INFORMATIQUE THÉORIQUE ET APPLICATIONS

MEHMET ALI ORGUN

Incorporating an implicit time dimension into the relational model and algebra

Informatique théorique et applications, tome 30, n° 3 (1996), p. 231-260

<http://www.numdam.org/item?id=ITA_1996__30_3_231_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 30, n° 3, 1996, pp. 231-260)

INCORPORATING AN IMPLICIT TIME DIMENSION INTO THE RELATIONAL MODEL AND ALGEBRA (*)

by Mehmet Ali ORGUN ([1])

Communicated by Christian CHOFFRUT

Abstract. – *The prevailing approach to modeling the time dimension in databases is, in one form or another, the use of an explicit representation of time. In this paper we propose a temporal relational model and algebra, based on temporal semantics, to incorporate an* implicit *time dimension in databases. In temporal semantics, time-varying relations are an indexed collection of ordinary relations, one for each moment in time. A temporal database is modeled by a collection of time-varying relations. Temporal databases are queried using a temporal relational algebra (TRA) which extends the relational algebra point-wise upon the set of natural numbers. Although TRA lacks the ability to explicitly manipulate time, we show how temporal aggregation and when-type queries can be formulated using a technique called "tagging". The formal properties of TRA, which can be used for query optimization, are also outlined. We also compare our work with other proposed temporal algebras.*

Keywords: *Temporal Databases, Temporal Algebras, Temporal Logic, Formal Semantics, Temporal Aggregation, Algebraic Properties of Operators, Temporal Queries.*

Résumé. – *L'approche la plus répandue pour la modélisation de la dimension temporelle dans les bases de données est, dans une forme ou une autre, l'utilisation d'une représentation et d'une manipulation explicite du temps. Dans cet article, nous proposons un modèle et une algèbre temporels et relationnels fondés sur la sémantique temporelle pour introduire une dimension temporelle* implicite *dans les bases de données. Dans la sémantique temporelle, les relations temporelles sont des ensembles ordonnés de relations, chacune représentant un moment différent dans l'évolution de la base de données. Une base de données temporelle est modélisée par un ensemble de relations temporelles. Les données temporelles sont extraites en utilisant une algèbre temporelle relationnelle (TRA) qui étend l'algèbre relationnelle point à point avec l'ensemble des nombres naturels. Quoique TRA ne soit pas en mesure de manipuler explicitement le temps, nous démontrons comment l'agrégation temporelle et les requêtes de type "when" peuvent être formulées en utilisant une technique dite* d'étiquettage. *Les propriétés formelles de TRA, pouvant être utilisées pour l'optimisation de requêtes, sont aussi présentées. Nous comparons aussi TRA avec certaines autres propositions d'algèbres temporelles.*

Mots clés : *Bases de données temporelles, algèbres temporelles, logique temporelle, sémantique formelle, agrégation temporelle, propriétés algébriques des opérateurs, requêtes temporelles.*

(*) Received October 31, 1994.

([1]) Department of Computing, Macquarie University, Sydney, NSW 2109, Australia Tel: +61 2 850-9570, Fax: +61 2 850-9551, E-mail:mehmet@mpce.mq.edu.au

AMS Subject Classification: 68 P 15 Database Theory.

1. INTRODUCTION

The relational algebra [8] operates on a model in which each relation reflects the current reality as it is best known of the enterprise being modeled. The model cannot deal with the notion of a history, or how each relation has evolved into what it is now. Having recognized the need to incorporate the time dimension into the relational model, a significant number of research efforts have been directed towards studying various aspects of this problem. The prevailing approach to modeling the time dimension in databases is, in one form or another, the use of an explicit representation of time at both the tuple level and the attribute level [6, 7, 11, 26]. Included in that effort is temporal extensions of the relational algebra. The current status of research on temporal databases can be found in the collection by Tansel *et al.* [31], and Kline [17] provides a recent bibliography.

Most of the temporal algebras reported in the literature are not algebras in the mathematical sense, that is, they are not *closed* since expressions do not always evaluate to time-varying relations of the underlying models. This is one of the consequences of the use of an explicit time dimension in databases: the operators provided in these algebras are in fact designed to exploit actual representations of time-varying data and to manipulate complex time-varying attributes and temporal elements. Some algebras are not unisorted; in other words, the results of expressions can be time-varying relations, snapshot relations, or even temporal elements such as intervals. For example, Clifford's [6] and Sarda's [26] algebras include an operator to drop the time components of a given time-varying relation to obtain *snapshot* relations, and Gadia's [11] algebra allows temporal elements. Very few algebras [20, 32] support the standard definitions of algebraic operators such as intersection and θ-join.

McKenzie and Snodgrass [21] give an extensive evaluation of temporal algebras, and they outline a number of criteria for the comparison of algebras. Some of the important criteria are: (1) a temporal algebra should be a consistent extension of the relational algebra, (2) its formal semantics is well-defined, (3) it is an algebra, (3) it supports basic algebraic equivalences, (4) it reduces to the relational algebra (over moments in time), (5) it supports the standard definitions of intersection ($\cap$), θ-join, natural join ($\bowtie$), and quotient ($\div$), and (6) it includes aggregates. Most of the proposed algebras do not satisfy all of these criteria, and only a few algebras include aggregates because of the complicated interval semantics. These criteria are our starting point.

In this paper, we consider an extension of the relational model based on temporal semantics. In the underlying relational model for temporal databases, time-varying relations are an (infinite) collection of finite relations indexed by moments in time. As a query language for the model, we introduce a temporal (relational) algebra, which we call TRA, as a *point-wise* extension of the relational algebra. The algebra TRA is a revised version of a temporal algebra, originally proposed by Orgun and Müller [23]. The algebra TRA by design has the relational algebra as a special case for each moment in time; hence it inherits all the properties of the relational algebra. Time-varying data from different moments in time are joined through the use of temporal operators, not by the use of explicit references to time. In other words, time is *implicit* in the model and algebra. However, temporal databases of the underlying model can only model *valid-time* [14].

The temporal relational model corresponds to a representation of temporal data based on tuples extended with moments in time as time-stamps, hence it satisfies another criterion of McKenzie and Snodgrass [21]. In Clifford *et al.* [5], temporal models that support 1NF time-varying relations, such as our temporal relational model, are called *temporally ungrouped* data models, as opposed to *temporally grouped* data models. Therefore TRA can be classified as a temporally ungrouped algebra. The model may introduce a fair amount of redundancies, but the representation at the implementation level need not be based on this abstract model; it can use event-based interval time-stamps to save space. If the representation is *homogeneous* [11], that is, attribute values coexist in any given tuple, we can switch from an event-based interval representation at the tuple or attribute level to time-stamping of tuples and *vice versa.*

Aggregation of time-varying data is an important operation in temporal databases, but it is supported by a few of the proposed algebras (e.g., Tansel [29]). Other attempts to provide temporal aggregation include TQUEL of Snodgrass [27], and the query language of Wuu and Dayal [9], which is based on an object-oriented model supporting multi-sets. TRA also supports point-wise extensions of the standard aggregation operators such as `sum`, `avg`, `max`, and so on. Another important operation in temporal databases is to ask queries to find times of when things happen. Since time dimension is implicit in TRA, it lacks the ability to directly formulate when-type queries. Therefore we provide an "indirect" method called *tagging* for manipulating time, which can be used to mimic when-type queries and formulate temporal aggregation across time. Tags are similar to user-defined time-stamps, hence they are transparent to the algebra. Gabbay and McBrien [10] proposed a

similar technique for expressing when-type queries. Another approach is discussed by Lorentzos and Johnson [18].

The structure of the paper is as follows. Section 2 outlines the temporal data model TRA is based on. In section 3 we discuss TRA in more detail, and show that it is a consistent extension of the relational algebra. Section 4 gives examples of temporal selection, and of combining time-varying data by temporal intersection, union, and splicing. In section 5, we discuss *tagging* in detail. In section 6, we outline the formal (algebraic) properties of TRA which can be used in query optimization. In particular we show that TRA inherits the properties of the relational algebra such as monotonicity, associativity and distributivity as well as the standard definitions of defined operators such as $\cap$, ($\bowtie$), and $\div$. In section 7, we relate our work to other approaches to temporal algebras and discuss future work.

2. TEMPORAL DATA MODEL

The notion of a time-varying relation is not well-defined. In most of the reported works, time-stamps and event-based interval-stamps are usually used to represent time-varying data at both the tuple and attribute levels. As in the recent works of Tuzhilin and Clifford [32], Gabbay and McBrien [10], and Orgun and Müller [23], we seek the answer in temporal logic [2]. The meaning of a predicate symbol in temporal semantics is a mapping from moments in time to relations over a given universe of discourse (domain). The collection of moments in time is usually a discrete and linearly ordered set with an unbounded future, such as the set ω of natural numbers with its usual ordering relation *earlier-than*, $<$. The elements of ω can be interpreted uniformly in terms of hours, days, weeks and so on, depending on the intended application.

In our model, a time-varying relation is a mapping, just like the meaning of a predicate symbol in temporal semantics. Let U denote a given universe of discourse (domain), U^n the n-folded cross product of U, and $\mathcal{P}(U^n)$ the power set (the set of subsets) of U^n. Then a time-varying relation over the domain U is defined as follows.

DEFINITION 1: *A time-varying relation with arity* n ($n \geq 1$) *is an element of* $[\omega \longrightarrow \mathcal{P}(U^n)]$, *that is, an element of the set of functions from* ω to $\mathcal{P}(U^n)$.

Given a time-varying relation tr, we write $tr = \langle tr(0), tr(1), tr(2), \cdots \rangle$ where for all $t \in \omega$, $tr(t) \in \mathcal{P}(U^n)$. We also have an ordering relation between time-varying relations defined as follows.

DEFINITION 2: *Let r and s be time-varying relations with the same arity. We write $r \sqsubseteq s$ if and only if for all $t \in \omega$, $r(t) \subseteq s(t)$.*

Let Rel be a countable set of relation symbols and U a given domain over which relations are defined. We require that U have at least one element and be countable. In practice, we may require that U be finite. The set of temporal databases over U, denoted as $\mathcal{DB}$, is defined as follows:

$$\mathcal{DB} =_{df} [Rel \rightarrow \bigcup_{n>0} [\omega \rightarrow \mathcal{P}(U^n)]].$$

It is also the case that $\mathcal{DB}$ is a complete lattice induced by the ordering relation $\sqsubseteq$. In more technical terms, members of $\mathcal{DB}$ are called *temporal interpretations* [2]. For any given temporal database $db \in \mathcal{DB}$ and an element p of Rel with arity n, the meaning of p in db is a time-varying relation given as

$$db(p) \in [\omega \rightarrow \mathcal{P}(U^n)].$$

As complex time-varying attributes or (event-based) intervals are not employed in the abstract model, time-varying relations are in the First Normal Form (1NF) [19, 33]. 1NF is also preserved in some of the other temporal models, for example, those of Lorentzos and Johnson [18] and Jensen and Mark [15]. In Clifford *et al.* [5], temporal models that support 1NF time-varying relations are called *temporally ungrouped* data models. We now give an example of a temporal database.

Example 1: This database for a university keeps personal and professional data for faculty members and the departments they work for; it is a modified version of a database given in [16]. It consists of four relations:

`dept` (DEPT, SECRETARY, HEAD)
`pers` (NAME, SEX, STATUS, ADDRESS)
`prof` (NAME, DEPT, RANK, SALARY)

`publ` (NAME, JOURNAL, ISSUE)

Here `dept` holds information about each department, `pers` holds personal information for each faculty member, `prof` holds professional information for each faculty member, and `publ` holds information for publications of each faculty member. We have that `dept`, `pers`, `prof`, `publ` $\in$ *Rel*. Attribute names are self-explanatory. The DEPT attribute is the key for `dept` relation, and the NAME attribute for the `pers`, `prof`, and `publ` relations. The attribute HEAD is also used for faculty member names.

We assume that the database came into existence at time 0. Given a $db \in \mathcal{DB}$, it is possible that $db(\texttt{prof})$ is the time-varying relation whose portion from time 0 to time 3 is depicted in figure 1. In this example the logical time is interpreted as months. ■

$0 \longmapsto$

Ali	Phil	Assist	2400
Carol	CompSci	Assoc	2750

$1 \longmapsto$

Ali	Maths	Assist	2500
Carol	Maths	Assoc	2750
Joy	CompSci	Assist	2350

$2 \longmapsto$

Ali	Maths	Assist	2500
Carol	Maths	Assoc	2750
Joy	CompSci	Assist	2350

$3 \longmapsto$

Ali	Phil	Assoc	2900
Carol	Maths	Full	3250
Joy	CompSci	Assist	2450
Kim	Ling	Full	3500

Figure 1. – The `prof` relation from time 0 to time 3

Since the abstract model does not preclude any form of temporally grouped representation of time-varying data, we can use time-stamped tuples, event-based intervals, time-varying attributes and so on (or even a non-uniform representation within the same model). The choice for the representation does not affect TRA or its properties, because the temporal operators of TRA are not designed to exploit actual representations of time-varying data (*see* below). The representation only affects how TRA is going to be implemented. However, it should be noted that a realistic implementation of TRA is a challenging task due to the level of abstraction the algebra offers.

3. TEMPORAL RELATIONAL ALGEBRA

This section first introduces the temporal relational algebra TRA, and then provides the denotational semantics of its expressions. In this paper, we build on the work of Orgun and Müller [23]. Their algebra, also called TRA, is based on the temporal model outlined in section 2. TRA introduced four

temporal operators, namely, `first`, `next`, `prev`, and `fby[.]`. Informally, the operator `first` refers to the initial moment in time (time 0), `next` refers to the next moment in time, and `prev` refers to the previous moment in time. The binary `fby[.]` is a parameterized cut-and-paste operator on time-varying relations. In our temporal relational algebra, we keep `first` and `next` as primitive operators, as well as a binary `fby` operator with simplified semantics. In the revised algebra, `prev` and parameterized `fby[.]` of TRA can be defined in terms of other operators. Therefore the revised algebra has a smoother semantics than the original TRA, without any loss of expressive power.

From here on, we refer to the revised TRA simply as TRA. Example uses of the operators of the algebra are given in sections 4 and 5.

3.1. Point-wise Extensions

TRA is a point-wise extension of the relational algebra upon ω enriched by adding temporal operators. In particular, the signature of TRA includes all operators of the relational algebra [8], and its universe is the set of time-varying relations $\cup_{n>0}[\omega \rightarrow \mathcal{P}(U^n)]$. Let $\bigtriangledown$ be a unary operator of the relational algebra, and $\hat{\bigtriangledown}$ the corresponding point-wise operator of TRA. For any given time-varying relation tr, the following holds:

$$\text{For all } t \in \omega, \quad \hat{\bigtriangledown}(tr)(t) = \bigtriangledown(tr(t)).$$

We can give similar equations for the binary operators. Yaghi [34] introduced the notion of a point-wise operation in the context of intensional algebras.

We have the following lemma which states that the meaning of point-wise operators is the same at all moments in time. Therefore we can tell what a point-wise operator does to its argument by looking at the results from each moment in time. The lemma can be naturally extended to include binary point-wise operators.

LEMMA 1: *Let* $\hat{\bigtriangledown}$ *be a unary point-wise operator of* TRA. *For all time-varying relations* r *and* s *with the same arity, and for all* $t, u \in \omega$, if $r(t) = s(u)$, *then* $\hat{\bigtriangledown}(r)(t) = \hat{\bigtriangledown}(s)(u)$.

Proof: Let r and s be time-varying relations with the same arity, and t and $u \in \omega$. Suppose that $r(t) = s(u)$. Then we have that $\hat{\bigtriangledown}(r)(t) = \bigtriangledown(r(t))$ and $\hat{\bigtriangledown}(s)(u) = \bigtriangledown(s(u))$. Given that $r(t) = s(u)$, then it follows that $\bigtriangledown(r(t)) = \bigtriangledown(s(u))$ by the semantics of point-wise operators. ■

3.2. Operators

The operators of TRA can be classified under three headings: (1) point-wise operators, (2) temporal operators, and (3) aggregation operators. In the sequel, we adopt the infix notation for binary operators.

We first define the notion of a comparator. Given $x, y \geq 1$, the following are comparators: $x < y$, $x \leq y$, $x > y$, $x \geq y$ and $x \neq y$. We call x and y "comparator values". A comparator value is basically the ordinal number of an attribute in a relation. Given a temporal database *db*, we also allow comparators with one of their operands selected from the universe of *db*. A comparator formula is constructed out of conjunctions, disjunctions and comparators. A detailed discussion on the notions of a comparator and comparator formula can be found in Maier [19, section 3.5].

Point-wise Operators

Standard point-wise operators are $\hat{\cap}$, $\hat{\cup}$, $\hat{\times}$, $\hat{-}$, $\hat{\pi}_X$, and $\hat{\sigma}_F$. Here X is a finite sequence of comparator values, and F is a comparator formula. All of $\hat{\cap}$, $\hat{\cup}$ and $(\hat{-})$ take two arguments with the same arity (compatible time-varying relations). At any given time t, the outcome of a point-wise operation depends *only* on the values of its arguments at time t. For instance, given the expression $r \,\hat{\cap}\, s$ where r and s are time-varying relations, the resulting time-varying relation is the point-wise intersection of r and s, that is,

$$r \,\hat{\cap}\, s = \langle r(0) \cap s(0), r(1) \cap s(1), r(2) \cap s(2), \cdots \rangle,$$

or, using the familiar λ-notation, $r \,\hat{\cap}\, s = \lambda t.r(t) \cap s(t)$.

For any given expression of the form $\hat{\pi}_X r$ where r is a relation with arity n, it must be the case that n is not less than the maximum comparator value in X. For any given expression of the form $\hat{\sigma}_F r$, it must be the case that n is not less than the maximum comparator value in F.

Temporal Operators

There are three (primitive) temporal operators in TRA: unary `first` and `next`, and binary `fby`. The temporal operator `first` freezes a time-varying relation at its initial value; `next` is the "tomorrow" operator; and `fby` (read as *followed by*) does temporal splicing, that is, cutting and pasting of time-varying relations. In general, at any given time t, the outcome of a temporal operation may depend on the values of its arguments possibly at time t and at other moments in time.

The formal semantics of temporal operators are given as follows:

- $\texttt{first}(r) = \lambda t.r(0)$.
- $\texttt{next}(r) = \lambda t.r(t+1)$.
- $r \texttt{ fby } s = \lambda t.\begin{cases} r(0), & t = 0 \\ s(t-1), & t > 0. \end{cases}$

where r and s are compatible time-varying relations.

We now define another temporal operator, `prev` in terms of `fby` as follows:

$$\texttt{prev } (r) =_{df} \emptyset \texttt{ fby } r$$

The formal semantics of `prev` can be given using the semantics of `fby`. For the sake of clarity, we assume the following definition [23]:

- $\texttt{prev}(r) = \lambda t.\begin{cases} \emptyset, & t = 0 \\ r(t-1), & t > 0 \end{cases}$

In other words, `prev` is the "yesterday" operator. The value of any expression of the form $\texttt{prev}\, A$ at time 0 is the empty relation, because we cannot go into the past beyond time 0.

From here on, we use the notation `next[`n`]` and `prev[`n`]` as syntactic sugar for n-folded applications of `next` and `prev`. In case $n = 0$, `next[`n`]` and `prev[`n`]` are the empty string.

Aggregation Operators

Let x be a comparator value ≥ 1. The aggregation operators of TRA are $\hat{\texttt{sum}}_x$, $\hat{\texttt{avg}}_x$, $\hat{\texttt{count}}$, $\hat{\texttt{max}}_x$ and $\hat{\texttt{min}}_x$ with their obvious interpretations. These point-wise operators are applied to time-varying relations, and produce unary time-varying relations whose snapshots are single-valued relations. For instance, given the expression $\hat{\texttt{avg}}_x(tr)$ where tr is a time-varying relation whose arity is not less than x, the resulting time-varying relation is

$$\hat{\texttt{avg}}_x(tr) = \langle \texttt{avg}_x\, tr(0), \texttt{avg}_x\, tr(1), \texttt{avg}_x\, tr(2), \cdots \rangle,$$

or in the λ-notation: $\hat{\texttt{avg}}_x(tr) = \lambda t.\texttt{avg}_x(tr(t))$. Here $\texttt{avg}_x(r) = \{\langle e \rangle\}$ where e is the average of the values of the x^{th} attribute from the relation r.

We also assume that aggregation operations can be associated with a *by-list*, producing several tuples determined by calculating the aggregate over a subset of the relation [27]. Examples of temporal aggregation are given in section 5.

3.3. Denotational Semantics

The following discussion is a more formal exposition to TRA. Given a temporal database db, an expression E over db contains only those relation symbols defined in db, and values from the domain of db. Let $[\![E]\!](db)$ denote the meaning (denotation) of E with respect to db. We assume that all expressions are legal, *i.e.*, arities of arguments given in an expression match with respect to the operations involved, and so do the types of attributes over which aggregation operations are performed.

For a relation symbol $p \in Rel$ with arity n, we have that

$db(p) \in [\omega \rightarrow \mathcal{P}(U^n)]$ where U is the domain of db. Thus $[\![E]\!](db)$ is also an element of $[\omega \rightarrow \mathcal{P}(U^k)]$ for some $k \in \omega$. In general, given an expression E, we have that

$$[\![E]\!] \in [\mathcal{DB} \rightarrow \bigcup_{n>0} [\omega \rightarrow P(U^n)]].$$

The following is the formal definition of the denotation function $[\![.]\!]$.

DEFINITION 3: *Let $db \in \mathcal{DB}$, and A and B be* TRA *expressions. The denotations of each kind of expressions of* TRA *are defined as follows.*

1. $[\![p]\!](db) = db(p)$ where $p \in Rel$.
2. $[\![\hat{\bigtriangledown} A]\!](db) = \hat{\bigtriangledown} [\![A]\!](db)$ *where* $\hat{\bigtriangledown}$ *is any unary operator.*
3. $[\![A \hat{\bigtriangledown} B]\!](db) = [\![A]\!](db) \hat{\bigtriangledown} [\![B]\!](db)$ *where* $\hat{\bigtriangledown}$ *is any binary operator.*

For TRA to be an algebra in the mathematical sense, we need to show that all operations of TRA are *closed*; in other words, the denotation function $[\![.]\!]$ assigns a time-varying relation to each (legal) expression.

THEOREM 2: *The denotation function $[\![.]\!]$ is well-defined, that is, for any legal expression E over a temporal database db, $[\![E]\!](db) \in \cup_{n>0} [\omega \rightarrow \mathcal{P}(U^n)]$.*

Proof: The expression E can be of the form (1) p where $p \in Rel$, (2) $\hat{\bigtriangledown} A$ where $\hat{\bigtriangledown}$ is a unary operator, or (3) $A \hat{\bigtriangledown} B$ where $\hat{\bigtriangledown}$ is a binary operator. (Items 1, 2 and 3 in the definition 3.) We can prove the theorem by induction where item 1 is the basis case:

1. Let n be the arity of p. By definition, we have that

$db(p) \in [\omega \rightarrow \mathcal{P}(\mathcal{U})^n]$. Hence $[\![p]\!](db)$ is a time-varying relation.

2. By induction hypothesis $[\![A]\!](db)$ is a time-varying relation. Let $tr = [\![A]\!](db)$. If $\hat{\bigtriangledown}$ is a point-wise operator, we have that $\hat{\bigtriangledown}\, tr = \lambda t.\ \bigtriangledown\ tr(t)$, which is a time-varying relation. If $\hat{\bigtriangledown}$ is a unary temporal operator, then $\hat{\bigtriangledown}\, tr$ is a time-varying relation by the semantics of temporal operators.

3. The proof of item 3 is very similar to that of item 2. ■

We assume the usual point-wise definitions of *quotient* ($\hat{\div}$), *θ-join* ($\hat{\bowtie}_F$), and *natural join* ($\hat{\bowtie}$) in terms of other point-wise operators of TRA; *see* [33] for details. Intersection is employed as a primitive operator in TRA. Note that all the defined operators are also point-wise. We can import the definitions of these point-wise operators, because all the properties of the relational algebra are preserved in TRA (*see* section 6).

We state the following theorem for completeness which says that the original TRA [23] and the revised TRA have the same expressive power. The semantics of `fby[.]` of the original TRA is given as follows. Let $z \in \omega$.

$$\bullet\ r\ \texttt{fby}\ [z]\ s = \lambda t. \begin{cases} r(t), & t \leq z \\ s(t), & t > z \end{cases}$$

THEOREM 3: *All of the primitive operators of the original* TRA *can be defined in the revised* TRA *and* vice versa. *Therefore they are equivalent in expressive power.*

Proof: Both `first` and `next` are amongst the primitive operators of the original and revised TRA, with the same semantics. Therefore it suffices to show that (i) `prev` and `fby[.]` of the original TRA can be defined using the operators of the revised TRA; and (ii) `fby` of the revised TRA can be defined using the operators of the original TRA.

Proof of (i)*:* We have given the definition of `prev` already; the definition of the parameterized `fby[.]` is given in section 4.3. It can be easily verified that the definitions conform to the semantics of `prev` and `fby[.]`.

Proof of (ii)*:* The definition of `fby` for the original TRA can be given as follows:

$$r\ \texttt{fby}\ s =_{df} r\ \texttt{fby}\ [0]\ (\texttt{prev}\ s).$$

Again, it can easily be verified that the definition conforms to the semantics of `fby`. ■

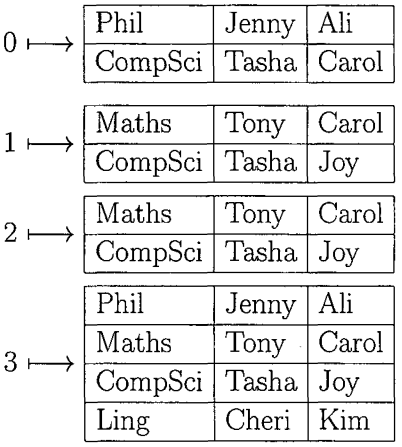

Figure 2. – The `dept` relation from time 0 to time 3

4. EXAMPLES: QUERYING TIME-VARYING DATA

In the following, we stipulate that all queries be evaluated at a given moment t in time (say, "now", "today", "this month" and so on) or over an event-based interval identified by its starting and ending times. In the examples given below, the logical time is interpreted as "months". The evaluation of a query over an interval is performed by evaluating the query at all moments over the interval. We regard an interval as a window through which time-varying data may be queried. This is simply a computability requirement: given a finite domain U, time-varying relations over the domain are finite at particular moments in time or over intervals, but infinite in the aggregate. Some of the examples in this and the next section are based on variations of the benchmark queries posed by Kalua and Robertson [16].

4.1. Temporal Selection

In many applications, a snapshot of time-varying data at a particular moment in time is of interest. Queries of this kind are time-specific. Suppose that we are given the temporal database from example 1. It is possible that $db(\texttt{dept})$ is the time-varying relation whose portion from time 0 to 3 is depicted in figure 2. From here on, we refer to the attributes in a given relation by their ordinal numbers. For instance, NAME in `pers` is the 1^{st} attribute.

Example 2: Consider the query "What department did Kim head in month 3?"

$$\hat{\pi}_{3,1}(\hat{\sigma}_{3=\text{"Kim"}}(\texttt{first next[3] dept}))$$

The temporal operator(s) `first next[3]` move the context to month 3. Since Kim headed in the Linguistics department in month 3, the answer is the relation $\{\langle\ Kim, Ling\ \rangle\}$ whenever the query is evaluated. Let E be the expression given above. Then we have that

$$[\![E]\!](db) = \lambda t.\{\langle Kim, Ling\rangle\}. \quad \blacksquare$$

Sometimes we are interested in what happens relative to a reference point in time, say "now", "today" and so on. For instance, we may pose queries like "Is Ali going to get a raise next month?" In such a query, there is no explicit reference to time, thus the outcome of the query depends on *when* it is evaluated. Queries about the future are not problematic, because, if future data is not available yet, the answer would be the empty relation.

It is also possible to formulate queries about the past.

Example 3: Consider the query "Who was Joy's department head last month?".

$$\texttt{prev}(\hat{\pi}_{1,2,5}\,(\hat{\pi}_{1,2}\,(\hat{\sigma}_{1=\text{"Joy"}}\,\texttt{prof}) \underset{2=1}{\hat{\bowtie}}\, \texttt{dept}))$$

Here $\hat{\bowtie}_F$ is the point-wise θ-join operator. The comparator formula $2 = 1$ means that we compare the second and first attribute values from the corresponding tuples.

At time 0, the answer is the empty relation; at times 1 and 2, the answer is the relation $\{\langle Joy, CompSci, Joy\rangle\}$; and so on. We could also ask "Who will be Joy's department head next month?" by replacing `prev` by `next` in the expression. $\blacksquare$

4.2. Temporal Intersection and Union

We may also be interested in what happened over a period of time in a given relation. The period of interest may be time-specific or time-relative. In the following, we first introduce a notation for representing periods of time, that is, event-based intervals. Let $x, y \geq 0$.

- Time-specific intervals are denoted as $[x, y]$ where $x \leq y$.
- Time-relative intervals are denoted as $[(+/-)x, (+/-)y]$ where $(+/-)x \leq (+/-)y$. The following are time-relative intervals: $[-3, +5]$, $[-6, -1]$ and $[+2, +4]$. The actual interval for any given time-relative interval is in fact

determined by the time of evaluation, say now. For instance, the time-relative interval $[+2, +4]$ represents the time-specific interval $[now + 2, now + 4]$.

The expression $[x, y]E$ is called *temporal intersection* and it is interpreted as finding all those tuples in E that are common at every moment in time over the interval $[x, y]$. For instance, the expression $[-1, +1]E$ is a shorthand notation for

$$\texttt{prev}\ E \hat{\cap} E \hat{\cap} \texttt{next}\ E.$$

Example 4: Consider the (time-relative) query "Find all those faculty members in the Mathematics department who did not get a raise in the last two months".

$$\hat{\pi}_{1.2}([-2, -1](\hat{\sigma}_{2=\text{"Maths"}}\ \texttt{prof}))$$

If any faculty member in the Toys department did not get a raise over the interval $[-2 + now, -1 + now]$, the corresponding faculty member tuples would be the same. That is why we can formulate the query by temporal intersection. ■

We can also "accumulate" time-varying data from different moments in time by *temporal union*. We use a different notation for intervals over which time-varying data will be joined by unions. Let $\langle x, y\rangle$ denote an interval defined just like $[x, y]$. Then the expression $\langle 1, 3\rangle E$ is a syntactic sugar for

$$\texttt{first next}\ E \hat{\cup}\ \texttt{first next}[2]\ E \hat{\cup}\ \texttt{first next}[3]\ E.$$

Example 5: Consider the query "How many (distinct) departments did Kim head from months 1 to 12?" This query can be answered by temporal union as follows:

$$\texttt{count}(\langle 1, 12\rangle(\pi_1\ (\sigma_{3=\text{"Kim"}}\ \texttt{dept})))$$

Suppose that Kim headed three departments from months 1 to 12. Then we would only obtain three tuples by temporal union $\langle 1, 12\rangle$, because duplicate tuples are not supported in the underlying model. ■

We can also make use of the time of evaluation to restrict the range of temporal intersections and unions.

Example 6 [Kalua and Robertson [16] query QB3]: Consider the query "What departments were headed by Carol and Kim and who were their

secretaries?" We assume that the query is restricted to the interval from the beginning up to "now", or up to the time of evaluation of the query:

$$\langle 0, now \rangle (\hat{\sigma}_{3="Carol" \vee 3="Kim"} \texttt{dept})$$

This query is time-relative in the sense that the value of *now* will change and so will the result, depending on the time of evaluation. It is the responsibility of the implementation to replace *now* with the time of evaluation during the query evaluation process. ■

4.3. Temporal Splicing

In some cases, we are interested in cutting out a portion of a time-varying relation, and pasting a portion of another one in its place. This is achieved by the binary operator `fby`. We now consider a more general form of temporal splicing, that is, a parameterized `fby` operator defined in terms of the (primitive) operators of TRA as follows:

- r `fby[0]` $s =_{df} r$ `fby (next` s`)`,
- r `fby[`t`]` $s =_{df} r$ `fby ((next` r`) fby[`$t-1$`] (next` s`))`,

for $t > 0$.

Read r `fby[`t`]` s as "cut r at time t, and paste s to it from time $t+1$ on". We call time t as the *cut-point* for `fby[`t`]`. Given two time-varying relations r and s, then we have that r `fby[`t`]` $s = < r_0, r_1, \ldots, r_t, s_{t+1}, s_{t+2}, \ldots >$. We employ the parameterized `fby[.]` operator as a primitive operator in TRA as in the original TRA [23], because it has a straightforward implementation.

Example 7: Suppose that a 10% overall salary increase is planned as from month t and the university management wants to study its long term effects on the university. The temporal database is now extended with a new `prof` relation, say `projected-prof`, modeling the salary increase starting at month t. For analysis, all references to the `prof` relation before month t should be regarded as references to the original `prof` relation, and at and after month t as references to the `projected-prof` relation. In time-relative queries, it is impossible to determine which relation should be used since there are no explicit references to time. A solution to this problem can be provided by "temporal splicing". The following TRA expression can be used whenever the faculty member relation is required:

`prof fby[`$t-1$`] projected-prof.`

The operator $\texttt{fby}[t-1]$ guarantees that `prof` is used up to and including the cut point at $t-1$, and `projected-prof` from time t onwards. ■

5. EXPLICIT MANIPULATION OF TIME

It is very natural, especially in temporal databases, to ask queries to find times of when things happened. Since there is no explicit manipulation of time in expressions of TRA, such queries cannot be directly formulated. This is a tradeoff for the level of abstraction TRA offers. We adopt an indirect, but effective method, called *tagging*, which can be used to manipulate time as any other attribute in TRA. Through tagging, data coming from different moments in time can be identified so that temporal aggregation using standard point-wise aggregation operators becomes possible, and certain types of when-type queries can also be formulated.

5.1. Tagging and Temporal Aggregation

Suppose that we are interested in the average salary of a faculty member across time. A naïve solution would be to use temporal union followed by the aggregation operation $\hat{\texttt{avg}}_x$. There are problems with this solution, because the relational model does not support duplicate tuples (or *multi-sets*), but we need all the duplicate tuples to obtain a correct answer. For instance, the average salary of Ali from time 0 to time 3 is \$2575 (*see Fig.* 1). But using temporal union and aggregation as in the following expression

$$\hat{\texttt{avg}}_4(\langle 0,3\rangle(\hat{\sigma}_{1=\text{"}Ali\text{"}}\,\texttt{prof})),$$

we obtain the wrong result of \$2600. The problem is that the information about Ali stored in the `prof` relation is the same in months 1 and 2. What is needed is the ability in TRA to differentiate between tuples coming from different moments in time.

We provide a solution to the problem using the notion of a *tag*. Tags are basically time-stamps on data. Suppose that the following time-varying relation is given:

$$u = \langle\{\langle 0\rangle\},\{\langle 1\rangle\},\{\langle 2\rangle\},\cdots\rangle.$$

Then we can define a new point-wise operator, say `tag`, as follows:

- $\texttt{tag}(r) =_{df} r \times u$.

Tags are used much in the same way as *user-defined time* [14], but they are not stored in the database, rather they are dynamically created on demand during the evaluation of TRA expressions. An expression of the form $\mathtt{tag}(E)$ can be evaluated by tagging each tuple resulting from the evaluation of E by the time of evaluation. As a result, temporal union on tagged time-varying relations will not "lose" any time-sensitive information. Ali's average salary can now be obtained as

$$\hat{\mathtt{avg}}_4(\langle 0,3\rangle\ \mathtt{tag}\ (\hat{\sigma}_{1=\text{"}Ali\text{"}}\ \mathtt{prof})).$$

Gabbay and McBrien [10] also considered answering when-type queries using a special relation called *time*. This relation is identical to the relation u with an upper limit imposed on it. A similar approach is discussed by Lorentzos and Johnson [18]. There, a calendar relation is introduced, which consists of a finite relation of unary tuples, one for each moment in time. It is used when time-varying relations based on time-stamping with intervals at the tuple level are unfolded into time-varying relations based on tuple time-stamping.

Below is another example of the use of tagging.

Example 8: Consider the query "Did Laura earn more than her department head in the last three months (including *now*)?" This is a time-relative *yes/no* query; therefore we just provide as answer a binary tuple with Laura's and the total salary of her department head for the period. The query in TRA may be formed in two parts: an expression to find the total salary Laura earned, and an expression to find the total salary Laura's department head earned. Here is the first part, call it A:

$$\hat{\mathtt{sum}}_4(\langle -2,0\rangle\ \mathtt{tag}\ (\hat{\sigma}_{1=\text{"}Laura\text{"}}\ \mathtt{prof})).$$

Here is the second part, call it B:

$$\hat{\mathtt{sum}}_4(\langle -2,0\rangle\ \mathtt{tag}\ (\hat{\pi}_3\ ((\hat{\pi}_2\ (\hat{\sigma}_{1=\text{"}Laura\text{"}}\ \mathtt{prof}))\ \hat{\bowtie}_{1=1}\ \mathtt{dept})\ \hat{\bowtie}_{1=1}\ \mathtt{prof})).$$

The final expression is obtained by selection from the cross product of A and B, that is, $\hat{\sigma}_{1>2}(A \times B)$. Regardless of whether Laura worked under different department heads over the interval $\langle -2,0\rangle$ or not, the salaries of all those department heads would be summed. ■

5.2 Expressing When-Type Queries

When tagging is directly applied to the base relations (those relations stored in the temporal database), tags represent valid-time. The following examples show the way in which tags can be manipulated, so that certain types of when-type queries can also be formulated.

Example 9: Consider the query "Find all the months in which Joy was the head of the Computer Science department."

$$\hat{\pi}_{3,1,4}\left(\langle 0, now\rangle \,\mathtt{tag}\,(\hat{\sigma}_{1=\text{"CompSci"}\wedge 3=\text{"Joy"}}\,\mathtt{dept})\right)$$

Again, duplicate data coming from different moments in time are preserved in the combined relation owing to tagging. If the time of evaluation is 2, the answer is the relation $\{\langle Joy, CompSci, 1\rangle, \langle Joy, CompSci, 2\rangle\}$. ■

Example 10 [Kalua and Robertson [16] query QB8]: Consider the query "When did the associate professors attain this rank?"

$$\langle 0, now\rangle(\hat{\pi}_{1,7,9}\,(\mathtt{prev(prof)} \underset{1=1\wedge 3\neq 3}{\hat{\bowtie}} \hat{\sigma}_{3=\text{"Assoc"}}\,\mathtt{tag(prof)}))$$

We compare the rank of faculty members in consecutive months and if the rank has changed from "anything" to Associate Professor in any given month, the tagged faculty member tuple will be retained in the temporal union. The final result can be obtained by projection. If the time of evaluation is 3, the answer is the relation $\{\langle$ *Ali, Assoc,* 3 $\rangle\}$, because only Ali attained this rank in the period from time 0 to time 3. In order to include those faculty members who were associate professors at time 0, we can take union of the above query with the query

$$\mathtt{first}(\hat{\pi}_{1,3,5}(\hat{\sigma}_{3=\text{"Assoc"}}\mathtt{tag(prof)})).$$

Then the answer, again at time 0, is the relation

$$\{\langle Ali,\ Assoc,\ 3\rangle,\ \langle Carol,\ Assoc,\ 0\rangle\}.$$ ■

With tagging, aggregation operators can be utilized to express a variety of when-type queries. The following example shows the use of `count` with a *by-list.*

Example 11 [Kalua and Robertson [16] query QB7]: Consider the query "List all faculty who published in the same journal at least twice, along with the journal issues and publication dates (months)." We first formulate

an expression (call it A) to find the names of faculty members and journals that satisfy the condition of the query.

$$\hat{\pi}_{1,2}(\hat{\sigma}_{3\geq \text{"2"}}(\texttt{count}((\langle 0, now\rangle\ \texttt{tag}\,(\texttt{publ}))\ \texttt{by}\ 1,2)))$$

Here the resulting relation after `count` has the attributes: NAME, JOURNAL, and the number of tuples for each distinct pair of NAME and JOURNAL. We refer to these two attributes in the *by-list* using their corresponding ordinal numbers. Final query is formed by θ-join of A and a (sub-)query to obtain a relation with the names of faculty members, journals, issue numbers, and dates:

$$\hat{\pi}_{1,2,5,6}(A \underset{1=1\wedge 2=2}{\hat{\bowtie}} \langle 0, now\rangle\ \texttt{tag}\,(\texttt{publ})).$$

We could easily define another query for finding the total number of publications per month for each department in the university using `count` and *by-list* on tagged relations (this would be an instance of a group-by-time aggregation). ■

6. ALGEBRAIC PROPERTIES OF TRA

This section discusses some of the important algebraic properties of TRA. Many query optimization strategies involve transforming algebraic expressions into ones that lead to more efficient query execution plans [19, chapter 11]. An analogous strategy can also be employed for query optimization in temporal databases, and TRA can naturally serve as an appropriate target for a temporal query language processor.

We show that TRA inherits all the properties of the relational algebra such as the commutative laws, associative laws, and distributive laws of point-wise operators [1, 33, 19]. It is also shown that temporal operators distribute over the point-wise operators, and that there are laws governing the interaction between temporal operators. We assume that `prev` and the parameterized `fby[.]` operator are amongst the primitive operators of the algebra, because they are likely to be used frequently in expressions and they have straightforward implementations.

6.1. Inherited Properties and Monotonicity

An important property of operators is *monotonicity*. Monotonicity implies that if we know more information about the argument of an operator, we shall

know no less about the result. It is known that the operators of the relational algebra except the set difference are monotonic, and that monotonicity is closed under function composition [1, 33]. Monotonic operators are *well-behaved.*

Below is the formal definition of the property of monotonicity:

DEFINITION 4: *Let r and s be two time-varying relations with the same arity, and $\hat{\bigtriangledown}$ be a unary operator of* TRA. *We say that $\hat{\bigtriangledown}$ is monotonic if and only if it is the case that $\hat{\bigtriangledown}(r) \sqsubseteq \hat{\bigtriangledown}(s)$ whenever $r \sqsubseteq s$.*

Since TRA is a point-wise extension of the relational algebra, the monotonicity result immediately extends to the point-wise operators. In fact, TRA inherits all properties of the relational algebra, because the meaning (properties) of point-wise operators can be reduced to the meaning (properties) of the corresponding operators in the relational algebra. Thus, we have the following theorem.

THEOREM 4: *The properties of monotonicity, commutativity, distributivity and associativity are preserved under point-wise extensions.*

Proof: It suffices to show that the properties hold for each moment in time for the operators of the relational algebra corresponding to the point-wise operators of TRA. We omit the details. ■

Theorem 4 guarantees that, for instance, the definitions of quotient ($\div$) and θ-join can be imported to TRA, and that all the algebraic properties of the relational algebra which can be used in query optimization are still valid. In fact, the theorem is valid for *any* point-wise extension of the relational algebra over any domain, not just for TRA. In short, point-wise extensions enable technology transfer.

We also have that all temporal operators of TRA are monotonic.

THEOREM 5: *All temporal operators of* TRA *are monotonic.*

Proof: We show that `next` is monotonic. Given time-varying relations r and s where $r \sqsubseteq s$, we have that $\texttt{next}(r) \sqsubseteq \texttt{next}(s)$ if and only if for all $t \in \omega$, $\texttt{next}(r)(t) \subseteq \texttt{next}(s)(t)$ if and only if for all $t \in \omega$, $r(t+1) \subseteq s(t+1)$ by the semantics of `next`. Given that $r \sqsubseteq s$ holds, then we conclude that `next` is monotonic. Similar arguments apply to all the other temporal operators. ■

It can also be shown that in TRA monotonicity is closed under function composition. This result is well-known in programming language

semantics [28], and also for the operators of the relational algebra [1]. Therefore we have the following result.

THEOREM 6: *In* TRA, *monotonicity is closed under composition.*

6.2. Distributivity of Point-wise and Temporal Operators

Theorem 4 states that a point-wise extension of the commutative laws holds in TRA. We now investigate the interaction (distributive laws) between temporal and point-wise operators.

LEMMA 7: *The temporal operators* `first`, `next`, `prev`, `fby`, *and* `fby[.]` *distribute over the point-wise operators. In other words, let* $\hat{\bigtriangledown}$ *be any unary temporal operator,* $t \geq 0$, *and* $r, s, u, v \in [\omega \rightarrow P(U^n)]$ *for some* $n > 0$. *Then*

a) $\hat{\pi}_X(\hat{\bigtriangledown}(r)) = \hat{\bigtriangledown}(\hat{\pi}_X r)$,
$\hat{\pi}_X(r \texttt{ fby } s) = (\hat{\pi}_X r) \texttt{ fby } (\hat{\pi}_X s)$ *and for* `fby[`t`]`.

b) $\hat{\sigma}_F(\hat{\bigtriangledown}(r)) = \hat{\bigtriangledown}(\hat{\sigma}_F r)$,
$\hat{\sigma}_F(r \texttt{ fby } s) = (\hat{\sigma}_X r) \texttt{ fby } (\hat{\sigma}_F s)$ *and for* `fby[`t`]`.

c) $\hat{\bigtriangledown}(r \mathbin{\hat{\cap}} s) = (\hat{\bigtriangledown} r) \mathbin{\hat{\cap}} (\hat{\bigtriangledown} s)$,
$(r \mathbin{\hat{\cap}} s) \texttt{ fby } (u \mathbin{\hat{\cap}} v) = (r \texttt{ fby } u) \mathbin{\hat{\cap}} (s \texttt{ fby } v)$ *and for* `fby[`t`]`.
Similarly for $\hat{\cup}$, $\hat{\times}$ *and* $\hat{-}$.

Proof: We outline the proof of item (a). Suppose that $\hat{\bigtriangledown}$ is `first`. Then it suffices to show that, for all $t \in \omega$, we have that

$$(\hat{\pi}_X(\texttt{first}(r)))(t) = (\texttt{first}(\hat{\pi}_X r))(t).$$

We proceed as follows:

$$\begin{array}{lll}
\text{(i)} & (\hat{\pi}_X(\texttt{first}(r)))(t) = \pi_X((\texttt{first}(r))(t)) & \text{(point-wise } \pi_X\text{)} \\
& = \pi_X((\lambda z.r(0))(t)) & \text{(semantics of first)} \\
& = \pi_X(r(0)) & \text{(function application)} \\
\text{(ii)} & (\texttt{first}(\hat{\pi}_X(r)))(t) = (\lambda z.(\hat{\pi}_X(r))(0))(t) & \text{(semantics of first)} \\
& = (\hat{\pi}_X(r))(0) & \text{(function application)} \\
& = \pi_X(r(0)) & \text{(point-wise } \pi_X\text{)}
\end{array}$$

Both sides of the equation yield the same result. Similar proofs can be given for the other distributivity properties by making use of the semantics

of temporal operators, the semantics of point-wise operators, and function application. We omit the details. ■

6.3. Interaction Between Temporal Operators

We now investigate the interaction between temporal operators of TRA, and in particular the conditions under which temporal operators can be eliminated. The following lemmas state *some* important properties of the temporal operators.

LEMMA 8: *Let* $r \in [\omega \rightarrow \mathcal{P}(U^n)]$ *for some* $n > 0$. *The following are theorems of* TRA.

1. $\texttt{first}(\texttt{prev}(r)) = \lambda t.\emptyset$.
2. $\texttt{next}(\texttt{first}(r)) = \texttt{first}(r)$.
3. $\texttt{first}(\texttt{first}(r)) = \texttt{first}(r)$.
4. $\texttt{next}(\texttt{prev}(r)) = r$.

Proof: We outline the proof. Item 1 is implied by the semantics of `first` and `prev`. Item 2 can be proved as follows: Let $r = \langle r(0), r(1), r(2), \cdots \rangle$ be a time-varying relation. Given the definition of `first` as $\lambda r.\lambda t.r(0)$, we have that $\texttt{first}(r) = \langle r(0), r(0), r(0), \cdots \rangle$. Then, as `next` shifts a given time-varying relation to the left (and drops its first element), we have that $\texttt{next}(\texttt{first}(r)) = \langle r(0), r(0), r(0), \cdots \rangle$. We omit the proofs of the remaining items. ■

The intuitive understanding of these properties is given as follows: Item 1 is a straightforward consequence of the semantics of `prev` and `first`, and also shows us how to create an empty relation. Item 2 implies that $\texttt{first}(r)$ is an invariant of time. Item 3 is the cancellation axiom of `first` followed by `first`. Item 4 says that shifting a time-varying relation to the right and then immediately to the left has no effect.

The following lemma states the conditions under which `fby`'s can be distributed and/or eliminated.

LEMMA 9: *Let* $r, s, v \in [\omega \rightarrow \mathcal{P}(U^n)]$ *for some* $n > 0$ *and* $t, x \geq 0$. *The following are theorems of* TRA.

1. $\texttt{first}(r\ \texttt{fby}\ s) = \texttt{first}(r)$.
2. $\texttt{next}(r\ \texttt{fby}\ s) = s$.
3. $\texttt{first}(r\ \texttt{fby}[t]\ s) = \texttt{first}(r)$, *regardless* of t.
4. $r\ \texttt{fby}[t](s\ \texttt{fby}[x]\ v) = r\ \texttt{fby}[t]\ v$, *when* $t \geq x$.
5. $r\ \texttt{fby}[t](s\ \texttt{fby}[x]\ v) = (r\ \texttt{fby}[t]\ s)\ \texttt{fby}[x]\ v$, *when* $t \leq x$.

The first three items are consequences of the semantics of `first`, `next`, `fby`, and `fby[.]`. Item 4 asserts that, in nested applications of `fby[.]`'s with non-increasing cut points, only the first cut point matters. Item 5 states the condition under which `fby[.]` is associative.

There are some other properties of TRA which tell us when we cannot simplify an expression any further. For instance, it can be shown that `prev` does not distribute over `fby[.]`; and that `next` and `prev` are not complete inverses, that is, $\texttt{prev}(\texttt{next}(r)) \neq \texttt{next}(\texttt{prev}(r))$. The operators `next` and `prev` would be complete inverses where time is infinite at both ends.

6.4. The Property of Conjunctivity

Intersection is employed as a primitive operator in TRA, but it could also be introduced as a defined point-wise operator. Intersection has a useful property which we call *conjunctivity*. This property is also exhibited by some other point-wise operators and all the temporal operators.

DEFINITION 5: *Let* $\hat{\bigtriangledown}$ *be an operator of* TRA. *We say that* $\hat{\bigtriangledown}$ *is "conjunctive" if and only if* $\hat{\bigtriangledown}(r \hat{\cap} s) = \hat{\bigtriangledown}(r) \hat{\cap} \hat{\bigtriangledown}(s)$.

In other words, a conjunctive operator commutes with (or can be distributed over) $\hat{\cap}$, which also reaffirms that $\hat{\cap}$ is a point-wise operator. The following result is a straightforward corollary to theorem 4 and lemma 7.

COROLLARY 10: *The following operators have the conjunctivity property:* $\hat{\cap}$, $\hat{\pi}_X$, $\hat{\sigma}_F$, `first`, `next`, `prev`, `fby`, *and* `fby[.]`. *In other words, let* $r, s, u, v \in [\omega \rightarrow \mathcal{P}(U^n)]$ *for some* $n > 0$ *and* $t \geq 0$. *Then we have that*

1. $(r \hat{\cap} s) \hat{\cap} (u \hat{\cap} v) = (r \hat{\cap} u) \hat{\cap} (s \hat{\cap} v)$.
2. $\hat{\pi}_X(r \hat{\cap} s) = (\hat{\pi}_X r) \hat{\cap} (\hat{\pi}_X s)$.
3. $\hat{\sigma}_X(r \hat{\cap} s) = (\hat{\sigma}_X r) \hat{\cap} (\hat{\sigma}_X s)$.
4. $\texttt{first}(r \hat{\cap} s) = (\texttt{first}\ r) \hat{\cap} (\texttt{first}\ s)$.
5. $\texttt{next}(r \hat{\cap} s) = (\texttt{next}\ r) \hat{\cap} (\texttt{next}\ s)$.
6. $\texttt{prev}(r \hat{\cap} s) = (\texttt{prev}\ r) \hat{\cap} (\texttt{prev}\ s)$.
7. $(r \hat{\cap} s)\ \texttt{fby}\ (u \hat{\cap} v) = (r\ \texttt{fby}\ u) \hat{\cap} (s\ \texttt{fby}\ v)$.
8. $(r \hat{\cap} s)\ \texttt{fby[}t\texttt{]}\ (u \hat{\cap} v) = (r\ \texttt{fby[}t\texttt{]}\ u) \hat{\cap} (s\ \texttt{fby[}t\texttt{]}\ v)$.

Moreover, conjunctivity implies monotonicity.

THEOREM 11: *All conjunctive operators of* TRA *are also monotonic.*

Proof: We outline the proof for the operator `first`. Let r and s be time-varying relations with the same arity. By corollary 10, we have that $\texttt{first}(r \hat{\cap} s) = (\texttt{first}\ r) \hat{\cap} (\texttt{first}\ s)$. It is clear that if $r \sqsubseteq s$, then $r \hat{\cap} s = r$. Hence $\texttt{first}\ r = (\texttt{first}\ r) \hat{\cap} (\texttt{first}\ s)$, which implies that $\texttt{first}\ r \sqsubseteq \texttt{first}\ s$. Proofs for the other conjunctive operators also make use of Corollary 10 and the fact that $r \hat{\cap} s = r$ whenever $r \sqsubseteq s$ and *vice versa*. ■

In a given expression, we can use the conjunctivity property to push temporal operators inside, and then use the properties given in lemmas 8 and 9 (and others) to simplify the expression further.

7. DISCUSSION

In this section, we relate our work to other proposals for temporal algebras based on valid-time. We also point out further research directions.

7.1. Other Temporal Algebras

Clifford [6] proposed a model and a relational algebra for modeling and querying temporal (historical) databases. The model is based on an extension of the relational model with complex attributes, some of which are functions from moments in time or intervals to attribute values. In other words, the model is a temporally grouped model which does not restrict the relations to first normal form (1NF). In the algebra, the temporal operators are specifically designed to manipulate these complex attributes, and, as a consequence, it is not closed. Also, the algebra does not include aggregates.

Tansel [6] described another model and a temporal algebra to handle the time dimension (for valid time only). Just as in Clifford's approach, Tansel's model strives to minimize data redundancies using intervals and complex time-varying attributes, while at the same time providing a number of algebraic operators to manipulate various representations of time-varying data. The algebra also includes an operator to drop the time components of a given time-varying relation to obtain snapshot relations. It does not support all the algebraic equivalences of the relational algebra, and as a consequence it does not support the standard definitions of operators such as $\cap$, θ-join, and $\bowtie$. Tansel [29] defined an extension of this algebra with aggregates.

Sarda [26] proposed another temporal algebra in which the outcome of expressions may be time-varying relations, intervals and snapshot relations. Sarda also outlined a query language based on an extension of SQL with

temporal operators. It is not clear whether the algebraic properties of the relational algebra carry over Sarda's algebra or not. Gadia's approach [11] is similar to Sarda's. Temporal elements (intervals) are explicitly manipulated in the temporal algebra, as well as time-varying relations, but the definitions of standard algebraic operators are provided. Gadia also considered a calculus-based query language along with the temporal algebra and showed their equivalence.

McKenzie and Snodgrass [20] introduced an extension of the relational algebra with two *rollback* operators that can support both the valid time and transaction time. Rollback operators are used to fetch snapshots of time-varying relations in a given temporal database, which are then manipulated by the operators of the relational algebra. The denotational semantics of operators of the algebra are also given. The underlying model can only support a finite collection of snapshot states. The temporal algebra is also extended with an update operation, and the formal semantics of updates are given. The algebra does not include aggregates.

Lorentzos and Johnson [18] proposed a model and algebra based on tuple and/or attribute level time-stamping. The algebra introduces three new operators, namely, *extend*, *unfold*, and *fold*, defined using the standard operators of the relational algebra. These operators are used to switch from an event-based interval representation of valid-time to a representation based on tuple time-stamping and *vice versa*. In short, time-stamps are regarded as user-defined time, but their special status is recognized by the existence of the three additional operators. The formal semantics of the extended algebra are not given; and it is not clear whether the standard definitions of operators such as θ-join and quotient $\div$ carry over to the algebra. In this algebra, a special relation called calendar is used in unfolding time-varying relations. The use of the calendar relation is in spirit similar to the use of the tag relation u (*see* section 5).

Tuzhilin and Clifford [32] proposed a temporal algebra (called **TA**) as a basis for temporal relational completeness. The algebra includes the operators of the relational algebra plus two temporal linear recursive operators. The algebra is equivalent in expressive power to a temporal calculus based on a temporal logic with **since** and **until**. Their temporal relational model, unlike the underlying model of TRA, is based on discrete and bounded models of time. Just like TRA, Tuzhilin and Clifford's algebra **TA** is a consistent extension of the relational algebra, but **TA** does not have aggregates.

Gabbay and McBrien's [10] considered a refinement of **TA** which is also based on a temporal logic with **since** and **until**. They introduce two linear recursive operators, namely, *since-product* ($S_\times$) and *until-product* ($U_\times$). These operators closely resemble their counterparts in temporal logic. Gabbay and McBrien also considered explicit references to time using a special relation called *time* (as in tagging). They embed their algebra into an extended relational algebra with arithmetic capabilities in select and project operations, but the extended algebra does not include aggregates. They show that the embedding provides a convenient way to translate queries in a temporal extension of SQL into those in standard SQL. The algebraic properties of $S_\times$ and $U_\times$ are not discussed.

There are also some other approaches to temporal databases based on transaction time (time of updates to the database), for example, that of Jensen and Mark [15]. For a comparative study of temporal algebras, we refer the reader to the survey of McKenzie and Snodgrass [21]. Temporal query languages are also surveyed in the work of Chomicki [4].

7.2. Future Work

We are considering extending TRA with "modal" operators to support updates to temporal databases. Modal logic [3] can provide an adequate basis for such an extension. For instance, Golshani [12] proposed a modal extension of a functional model for databases, in which update operations are interpreted as *modal* operators, and the set of possible worlds for such an update modal logic is the set of all possible database states ($\mathcal{DB}$). In an extended TRA, an update operation (say μ) can be regarded as a modal operator; we can take its meaning to be a mapping between temporal databases: $[\![\mu]\!] \in [\mathcal{DB} \rightarrow \mathcal{DB}]$. Further work in this direction involves formulating a modal logic of updates, and investigating the interaction between update operators and temporal operators.

We are also considering a calculus-based temporal query language. It is based on the underlying temporal logic of TRA, in which `first`, `next`, and `fby` are temporal operators. A query in temporal calculus is an expression of the form

$$\{\langle x_1, x_2, \ldots, x_n\rangle \mid \phi(x_1, x_2, \ldots, x_n)\}$$

where ϕ is a *safe* temporal logic formula and x_i's are some of the free variables in ϕ. Each safe query represents a time-varying relation. Just as algebraic expressions, queries are evaluated at a given moment in time, or

over intervals. Here is example 2 in the temporal calculus: "What department did Kim head in month 3?"

$$\{\langle h, d\rangle \mid \texttt{first next[3]}\ \texttt{dept}(d, s, h) \wedge h = \text{``}\textit{Kim}\text{''}\}.$$

The answer to this query is the same whenever it is evaluated; in other words, it is the relation $\{\langle$ *Kim, Ling* $\rangle\}$ at any given moment in time.

The direct correspondence between the temporal operators of TRA and of the calculus can be established, which means that the temporal calculus underlies the expressive power as TRA. The formulation of the temporal calculus will also allow us to study the expressive power of TRA in relation to other proposed temporal algebras.

We are also working on an equational extension of TRA in which recursive queries can be formulated. A significant usage of `fby` is in defining recursive queries, as shown in the following equation

$$tc = r\ \texttt{fby}\ (tc \cup \pi_{1,3}(\texttt{first}\ r \underset{2=1}{\bowtie} tc)).$$

The equation defines an iterative solution to find the transitive closure of a given binary relation represented by r at time 0. The use of `fby` ensures that the initial value of tc is $r(0)$, and the next values of tc is computed from the previous value of tc. Therefore, as time progresses, tc step-by-step approximates to the transitive closure of $r(0)$. This form of recursion is called *linear temporal recursion.* There are recursive extensions of the relational algebra [13], but, to the best of our knowledge, recursion has not been explored in the context of temporal algebras. The algebras of Tuzhilin and Clifford [32] and Gabbay and McBrien [10] each offers two linear recursive operators, but no recursive equations.

We have not yet considered a temporal extension of a query language such as SQL or QUEL. The algebraic properties of TRA make it a suitable target language for optimizing queries in a temporal extension of SQL or QUEL. Temporal versions of SQL include: TSQL of Navathe and Ahmed [22], and TSQL-like query language of Sarda [26] (which has the same expressive power as Sarda's historical algebra). Temporal versions of QUEL include: TQUEL of Snodgrass [27] which also offers temporal aggregates, and HQUEL of Tansel [32] (Tansel's algebra [6] is the target language for optimizing queries in HQUEL).

8. CONCLUSIONS

The main difference between our approach to temporal algebras and many others reported in the literature is that TRA is not an algebra designed to manipulate explicit representations of time-varying data or temporal elements such as intervals. Consequently, it has a very smooth formal semantics. We can easily introduce extra temporal operators into TRA and try out new ideas owing to the level of abstraction offered by it. For instance, Orgun and Wadge [24] proposed the use of the original TRA as an algebraic front-end to a modular, temporal extension of Datalog. Temporal Datalog can naturally express recursion. Further work on Temporal Datalog is reported in Orgun [25].

In short, TRA is a representation-independent temporal relational algebra. Freedom from representation ensures portability, and it enables technology transfer from the established tools and techniques that are available for the relational algebra. Query optimization strategies can be directly based on the formal properties of TRA, and they should be considered in conjunction with actual representations and storage structures.

ACKNOWLEDGEMENTS

This work has been supported in part by a Macquarie University Research Grant (MURG). The work has benefited from discussions with Rajiv Bagai, Lee Flax, and Bill Wadge. Thanks are also due to two anonymous referees for their helpful comments and suggestions, and Joey Paquet for translating the abstract into French.

REFERENCES

1. A. V. AHO and J. D. ULLMAN, Universality of data retrieval languages, In *Proceedings of the Sixth ACM Symposium on Principles of Programming Languages*, pp. 110-120, San Antonio, Texas, ACM Press, 1979.
2. J. P. BURGESS, Basic tense logic, In D. M. Gabbay and F. Guethner, editors, *Handbook of Philosophical Logic, Vol. II*, pp. 89-134, D. Reidel Publishing Company, 1984.
3. B. F. CHELLAS, *Modal Logic: An Introduction*, Cambridge University Press, 1980.
4. J CHOMICKI, Temporal query languages: A survey. In D. M. Gabbay and H. J. Ohlbach, editors, *Proceedings of ICTL'94: The First International Conference on Temporal Logic*, *827* of LNAI, pp. 506-534, Gustav Stresemann Institut, Bonn, Germany, Springer-Verlag, 1994.
5. J. CLIFFORD, A. CROKER and A. TUZHILIN, On completeness of historical relational query languages, *ACM Transactions on Database Systems*, 1994, *19* (1), pp. 64-116.
6. J. CLIFFORD and A. U. TANSEL, On an algebra for historical relational databases: Two views, In S. Navathe, editor, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, pp. 247-265. ACM Press, 1985.

7. J. CLIFFORD and D. S. WARREN, Formal semantics for time in databases, *ACM Transactions on Database Systems*, 1983, *8* (2), pp. 214-254.
8. E. F. CODD, A relational model of data for large shared data banks, *Communications of the Association for Computing Machinery*, 1970, *13* (6), pp. 377-387.
9. U. DAYAL and G. T. J. WUU, A uniform approach to processing temporal queries, In *Proceedings of the 18th Very Large Data Bases Conference*, pp. 407-418, Vancouver, British Columbia, Canada, 1992. Morgan Kauffman, Los Altos, Calif.
10. D. GABBAY and P. MCBRIEN, Temporal logic & historical databases, In *Proceedings of the 17th Very Large Data Bases Conference*, pp. 423-430, Barcelona, Spain, September 1991. Morgan Kauffman, Los Altos, Calif.
11. S. K. GADIA, A homogeneous relational model and query languages for temporal databases, *ACM Transactions on Database Systems*, 1988, *13* (4): pp. 418-448.
12. F. GOLSHANI, Specification and design of expert database systems, In Larry Kerschberg, editor, *Expert Database Systems*, p. 369-381. The Benjamin/Cummings Publishing Company, 1986.
13. M. A. W. HOUTSMA and P. M. G. APERS, Algebraic optimization of recursive queries, *Data & Knowledge Engineering*, 1992, 7, pp. 299-325.
14. C. S. JENSEN *et al.*, A consensus glossary of temporal database concepts, *SIGMOD RECORD*,1994, *23* (1) pp. 52-64.
15. C. S. JENSEN and L. MARK, Queries on change in an extended relational model, *IEEE Transactions on Knowledge and Data Engineering*, 1992, *4*, (2): pp. 192-200.
16. P. P. KALUA and E. L. ROBERTSON, Benchmark queries for temporal databases, *Technical Report TR379,* Computer Science Department, Indiana University, Bloomington, Indiana 47405, USA, March 1993.
17. N. KLINE, An update of the temporal database bibliography, *SIGMOD RECORD*, 1993, *22* (4), pp. 66-80.
18. N. A. LORENTZOS and R. G. JOHNSON, TRA: A model for a temporal relational algebra, In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in Information Systems*, pp. 95-109, North-Holland, Amsterdam, 1988.
19. D. MAIER, *The Theory of Relational Databases*, Computer Science Press, 1983.
20. E. MCKENZIE and R. SNODGRASS, Extending the relational algebra to support transaction time, In U. Dayal and I. Traiger, editors, *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*, pp. 467-478, ACM Press, 1987.
21. L. EDWIN MCKENZIE Jr. and R. SNODGRASS, Evaluation of relational algebras incorporating the time dimension in databases, *ACM Computing Surveys*, 1991, *23* (4), pp. 501-543.
22. S. B. NAVATHE and R. AHMED, TSQL: A language interface for history databases. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in Information Systems*, pp. 109-122. North-Holland, Amsterdam, 1988.
23. M. A. ORGUN and H. A. MÜLLER, A temporal algebra based on an abstract model, In M. E. Orlowska and M. Papazoglou, editors, *Advances in Database Research: Proceedings of the 4th Australian Database Conference*, pp. 301-316, Brisbane, Queensland, Australia, February 1-2 1993. World Scientific, Singapore.
24. M. A. ORGUN and W. W. WADGE, A relational algebra as a query language for Temporal Datalog, In A. M. Tjoa and I. Ramos, editors, *Proceedings of DEXA'92: The Third International Conference on Database and Expert Systems Applications*, pp. 276-281, Valencia, Spain, September 2-4 1992, Springer-Verlag Wien.
25. M. A. ORGUN, On temporal deductive databases. *Computational Intelligence*, 1996, *12* (2), To appear.
26. N. L. SARDA, Algebra and query language for a historical data model, *The Computer Journal*, 1990, *33* (1), pp. 11-18.

27. R. T. Snodgrass, S. Gomez and L. Edwin McKenzie, Jr. Aggregates in the temporal query language TQuel, *IEEE Transactions on Knowledge and Data Engineering*, 1993, *5* (5), pp. 826-842.
28. J. E. Stoy, *Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.
29. A. U. Tansel, A statistical interface for historical relational databases, In *Proceedings of the International Conference on Data Engineering*, pp. 538-546, Los Angeles, Calif., February 1987, IEEE Computer Society Press.
30. A. U. Tansel, A historical query language, *Information Sciences*, 1991, *53*, pp. 101-133.
31. A. U. Tansel *et al.*, editors, *Temporal Databases: Theory, Design, and Implementation*, Benjamin/Cummings Publishing Company, Redwood City, CA, 1993.
32. A. Tuzhilin and J. Clifford , A temporal relational algebra as a basis for temporal relational completeness, In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases*, pp. 13-23, Brisbane, Australia, August 13-16 1990. Morgan Kaufmann Publishers Inc., Los Altos, Calif.
33. J. D. Ullman, *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.
34. A. A. Yaghi, *An Intensional Implementation Technique for Functional Languages*, PhD thesis, Department of Computer Science, University of Warwick, Coventry, England, 1984.