions of all temporal module schemes appearing in ψ are at most m, and (c) $q \leq n$, i.e., the number of time variables or constants in the answer set is at most n.

Note that we only count the number of free time variables in the subformulas for TCALC$_k$. There can be any number of bounded time variables.

Intuitively, a query in TCALC$_k^{m,n}$ is a query whose inputs are temporal modules of dimensions at most m, whose output is a temporal module of dimension at most n, and the number of free time variables in each subformula that starts with quantifications on data variables is at most k. This last point actually corresponds to the idea that the intermediate temporal modules shall be at most k-dimensional.

For example, the following query is in TCALC$_3^{1,2}$:

$$\{i_1, i_2{:}\mathtt{day}|\forall i_3 \exists y{:}\mathtt{day}(\mathtt{M}(y, i_1) \wedge \mathtt{M}(y, i_2) \wedge \mathtt{M}(y, i_3))\}.$$

This query finds out the pairs of days such that for all days, there is a common value that holds at all these three days.

It is interesting to note that, a (valid time) temporal query language (such as TL of [CCT94]) based on temporal logic with temporal modalities **Until** and **Since** is basically[5] TCALC$_1^{1,1}$, while a (valid time) temporal query language based on first-order logic with linear-order (such as TC of [CCT94]) is basically TCALC$^{1,1} = \bigcup_{k \geq 1}$ TCALC$_k^{1,1}$. The question whether TCALC$_1^{1,1}$ is equivalent to TCALC^{1,1} is still open, even if there is only one temporal type involved. Evidences show that they are not [Cho94]. In the next section, we prove that a safe segment of TCALC$_k^{m,n}$ is equivalent in expressiveness to TALG$_k^{m,n}$, for positive integers k, m and n. Thus, that TCALC$_1^{1,1}$ is equivalent to TCALC^{1,1} implies that the hierarchy TALG$_k^{1,1}$ ($k \geq 1$) collapse to TALG$_1^{1,1}$. However, we conjecture that the TALG$_k^{1,1}$ hierarchy does not collapse:

[5]Since our query languages deal with multiple temporal types while other languages do not, we have to restrict our queries to those that use only one temporal types to compare with other query languages.

Conjecture The hierarchy $\text{TALG}_k^{m,n}$ does not collapse for fixed m and n. That is $\text{TALG}_p^{m,n} \subset \text{TALG}_{p+1}^{m,n} \subset \cdots$, where $p = min(m, n)$. As a consequence, all the hierarchies TALG_k, $\text{TCALC}_k^{m,n}$ and TCALC_k do not collapse. The above hold even if there is only one temporal type involved.

Example 5 We now use TCALC to express the queries given in Example 4:

1. "Who moved before the first time Bliss picked?"

$$\{r, j | \exists i, j \exists p_1, p_2 (i < j \wedge \texttt{Robots}(r, 'Move', p_1, i) \wedge \texttt{Robots}('Bliss', 'Pick', p_2, j))\}$$

   Note that all temporal variables are of type `second`.

2. "Did Dan and Bliss ever perform the same task during the same minute?"

$$\{i{:}\texttt{minute} | \exists j_1{:}\texttt{second} \exists j_2{:}\texttt{second} \exists y \exists p_1, p_2 (\texttt{Robots}('Dan', y, p_1, j_1) \wedge \texttt{Robots}('Bliss', y, p_2, j_2) \wedge$$
$$\textbf{IntSec}_{\texttt{second},\texttt{minute}}(j_1, i) \wedge \textbf{IntSec}_{\texttt{second},\texttt{minute}}(j_2, i))\}$$

3. "Find the time when the workshop state is first repeated." (See the explanation in Example 4.) Let $\psi(i, j) = i < j \wedge \forall r \exists y, p_1, p_2 (\texttt{Robots}(r, y, p_1, i) \wedge \texttt{Robots}(r, y, p_2, j))$. That is, $\psi(i, j)$ is true if each robot is performing the same task (different robots may perform differently). Now the query can be expressed as follows:

$$\{i | \exists i' (\psi(i', i) \wedge \forall i'' (i'' < i \rightarrow \neg \exists i''' \psi(i''', i'')))\}$$

   Note all the temporal types are in `second`.

Note that the first two queries are in $\text{TCALC}_1^{1,1}$ and the last one is in $\text{TCALC}_2^{1,1}$. We conjecture that the last query cannot be expressed in $\text{TCALC}_1^{1,1}$. □

6 Data-domain independent $\text{TCALC}_k^{m,n}$ and its equivalence to $\text{TALG}_k^{m,n}$

In this section, we assume k, m and n are fixed positive integers such that $m \leq k$ and $n \leq k$.

As a well-known fact, an unrestricted relational calculus query may lead to infinite answers. Since TCALC_k can express all relational calculus queries, unrestricted TCALC_k queries may lead to infinite answers. In this section, we define a safe segment of TCALC_k. We use the following version of data-domain independent queries. First, we need the notion of active data domains:

Definition Given a set **M** of temporal modules and a TCALC_k query Q, then the *active domain of* **M** *and* Q, denoted $adom(\mathbf{M}, Q)$, is the set of values in the data domain that appear in some temporal module in **M** or in Q.

Note that a data value v appears in a temporal module $(R, \mu, p, \varphi, \tau)$ if v is in a tuple t, where $t \in \varphi(i_1, \ldots, i_p)$ for some positive integers $i_1, \ldots, i_p$. Note that the active domain is a set of values from the data domain. The time instances are not involved.

Definition A query $Q = \{x_1, \ldots, x_p, i_1, \ldots, i_q : \mu | \psi\}$ is said to be *data-domain independent* if (1) all the data values in the result of the query is from the active data domain (of this query and the involved temporal modules), and (2) the result of the query would not change if for each subformula $\exists x \psi'$ or $\forall x \psi'$ of ψ, where x is a data variable, the range of values considered were only the active data domain.

The above definition is similar to that on pages 151–152 [Ull88]. However, here we only talk about the data variables, instead of all variables. Thus, in evaluating a data domain independent query, we only need to consider the data values from the active data domain (which is finite) as the range of data variables. The range for time variables is still the whole set of positive integers.

We are now ready to show that following.

Theorem 1 The data-domain independent $\text{TCALC}_k^{m,n}$ is equivalent to $\text{TALG}_k^{m,n}$ in expressiveness. Hence, the data domain independent TCALC_k is equivalent to TALG_k in expressiveness.

Proof. (Sketch) We need to construct an equivalent query in the $\text{TCALC}_k^{m,n}$ calculus for each query in the $\text{TALG}_k^{m,n}$ algebra, and vise versa. (Here, "equivalent" means that the queries return the same result module if the input database modules are the same.) The direction from algebra to calculus is straightforward. We only need to note that at each intermediate step, since the result module is at most k dimensional, the number of free time variables is at most k. The direction from calculus to algebra is only a little more involved. We need to take each subformula that starts with a quantification on a data variable and convert it to an equivalent algebraic expression. We do this recursively from the bottom up. If there is no quantification on data variables, then it is easily seen that general time operations (plus perhaps a number of snapshot-wise operations) will suffice. Since at each of these steps, there are only at most k free time variables, we only need a k-dimensional temporal module to store the intermediate results. □

As an example, let the following be a $\text{TCALC}_1^{1,1}$ query

$$\{x, i : \mathtt{day} \mid \mathtt{M}_1(x, i) \wedge \exists j : \mathtt{day}(j \leq i \wedge \mathtt{M}_2(x, j))\}$$

This query intuitively picks up the tuples $\langle a \rangle$ of $\mathtt{M}_1$ at tick i if the same tuple $\langle a \rangle$ is at some earlier tick of $\mathtt{M}_2$. This query compares facts at two different ticks. The following is an equivalent algebra query:

$$\mathtt{M}_2 \cap [\![f]\!](\mathtt{M}_1)$$

where $f(i) = \exists j(j \leq i \wedge p_1(j))$. Intuitively, $[\![f]\!]$ "extends" the validity of tuples in $\mathtt{M}_1$ and picks up the common tuples from $\mathtt{M}_2$ and this extended $\mathtt{M}_1$. The parallel structure of the algebra query and the calculus query is apparent.

As another example, consider the following $\text{TCALC}_2^{1,1}$ query:

$$\{1 : \mathtt{day} \mid \forall i_1 : \mathtt{day} \forall i_2 : \mathtt{day} \exists x(\mathtt{M}(x, i_1) \wedge \mathtt{M}(x, i_2))\}$$

Intuitively, the query returns a non-empty set if for all possible pairs of days, you can always find a common tuple in these two days. (Kamp used such a query to show that $\text{TCALC}_1^{1,1}$ is *not* equivalent to $\text{TCALC}_2^{1,1}$ [Kam71], *if* the domain of the time ticks is the rational numbers and the temporal modules are allowed to be infinite.) The equivalent TALG$_2^{1,1}$ query is as follows:

$$\pi[\![f_1]\!]([\![f]\!](\mathtt{M}, \mathtt{M})),$$

where $f(i,j) = p_1(i) \wedge p_2(j)$, $f_1(i') = (\forall i \forall j p(i,j)) \wedge i' = 1$. (We omit the types, namely `day`, of the time variables.) Intuitively, f organize timestamps into pairs and f_1 just checks to see if all pairs are there. (Note the temporal module scheme for $[\![f]\!](\mathtt{M}, \mathtt{M}))$ is 2 dimensional.) The projection π gets rid of all data attributes from the resulting temporal module.

As a last note of this section, safety can be defined via a syntactic restriction similar to [BWBJ95]. We omit that here.

7 Adding aggregation functions to TALG

In many situations, information that is on one temporal type is converted to be on another temporal type via certain aggregation or other functions. For example, given a temporal module $(\langle \mathtt{Name}, \mathtt{Income}\rangle, \mathtt{month}, \varphi, \tau)$, to find the total income of a person for a particular year, we need to sum up all the income figures of the months belonging to that year. To support such a query, we need to perform aggregation functions based on temporal types.

An aggregation function maps a subset of the data domain to a single value of the data domain. For example, **sum**, **avg** (average), **count** etc are aggregation functions. Note that we differ here from [Tan87]. In [Tan87], an aggregation function maps a set of pairs of the form (v, i), where v is a data value and i a timestamp. Since timestamps are included in the set, the aggregation function in [Tan87] perform many time related tasks. For example, **last** is an aggregation function in [Tan87] that retrieves the last tuple from the set. In this paper, we use aggregation functions in the traditional sense, namely a function that maps a set of values to a single value. Any time related tasks will be performed by other operations in the algebra. For instance, Example 2 gives a query that retrieves the last tuple (of the first minute). Obviously, if the temporal type `minute` is replaced by the temporal type `TOP`,[6] then the query in Example 2 will retrieve the very last action for Bliss.

Following [Klu82], we assume that we have an aggregation function g_A for each (data) attribute A of a temporal module, where g is a regular aggregation function such as **sum** etc. The semantics is that the values under the attribute (regardless of the timestamps) will be used as the input to the aggregation function.

Assume $M = (R, \mu, n, \varphi, \tau)$ is a temporal module and g an aggregation function.

Aggregation. If A is an attribute of R, $X \subset R - \{A\}$, and ν a temporal type, then $\alpha_{g,A,X,\nu}(M) = (XA, \nu, n, \varphi', \tau')$, where

$$\varphi'(j_1, \ldots, j_n) = \{t | \exists i_{11}, \ldots, i_{1n}, \exists t_1 (t_1 \in \varphi'(i_{11}, \ldots, i_{1n}) \wedge \\ t[X] = t_1[X] \wedge t[A] = g_A(\sigma_{X=t_1[X]}(M_{j_1,\ldots,j_n}))$$

[6]The type `TOP` is defined as follows: $\mathtt{TOP}(1) = \mathcal{N}$ and $\mathtt{TOP}(i) = \emptyset$ for each $i \geq 2$.

where $M_{j_1,\ldots,j_n} = (R, \mu, n, \varphi', \tau') = [\![\wedge_{1 \le p \le n} \textbf{IntSec}_{\mu,\nu}(i_{2p}, j_p)]\!](M)$.

The temporal module $M_{j_1,\ldots,j_n}$ is by selecting the tuples from M those that have timestamps intersect with $j_1, \ldots, j_n$. The above formula says that a tuple t is in the tick $(j_1, \ldots, j_n)$ of the resulting module if its values for the attributes X are from a tuple in $M_{j_1,\ldots,j_n}$ and its value for A is the result of the aggregation function g applying on attribute A (i.e., g_A) of all the tuples t_2 in $M_{j_1,\ldots,j_n}$ such that $t_2[X] = t[X]$. The attributes in $R - (X\,A)$ are dropped from the temporal module.

In other words, attributes X and the tick $(j_1, \ldots, j_n)$ of ν serve to partition the tuples of the original module. The tuple t_1 at tick $(i_1, \ldots, i_n)$ of μ and the tuple t_2 at tick $i'_1, \ldots, i'_n$ of μ are in the same group iff $t_1[X] = t_2[X]$ and $\textbf{IntSec}_{\mu,\nu}(i_p, j_p) = \textbf{IntSec}_{\mu,\nu}(i'_p, j_p) = \texttt{true}$ for each $p = 1, \ldots, n$. The aggregation function is then applied to each group on the attribute A values. For each such group, there is only one tuple returned.

Notice that if the temporal type ν in the above definition is the same as μ, the aggregation functions here are used in same way as in the algebra of [Klu82]. That is, the way we use aggregation functions is a proper extension of that in [Klu82].

As an example, the following query finds the power consumptions by Wuyi in each minute:

$$\alpha_{\mathbf{sum},PowerConsumption,Robot,\texttt{minute}}(\sigma_{Robot='Wuyi'}(\texttt{Robots})).$$

8 Conclusion

In this paper, we introduced temporal algebras on a temporal database model that incorporate multiple temporal types. The novel features of the algebra include its separation of two groups of operations, its multidimensionality and its aggregation functions. The separation of the operations of operations into two groups makes the algebra more intuitive and gives rise natural manipulation of multiple temporal types. We conjecture that the multidimensionality increases the expressive power of the algebra, even if the input and output are all of one dimensional. The aggregation functions in the algebra take advantage of the multiple temporal types to group tuples.

To simplify the presentation, we defined our temporal types on the discrete absolute time line. We believe that continuous absolute time can be incorporated without much difficulty. Another simplification we made was that each multidimensional temporal module is of one temporal type. This restriction can be easily lifted by giving each dimension a different temporal type. The algebra and calculus query languages given in this paper can easily been modified to work on such temporal modules.

This paper also introduced the notion of data-domain independence. The purpose is to restrict the range that data variables must go through in order to evaluate the query. The time domain still remains infinite. And obviously, certain queries will have infinite answers in the time dimension. One solution to this problem is to set an upper limit on the time dimension. Another is to study the structure of these infinite answers. In [BWBJ95], the notion of "first-order finitely partitioned" is introduced as such an alternative.

Regarding future research, we believe a proof or disproof of our conjecture that $\text{TALG}_k^{1,1}$ does not collapse is important not only theoretically, but also for practical purposes. Indeed, if the hierarchy does collapse, we can simplify our algebras tremendously. Another important area is the evaluation and optimization of TALG expressions. Designing a calculus with aggregation functions is also of some interest, which we believe can be achieved following the line of [Klu82].

References

[BWBJ95] C. Bettini, X. Wang, E. Bertino, and S. Jajodia. Semantic assumptions and query evaluation in temporal databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*. ACM, 1995. To appear.

[CCT94] J. Clifford, A. Croker, and A. Tuzhilin. On completeness of historical relational query languages. *ACM Transactions on Database Systems*, 19(1):64–116, March 1994.

[Cho94] J. Chomicki. Temporal query languages: A survey. In D.M.Gabbay and H.J. Ohlbach, editors, *Temporal Logic, First International Conf.*, pages 506–534, Bonn, Germany, 1994. Springer-Verlag.

[CI94] J. Clifford and T. Isakowitz. On the semantics of (bi)temporal variable databases. In *Proc. EDBT*, pages 215–230, March 1994.

[CR87] J. Clifford and A. Rao. A simple, general structure for temporal domains. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 23–30, France, May 1987.

[CSS94] R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars and temporal rules in next generation databases. In *Proceedings of the International Conference on Data Engineering*, 1994.

[DS94] C. E. Dyreson and R. T. Snodgrass. Temporal granularity and indeterminacy: Two sides of the same coin. Technical report, Computer Science Department, University of Arizona, February 1994. TR 94-06.

[JSS94] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying temporal data models via a conceptual model. *Information systems*, 19(7):513, 1994.

[Kam71] H. Kamp. Formal properties of 'now'. *Theoria*, 37:227–273, 1971.

[Klu82] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of ACM*, 29(3):699–717, July 1982.

[NS92] M. Niezette and J.-M. Stevenne. An efficient symbolic representation of periodic time. In *Proc. of First International Conference on Information and Knowledge management*, 1992.

[SA85] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In S. Navathe, editor, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 236–246, Austin, TX, May 1985. ACM.

[SGM93] R. T. Snodgrass, S. Gomez, and L. E. McKenzie, Jr. Aggregates in the temporal query language TQuel. *IEEE Transactions on Knowledge and Data Engineering*, 5(3):826–842, October 1993.

[SJS94] M.D. Soo, C. S. Jensen, and R. T. Snodgrass. An algebra for TSQL2. A part of the TSQL2 commentaries, see [TSQ94], September 1994.

[SS92] M.D. Soo and R. Snodgrass. Multiple calendar support for converntional database management systems. Technical Report 92-7, Computer science department, University of Arizona, Feb 1992.

[Tan87] A. U. Tansel. A statistical interface for historical relational databases. In *Proc. Data Engineering*, pages 538–546, February 1987.

[TCG+93] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal databases: Theory, design, and implementation*. Benjamin/Cummings, 1993.

[TSQ94] Announcement: The temporal query language TSQL2 final language definition. SIGMOD Record, September 1994. Vol. 23, No. 3.

[Ull88] J. D. Ullman. *Priciples of Database and Knowledge-base Systems*. Computer Science Press, 1988.

[WJS95] X. Wang, S. Jajodia, and V.S. Subrahmanian. Temporal modules: An approach toward federated temporal databases. *Information Sciences*, 82:103–128, 1995. A preliminary version of this paper appeared in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, DC, May 1993, pp. 227–236.