A Recursive Temporal Algebra and Temporal Completeness

Mehmet A. Orgun
Department of Computing
Macquarie University
Sydney, NSW 2109, Australia
http://www-comp.mpce.mq.edu.au/~mehmet

Abstract

This paper introduces a recursive temporal algebra based on temporal semantics for querying time-varying data. The algebra, called $\Re^*$, is based on a temporal relational data model in which a temporal database is modeled as a collection of time-varying relations. Each time-varying relation is a collection of ordinary relations indexed by moments in time. In $\Re^*$, recursive queries (such as the transitive closure of a given relation) can be formulated through equations. It is shown that other forms of recursion, such as linear recursion, can also be expressed using iteration through time. Temporal completeness of $\Re^*$ is established, with respect to two other temporal algebras based on temporal semantics which offer linear recursive operators.

1 Introduction

The relational algebra [8] operates on a model which cannot naturally 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 status of research on temporal databases can be found in the survey of Özsoyoğlu and Snodgrass [21], and in the collection by Tansel et al [26]. Included in that effort are temporal extensions of the relational algebra. However, most of the temporal algebras reported in the literature provide operators to manipulate complex time-varying attributes and temporal elements directly [17].

In order to overcome some other limitations of the relational algebra in expressing certain types of queries (such as the transitive closure of a given relation), recursive extensions of the relational algebra and recursive extensions of SQL-type query languages have also been proposed [1, 12, 15]. For example, Houtsma and Apers [12] employ a μ-calculus with a fixed-point operator to express recursive queries. In the context of temporal databases, Tuzhilin and Clifford [27] propose a temporal algebra with two extra operators of temporal linear recursion. Gabbay and McBrien [9] define a refinement of the algebra of Tuzhilin and Clifford [27]. Neither algebra supports recursive equations.

In this paper we propose a temporal algebra which supports recursive queries through an equational extension. The algebra, called $\Re^*$, is based on an algebra proposed by Orgun and Müller [20]. $\Re^*$ satisfies some of the important criteria proposed by McKenzie and Snodgrass [17]; it is a consistent extension of the relational algebra, it is an algebra in the mathematical sense, it has a well-defined formal semantics, it supports basic algebraic equivalences, and it includes aggregates. Most of the proposed algebras satisfy only a few of these criteria, and do not include aggregates because of complicated interval semantics.

$\Re^*$ is a *pointwise* extension of the relational algebra over the set of natural numbers (that is, the collection of points in time). It is based on temporal semantics, and hence it does not allow an explicit manipulation of time. In the underlying relational model based on temporal semantics, a temporal database is modeled by a collection of time-varying relations, each of which is a collection of ordinary relations indexed by moments in time. $\Re^*$ is also enriched with pointwise extensions of the standard aggregation operators.

The rest of the paper is organized as follows. In section 2, we outline the underlying temporal data model. In Section 3, we describe the operations of $\Re^*$. In Section 4, we discuss recursive equations. In Section 5, it is shown that $\Re^*$ with bounded time is *temporally complete* with respect to the algebras of Tuzhilin and Clifford [27] and Gabbay and McBrien [9]. Section 6 concludes the paper with a brief summary.

2 Temporal Data Model

In our model, a temporal database is modeled by a collection of time-varying relations, each of which is a collection of ordinary relations indexed by mo-

ments in time. The collection of moments in time is modeled by the set of natural numbers, denoted as ω. At this stage, we assume that the interpretation of elements of ω is uniform with respect to the granularity of time. We refer the reader to the algebras proposed by Ladkin [16] and Wiederhold et al [28] which address the issue of time granularity.

Let a *relation scheme* (or just *scheme*) Σ be a finite set of *attribute names*, where for any attribute name $A \in \Sigma$, $dom(A)$ is a non-empty *domain* of values for A. A *tuple* on Σ is any map $t : \Sigma \to \cup_{A \in \Sigma} dom(A)$, such that $t(A) \in dom(A)$, for each $A \in \Sigma$. Let $\tau(\Sigma)$ denote the set of all tuples on Σ.

We have the followins definitions of a relation (of the relational model) and time-varying relation:

Definition 1 A *relation* R on scheme Σ is any subset of $\tau(\Sigma)$. We let $\mathcal{P}(\Sigma)$ be the set of all relations on Σ. □

Definition 2 A *time-varying relation* on scheme Σ is a map $T : \omega \to \mathcal{P}(\Sigma)$. □

Intuitively, $T(0)$ is the value (or *extension*) of T at time 0, $T(1)$ at time 1, and so on. If T and S are time-varying relations on the same scheme, then we say that $T \sqsubseteq S$ if $T(t) \subseteq S(t)$, for all $t \in \omega$.

Let Rel be a countable set of relation symbols, such that for any symbol $p \in Rel$, Δ_p is a scheme. Then a *temporal database* for Rel is a map

$$db_{Rel} : Rel \to \bigcup_{p \in Rel} (\omega \to \mathcal{P}(\Delta_p)),$$

such that $db_{Rel}(p) \in \omega \to \mathcal{P}(\Delta_p)$, for all $p \in Rel$. In other words, a temporal database associates with each relation symbol some time-varying relation on the correct scheme. The $\sqsubseteq$ relation on time-varying relations induces the following $\sqsubseteq$ relation on temporal databases:

$db_{Rel} \sqsubseteq db'_{Rel}$ if for each $p \in Rel$, $db_{Rel}(p) \sqsubseteq db'_{Rel}(p)$.

We let $\mathcal{DB}_{Rel}$ be the set of all temporal databases for Rel; this set can be shown to be a complete lattice under the $\sqsubseteq$ relation.

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 Kalua and Robertson [14]. It consists of four relations:

dept(<u>DEPT</u>, SECRETARY, HEAD)
pers(<u>NAME</u>, SEX, STATUS, ADDRESS)
prof(<u>NAME</u>, DEPT, RANK, SALARY)
publ(<u>NAME</u>, JOURNAL, ISSUE)

$0 \longmapsto$

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

Given a $db \in \mathcal{DB}_{Rel}$, it is possible that db(**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. □

3 Temporal Algebra

This section discusses the basic operators of $\Re^*$. A basic expression of $\Re^*$ consists of relation symbols, and their compositions using basic operators. We assume that expressions are evaluated at particular moments in time, or over event-based intervals. This is a computability requirement: time-varying relations represent finite relations at particular moments in time, but they are infinite in the aggregate.

3.1 Pointwise Operators

An operator of $\Re^*$ is called *pointwise* if the value of any expression involving that operator at any time t depends entirely on the operand values at the same time t. For any n-ary operator Θ on relations defined in the relational algebra [8], we let $\vec{\Theta}$ be an n-ary pointwise operator of $\Re^*$, such that for any time-varying relations $T_1, \ldots, T_n$, $\vec{\Theta}(T_1, \ldots, T_n)$ is the time-varying relation

$$\langle \Theta(T_1(0), \ldots, T_n(0)), \Theta(T_1(1), \ldots, T_n(1)), \ldots \rangle.$$

Using the familiar λ-notation,

$$\vec{\Theta}(T_1, \ldots, T_n) = \lambda t.\, \Theta(T_1(t), \ldots, T_n(t)).$$

Thus, $\vec{\pi}_\Sigma$ and $\vec{\sigma}_F$ are unary operators and $\vec{\cup}$, $\vec{\cap}$, $\vec{-}$, $\vec{\times}$ and $\vec{\bowtie}$ are binary operators of $\Re^*$.

Let $x \geq 1$. The aggregation operators are $\vec{\mathbf{sum}}_x$, $\vec{\mathbf{avg}}_x$, $\vec{\mathbf{count}}$, $\vec{\mathbf{max}}_x$ and $\vec{\mathbf{min}}_x$ with their obvious interpretations. These pointwise operators are applied to time-varying relations with arbitrary arities, and produce unary time-varying relations whose snapshots at particular moments in time are single-valued relations.

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 [24].

Note that since $\Re^*$ is a pointwise extension of the relational algebra, the meaning (properties) of pointwise operators can be reduced to the meaning (properties) of the corresponding operators in the relational algebra. Therefore $\Re^*$ inherits all properties of the relational algebra, such as distributive, associative laws and so on. A detailed discussion on the properties of $\Re^*$ is given in Orgun [19].

3.2 Temporal operators

An operator of $\Re^*$ is called *temporal* if it allows looking into the future or past values of the operands in arriving at the 'current' value of an expression. In other words, temporal operators are used for navigation through time.

The simplest of these is the unary operator **first**, which results in the propagation of the time 0 value of its operand. Thus, for any time-varying relation T, **first** T is the sequence

$$\langle T(0), T(0), T(0), \ldots \rangle,$$

or, using the λ-notation, **first** $T = \lambda t \in \omega.T(0)$.

The unary operator **next** permits looking one step in the future of its operand. So, **next** T is the sequence

$$\langle T(1), T(2), T(3), \ldots \rangle,$$

i.e., **next** $T = \lambda t \in \omega.T(t+1)$. We write **next**$[n]$ for n iterated applications of **next**.

For instance, consider the temporal database from example 1. It is possible that $db($**dept**$)$ is the time-varying relation whose portion from time 0 to 3 is depicted in Figure 2.

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

$$\vec{\pi}_{HEAD,DEPT}(\vec{\sigma}_{HEAD=\text{“}Kim\text{”}}(\textbf{first next}[3]\ \textbf{dept}))$$

Since Kim headed in the Linguistics department in month 3, the answer is the relation $\{\langle Kim, Ling\rangle\}$ at any time at which the query is evaluated. □

$0 \longmapsto$

Phil	Jenny	Ali
CompSci	Tasha	Carol

$1 \longmapsto$

Maths	Tony	Carol
CompSci	Tasha	Joy

$2 \longmapsto$

Maths	Tony	Carol
CompSci	Tasha	Joy

$3 \longmapsto$

Phil	Jenny	Ali
Maths	Tony	Carol
CompSci	Tasha	Joy
Ling	Cheri	Kim

Figure 2: The **dept** relation from time 0 to time 3

Note that the *earlier-than* relation, $<$, on the set ω of all time points is well-founded, i.e. there is no infinite decreasing sequence of time points. If, for example, the sequence of valid time points were $\langle \ldots, -1, 0, 1, \ldots \rangle$, we could have had a unary operator **prev**, the complete inverse of **next**, defined as

$$\textbf{prev}\ T = \lambda t \in \mathcal{Z}.T(t-1),$$

where $\mathcal{Z}$ is the set of all integers. However, such an operator is not possible now, because the function $\lambda t \in \omega.T(t-1)$ is not defined for $t = 0$.

We handle this by introducing a binary operator **fby**, which uses the time 0 value of its first operand only in that special case $t = 0$. Let S and T be time-varying relations on the same scheme. So, S **fby** T is the sequence $\langle S(0), T(0), T(1), T(2), \ldots \rangle$, i.e.

$$S\ \textbf{fby}\ T = \lambda t \in \omega. \begin{cases} S(0) & \text{if } t = 0, \\ T(t-1) & \text{if } t > 0. \end{cases}$$

Now **prev** can be defined in terms of **fby** as follows:

$$\textbf{prev}\ T\ =_{df} \emptyset\ \textbf{fby}\ T$$

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

$$(\vec{\pi}_{NAME,DEPT}(\vec{\sigma}_{NAME=\text{“}Joy\text{”}}\textbf{prof}) \vec{\bowtie} \textbf{dept}))$$

Then we have the following query:

$$\textbf{prev}(\vec{\pi}_{NAME,DEPT,HEAD}\ E)$$

At time 0, the answer is the empty relation; at time 1, the answer is the relation $\{\langle Joy, CompSci, Joy\rangle\}$; and so on. □

3.3 Tagging

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.

For instance, suppose that we are interested in the total salary of Ali from time 0 to time 3 which is \$10300 (see figure 1). A naïve solution would be to take the union of **prof** relation from time 0 to time 3 followed by the aggregation operation $\vec{\mathbf{sum}}_x$. So, we introduce a succinct notation for *temporal union:*

$$\langle x, y\rangle\ E =_{df}\ \vec{\bigcup}_{i=x,\ldots,y}\ (\mathbf{first}\ \mathbf{next}[i]\ E)$$

where $x \leq y$. This leads to the following expression on scheme Σ:

$$\vec{\mathbf{sum}}_{SALARY}\ (\langle 0,3\rangle(\vec{\sigma}_{NAME=\text{“}Ali\text{”}}\mathbf{prof}))\,,$$

which gives the relation $\{\langle 9800\rangle\}$ at any moment in time. The result is not "correct", because the information about Ali stored in the **dept** relation is the same in months 1 and 2, and the relational model does not support duplicate tuples.

We need to be able to differentiate between tuples coming from different moments in time. Hence we provide a solution to the problem using the notion of a *tag*. Suppose that the following time-varying relation on scheme {TIME} is given:

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

We can now define a new pointwise operator, say **tag**, as follows:

- $\mathbf{tag}(T) =_{df}\ T\ \vec{\times}\ U$.

Ali's total salary can now be computed as

$$\vec{\mathbf{sum}}_{SALARY}\ (\langle 0,3\rangle\ \mathbf{tag}\ (\vec{\sigma}_{NAME=\text{“}Ali\text{”}}\mathbf{prof}))\,.$$

Tags are used much in the same way as *user-defined time* [13], but they are only created on the fly when needed. An expression of the form $\mathbf{tag}(E)$ can be evaluated by tagging each tuple resulting from the evaluation of E by the time of evaluation. Observe that tagging can be used together with aggregation operators to answer a variety of when-type queries.

In short, tagging provides a remedy which is completely transparent to $\Re^*$ for the lack of ability in it to manipulate time explicitly. As a result, the formal semantics of $\Re^*$ is not sensitive to the use of tags.

Gabbay and McBrien [9] also considered answering when-type queries using a special relation called *time*. This relation is identical to the relation U except that it is restricted to a last moment in time.

4 Recursive Queries

We now extend basic expressions of $\Re^*$ with equations which can be used to express recursion. As Beeri and Milo [5] point out, equational extensions increase the expressive power of algebras significantly.

An expression of $\Re^*$ consists a basic expression followed by a *where* clause:

$$\begin{array}{l} < \textit{expression} > \\ \quad \mathbf{where} \\ \quad\quad < \textit{definitions} > \\ \quad \mathbf{end} \end{array}$$

where $<$ *definitions* $>$ is a list of *operator* definitions, each of which is of the form:

$$\Theta(X_1, \ldots, X_n) = F(X_1, \ldots, X_n);$$

for some $n \geq 0$. Here Θ is the name of a defined operator (function) of n arguments, and F is a basic expression consisting of relation symbols, arguments X_1 through X_n, and operators. And $<$ *expression* $>$ is a basic expression of $\Re^*$, but it may also contain applications of operators defined in D.

Note that nested **where** clauses are not allowed, because they can be flattened in a straightforward manner. We omit the details.

We stipulate that the operator of each definition be unique. Also for the sake of simplicity in this paper, recursion within a definition is allowed, but *mutual* recursion is disallowed.

In *pointwise recursion* [18], the definition of an operator does not contain any temporal operators. For instance, we can easily define a pointwise transitive closure operator which gives the transitive closure of its argument at any given moment in time.

The operational semantics of the definition of pointwise operators can be provided using a standard iterative evaluation process. For instance, naïve or seminaïve techniques such as Jacobi and Gauss-Seidel algorithms described in Ceri et al [6] can be easily adapted to the evaluation of pointwise operators. There are also certain techniques available to optimize the evaluation of (point-wise) recursive definitions, for example, see [12].

In *linear recursion*, the definition of an operator refers to the values of its arguments in the past. The operator **fby** looks into the past for its second argument, therefore its most common usage is to define linear recursive operators, as in Example 4. Before we give that example, we need to define a scheme renaming operator.

Definition 3 Let R be a time-varying relation on some scheme Σ. Let $\Delta = \{A_1, \ldots, A_n\}$ be a scheme for which there exists a one-one correspondence $\psi : \Delta \to \Sigma$ such that for all i, $1 \leq i \leq n$, $dom(A_i) = dom(\psi(A_i))$. Then

$$R^{A_1,\ldots,A_n}_{\psi(A_1),\ldots,\psi(A_n)}$$

is the following time-varying relation on scheme Δ:

$$\lambda t \in \omega.\{\psi \circ f \mid f \in R(t)\},$$

where $\psi \circ f$ denotes the composition of ψ and f, i.e. $(\psi \circ f)(A) = f(\psi(A))$, for all $A \in \Delta$. □

Example 4 This example is adapted from Bagai and Orgun [2]. Let $G = \langle V, E\rangle$ be a directed graph, whose vertex set V is possibly infinite.

We let Σ be the scheme {S,T}, where $dom(\mathtt{S}) = dom(\mathtt{T}) = V$. Informally, the attribute names S and T stand for 'Source' and 'Target' vertex, respectively. Let *edge* be the following relation on scheme Σ:

$$\{f : \Sigma \to V \mid \langle f(\mathtt{S}), f(\mathtt{T})\rangle \in E\}.$$

In other words, *edge* is the set of all pairs of vertices between which there exists a direct edge. Now let *EDGE* be the (constant) time-varying relation on scheme Σ, given by $EDGE(t) = edge$, for all $t \in \omega$.

We have the following recursive definition of *ltc*:

ltc(*EDGE*)
  where
    $ltc(E) = E$ **fby**
      $(\vec{\pi}_{\{\mathtt{X},\mathtt{Z}\}}(ltc(E)^{\mathtt{X},\mathtt{Y}}_{\mathtt{S},\mathtt{T}} \vec{\bowtie} E^{\mathtt{Y},\mathtt{Z}}_{\mathtt{S},\mathtt{T}}))^{\mathtt{S},\mathtt{T}}_{\mathtt{X},\mathtt{Z}}$;
  end

where X, Y and Z are new attribute names. It can be shown that *ltc(EDGE)*(t) is essentially the set of all pairs of vertices (S, T) such that there exists a path in G of length $t + 1$ from S to T.

The following is a simple modification of the above definition and defines *ltc(EDGE)* to be a time-varying relation that converges to the transitive closure of G.

ltc(*EDGE*)
  where
    $ltc(E) = E$ **fby**
      $(ltc(E) \vec{\cup} (\vec{\pi}_{\{\mathtt{X},\mathtt{Z}\}}(ltc(E)^{\mathtt{X},\mathtt{Y}}_{\mathtt{S},\mathtt{T}} \vec{\bowtie} E^{\mathtt{Y},\mathtt{Z}}_{\mathtt{S},\mathtt{T}}))^{\mathtt{S},\mathtt{T}}_{\mathtt{X},\mathtt{Z}})$;
  end

□

An advantage of linear recursion over pointwise recursion is that it enables access to the intermediate results of recursion in $\Re^*$ expressions, for example, **first next**$[n]$ $ltc(EDGE)$.

The evaluation of linear recursive operators is quite different than that of pointwise operators, and it can be performed as follows. Suppose that we are given the expression $\Theta(E_1, \ldots, E_n)$ where Θ is an operator defined by $\Theta(X_1, \ldots, X_n) = E(X_1, \ldots, X_n)$. Suppose that the time of evaluation is t. We first evaluate E_1 through E_n at time t to obtain the relations they represent. Let $R_1, \ldots, R_n$ be those relations. We then evaluate the expression $E(R_1, \ldots, R_n)$ at time t (which is the right-hand side of the definition of Θ with each X_i replaced by the corresponding R_i), as if it is just an expression of $\Re^*$. Recursive calls unfold naturally as the evaluation proceeds.

Termination of recursive query evaluation is, in general, not always guaranteed because we may have future recursion in which an operator is defined in terms of its own future results, for example,

$$tu\,(r) = r \vec{\cup} \textbf{ next } tu\,(r).$$

Thus, we stipulate that all definitions be *causal*, that is, future data may be defined only in terms of past data.

In short, causality is a sufficient condition under which each equation defines a linear recursive operator, and hence the termination of recursive query evaluation is guaranteed. Causality of definitions can be checked syntactically.

A terminating algorithm for checking the causality of a recursive equation is given in [19].

Standard fixed-point techniques [25] can be employed to explain the meaning of recursive definitions. For $\Re^*$ to be an algebra in the mathematical sense, it can be shown that the meaning of an expression 'E **where** D **end** ' with respect to a given *db* is also a time-varying relation.

A straightforward extension to $\Re^*$ is to allow mutual recursion. For the evaluation of a set of mutual recursive equations defining pointwise operators, we can borrow the techniques described by Ceri et al [6], Houtsma and Apers [12], and Guessarian [11].

For the evaluation of a set of mutual recursive equations defining non-pointwise operators, we first need to ensure the linearity of equations. The notion of causality can be generalized to a set of equations so that each defined operator can only refer to the past values from the same or *other* defined operators. This notion of causality is analogous to the notions of stratification in deductive databases [6], and linearization in logic programming [10].

For the evaluation of a mixed set of mutually recursive equations defining both pointwise and linear recursive operators, we can use the evaluation method corresponding to each defined operator, and switch from one method to the other as required.

We plan to study the connections between the recursive equations of $\Re^*$ and temporal fixpoints [3]. In a different context, that of temporal logic programming, Baudinet [4] studied the expressiveness of temporal properties defined by program clauses.

5 Temporal Completeness

Temporal completeness is not a well-established notion yet; there are many approaches to temporal algebras, based on a variety of interpretations of time, and it is not clear what set of temporal operators are necessary in a temporal relational algebra. There are still several proposals for temporal completeness. We now discuss temporal completeness of $\Re^*$ with respect to the algebras of Tuzhilin and Clifford [27] and Gabbay and McBrien [9] which are also based on temporal semantics. We show that past linear recursive operators of these algebras can be defined using recursive equations of $\Re^*$. Future linear recursive operators can be defined with some modifications on the underlying model and algebra.

Tuzhilin and Clifford [27] proposed a temporal relational algebra (called **TA**) as a basis for temporal relational completeness. Clifford et al [7] call this notion of temporal completeness *TU-completeness* (temporally ungrouped completeness). **TA**) 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 temporal operators **since** and **until**; their motivation, as well as that of Gabbay and McBrien [9], is based on the fact that the temporal logic of **since** and **until** is fully expressive for a historical data model in the same sense that first-order logic is for a non-temporal data model over the reals or integers. Their temporal relational model, unlike the underlying model of $\Re^*$, is based on discrete and bounded models of time.

The definitions of the past and future linear recursive operators are as follows (in a slightly modified form in λ-notation):

$$L_P(R,S) =$$
$$\lambda t.\begin{cases} \emptyset, & t=0 \\ (R(t-1) \cap L_P(R,S)(t-1)) \cup S(t-1), & t>0 \end{cases}$$

$$L_F(R,S) =$$
$$\lambda t.\begin{cases} \emptyset, & t=n \\ (R(t+1) \cap L_F(R,S)(t+1)) \cup S(t+1), & t<n \end{cases}$$

where R and S are union compatible time-varying relations (they are defined on the same scheme), and n is the *last* moment in time.

In $\Re^*$ we do not have analogues of these two operators, because $\Re^*$ is based on a model of time with an unbounded future. Therefore we consider a model of time with a last moment in time, but we propose to use the recursive equations of $\Re^*$ to define these two linear recursive operators.

We first enrich $\Re^*$ with another operator which is symmetric to **fby**, say **pby** (read as *preceded-by*).

- R **pby** $S = \lambda t.\begin{cases} R(t), & t=n \\ S(t+1), & t<n \end{cases}$

where R and S are union compatible time-varying relations, and n is the last moment in time. The resulting algebra with **pby** is referred to as $\Re^*$ *with bounded time*.

Now the definitions of L_F and L_P are straightforward:

$$L_P(R,S) = \emptyset \textbf{ fby } ((R \vec{\cap}\ L_P(R,S)) \vec{\cup}\ S)$$

$$L_F(R,S) = \emptyset \textbf{ pby } ((R \vec{\cap}\ L_F(R,S)) \vec{\cup}\ S)$$

Therefore we conclude that $\Re^*$ with bounded time is *TU-complete* with respect to **TA** of Tuzhilin and Clifford. In fact, $\Re^*$ with bounded time is more expressive, because, through the use of recursive equations, we can also define other recursive operators that cannot be defined in **TA**.

The starting point of Gabbay and McBrien's algebra [9] is also a temporal logic with **since** and **until**. They refine **TA** by replacing L_P and L_F with two other linear recursive operators, namely, *since-product* ($S_\times$) and *until-product* ($U_\times$). The recursive definitions of these two operators in $\Re^*$ are as follows:

$$S_\times(R,S) = S_\times(R,S,\mathcal{E})$$
$$S_\times(R,S,E) = \emptyset \textbf{ fby } ((E \vec{\times}\ S) \vec{\cup}\ S_\times(R,S,(R \vec{\cap}\ E)))$$

and

$$U_\times(R,S) = U_\times(R,S,\mathcal{E})$$
$$U_\times(R,S,E) = \emptyset \textbf{ pby } ((E \vec{\times}\ S) \vec{\cup} U_\times(R,S,(R \vec{\cap}\ E)))$$

where $\mathcal{E}$ is the universal set, and R and S need not be union compatible.

Again, it follows that $\Re^*$ with bounded time is *TU-complete* with respect to **TA** of Gabbay and McBrien.

Gabbay and McBrien [9] embed their algebra into an extended relational algebra with arithmetic capabilities in select and project operations, and also show that the embedding provides a convenient way to translate queries in a temporal extension of SQL into those in standard SQL.

6 Concluding Remarks

We have introduced a temporal algebra $\Re^*$ which is capable of expressing recursive queries through definitional equations. The difference between our approach to temporal algebras and many others reported in the literature is that $\Re^*$ is a "temporal algebra" in the mathematical sense, and that it is based on temporal semantics. However, in its current form $\Re^*$ can only support *valid-time*, not *transaction-time* [13]. Transaction-time is orthogonal to valid-time, which records times of updates (e.g., when facts became known to the temporal database).

We believe that $\Re^*$ is a suitable target language for optimizing queries in a (recursive) temporal extension of SQL or QUEL. Temporal query languages include TSQL-like query language of Sarda [22], and TQUEL of Snodgrass [23]. Query optimization strategies can be directly based on the formal properties of $\Re^*$ [19], and storage structures for time-varying relations.

Acknowledgements

This research has been supported in part by an Australian Research Council (ARC) grant and a Macquarie University Research Grant (MURG). Thanks are due to Lee Flax and Rajiv Bagai for many useful discussions.

References

[1] R. Ahad and B. Yao. RQL: A recursive query language. *IEEE Transactions on Knowledge and Data Engineering*, 5(3):451–461, June 1993.

[2] R. Bagai and M.A. Orgun. A temporal paraconsistent relational algebra for incomplete and inconsistent information. In *Proceedings of the 33rd ACM Southeast Conference*, pages 240–248, Clemson University, South Carolina, USA, 1995. ACM Press.

[3] H. Barringer. The use of temporal logic in the compositional specification of concurrent systems. In A. Galton, editor, *Temporal Logics and Their Applications*, pages 53–90. Academic Press, 1987.

[4] Marianne Baudinet. On the expressiveness of temporal logic programming. *Information and Computation*, 117:157–180, 1995.

[5] C. Beeri and T. Milo. On the power of algebras with recursion. In P. Buneman and S. Jajoida, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 377–386. ACM Press, 1993.

[6] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, Berlin Heidelberg, 1990.

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

[8] E. F. Codd. A relational model of data for large shared data banks. *Communications of the Association for Computing Machinery*, 13(6):377–387, 1970.

[9] D. Gabbay and P. McBrien. Temporal logic & historical databases. In *Proceedings of the 17th Very Large Data Bases Conference*, pages 423–430, Barcelona, Spain, September 1991. Morgan Kauffman, Los Altos, Calif.

[10] I. Guessarian and J.-E. Pin. Linearizing some recursive logic programs. *IEEE Transactions on Knowledge and Data Engineering*, 7(1):137–149, February 1995.

[11] Irène Guessarian. A note on fixpoint techniques in data base recursive logic programs. *RAIRO Theoretical Informatics and Applications*, 22:49–56, 1988.

[12] Maurice A. W. Houtsma and Peter M. G. Apers. Algebraic optimization of recursive queries. *Data & Knowledge Engineering*, 7:299–325, 1992.

[13] C. S. Jensen et al. A consensus glossary of temporal database concepts. *SIGMOD RECORD*, 23(1):52–64, March 1994.

[14] Patrick P. Kalua and Edward L. Robertson. Benchmark queries for temporal databases. Technical Report TR379, Computer Science Department, Indiana University, Bloomington, Indiana 47405, USA, March 1993.

[15] Kemal Köymen and Qujun Cai. SQL*: A recursive SQL. *Information Systems*, 18(2):121–128, 1993.

[16] P. Ladkin. *The Logic of Time Representation.* PhD thesis, University of California, Berkeley, California, USA, 1987.

[17] L. Edwin McKenzie Jr. and R. Snodgrass. Evaluation of relational algebras incorporating the time dimension in databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[18] M. A. Orgun. Incorporating an implicit time dimension into the relational model and algebra. Department of Computing, Macquarie University, Sydney, NSW 2109, Australia, 1994.

[19] M. A. Orgun. A temporal algebra with recursive equations. Technical Report 95-05, Department of Computing, Macquarie University, Sydney, NSW 2109, Australia, 1995.

[20] 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*, pages 301–316, Brisbane, Queensland, Australia, February 1–2 1993. World Scientific, Singapore.

[21] G. Özsoyoğlu and R. T. Snodgrass. Temporal and real-time databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513–532, August 1995.

[22] N. L. Sarda. Algebra and query language for a historical data model. *The Computer Journal*, 33(1):11–18, 1990.

[23] R. Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, 1987.

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

[25] J. E. Stoy. *Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[26] A. U. Tansel et al., editors. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings Publishing Company, Redwood City, CA, 1993.

[27] 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*, pages 13–23, Brisbane, Australia, August 13–16 1990. Morgan Kaufmann Publishers Inc., Los Altos, Calif.

[28] G. Wiederhold, S. Jajoida, and W. Litwin. Dealing with granularity of time in temporal databases. In R. Andersen, J. A. Bubenko, and A . Sølvberg, editors, *Advanced Information Systems Enginering: Proceedings of the Third International Conference CAiSE'91*, pages 124–140, Trondheim, Norway, May 13–15 1991. Springer-Verlag.