ON PERIODICITY IN TEMPORAL DATABASES

Alex Tuzhilin

James Clifford

Stern #IS-95-6

ON PERIODICITY IN TEMPORAL DATABASES

Alex Tuzhilin

James Clifford

Forthcoming in Information Systems, Courant Institute of Mathematical Sciences, Department of Computer Science, New York University

Working Paper Series
STERN IS-95-6

On Periodicity in Temporal Databases*

Alexander Tuzhilin and James Clifford

Information Systems Department
Stern School of Business
New York University

Abstract

The issue of periodicity is generally understood to be a desirable property of temporal data that should be supported by temporal database models and their query languages. Nevertheless, there has so far not been any systematic examination of how to incorporate this concept into a temporal DBMS. In this paper we describe two concepts of periodicity, which we call *strong periodicity* and *near periodicity*, and discuss how they capture formally two of the intuitive meanings of this term. We formally compare the expressive power of these two concepts, relate them to existing temporal query languages, and show how they can be incorporated into temporal relational database query languages, such as the proposed temporal extension to SQL, in a clean and straightforward manner.

1 Introduction

Periodicity is a feature of temporal phenomena that has received relatively little attention in the literature of temporal databases. The work that has been done related to periodicity, which we survey in Section 1.1, has been either mostly of a theoretical nature or is related to the notion of a *calendar* [26, 23, 6]. However, there has been no comprehensive treatment of the subject of periodicity in the temporal database literature that thoroughly addresses both the meaning of periodicity in the database context as well as how this aspect of time can be supported in commercial temporal database systems. As a result of the limited attention to this concept, periodicity is not supported in the TSQL2 standard [25] and no periodic queries were included in the test suite of temporal queries [15].

*The work of the authors was supported in part by the NSF under Grant IRI-93-18773.

This situation is unfortunate since many of the existing temporal query language proposals cannot express periodic queries, as we shall discuss in Section 3. For example, they cannot express even such a simple periodic query as "Find the names and attendance times of people who attended any of the Tuesday meetings."

The goal of this paper is twofold: we explore the meaning of the concept of periodicity in the context of temporal databases, *and* we show how it can be supported in commercial temporal query languages and in the TSQL2 standard in particular. Our objective is mainly *pragmatic*: we want to add periodicity in a *practical* way to temporal query languages, including TSQL2. However, it is impossible to do this in a comprehensive and theoretically sound way without a proper understanding on the conceptual level of what periodicity means and how it can be formally represented. Therefore, we study periodicity *both* on the theoretical and the pragmatic levels.

To understand what periodicity means intuitively, a good place to start is in the dictionary. The entry in the American Heritage Dictionary [1] gives the following three meanings for the term *periodic*:

1. having periods or repeated cycles
2. happening or appearing at regular intervals
3. taking place now and then; intermittent.

The first of these meanings for the term *periodic* says that the events occur at such moments of time that the distance between these moments is *the same*. For example, a class might be scheduled to meet periodically, say, once a week on Wednesdays at 11 am. Note that the time between two successive occurrences in this case is always exactly seven days.

The second meaning says that the events occur at *regular* intervals, meaning that the events do not necessarily occur at equally distant moments of time. However, there is a certain degree of "regularity" in their occurrence: they are "regularly" distanced and thus occur within a certain limit from each other. For example, assume that the head of a state of some country has to travel *once every week* on some business but at the same time is concerned about security because of the recent increase in the terrorist activities in that country. To make the trips safer, the schedule of trips is kept secret and is organized so that the day of the week when the politician travels appears to be totally random. In this example, the set of trips is "regular" in the sense that they occur once a week. On the other hand, they are not "regular" in the first sense of the word: they are not equally distanced from each other, *nor* can this set of trips be obtained as a union of equal-distance sequences (as will be shown later in the paper). This concept of periodicity is clearly a weaker concept than the first one.

The third meaning for the term *periodic* says that the events occur "intermittently," meaning that if one event occurred then the next will follow some time in the future, but it is not clear when. For example, a person may visit a pub "periodically," meaning that the visits can be quite irregular but the person will keep going to the pub. Clearly, this is the weakest concept of periodicity among the three.

In this paper, we provide formalizations for the first two concepts of periodicity in the context of querying a database along the valid time or transaction time dimensions of its data. We do not study the third notion of periodicity because we feel that it is "too weak" to be formally supported by temporal query languages: technically, all that it says is that periodic sets are unbounded in the future.

After we formalize the two notions of periodicity described above and study their properties and the relationship between them, we address the pragmatic question of how to add periodicity in a practical way to commercially oriented temporal query languages. In this paper, we concentrate on the *ungrouped* temporal data models [8] and, therefore, consider ungrouped temporal query languages, and in particular the TSQL2 standard. However, we believe that the ideas presented in the paper can be extended to the *grouped* temporal data models [8] as well. In the process of studying of how periodicity can be added to commercially oriented temporal query languages, we rely on the theoretical results about periodicity described earlier in the paper.

When studying periodicity, one has to address two questions. The first question is how to define *periodic sets of times*. For example, one might want to define the set consisting of "every Wednesday," or "the first day of each month," or a "set of visits of the head of a state." The second question is how to define *periodic queries*, such as a query to find people who attended every meeting held on Wednesdays, or people that came to the meetings once a week.

This distinction between periodic sets of time and periodic queries corresponds to the distinction between data expressiveness and query expressiveness discussed in [5]. The *data expressiveness* of a formalism specifies what types of infinite temporal databases can be defined with the formalism. It arises only in the context of infinite temporal databases because in the finite case all the finite databases can be represented explicitly. The *query expressiveness* of a formalism specifies what types of queries can be expressed in the formalism and is more akin to the standard notion of expressiveness of query languages. Therefore, in order to understand how periodicity can be added to TSQL2 in the best possible way, we will study the data and query expressiveness of different periodic extensions studied in this paper.

1.1 Related Work

Perhaps the first approach to analyzing the notion of "time" and its relationship to conceptual modeling of data was proposed in [4]. Among the many aspects of time that are discussed here is the notion of periodicity. A periodic set is defined here in a set-theoretic framework in terms of its "period, duration, and cover interval." Periodic sets, along with all of the time aspects discussed in the paper, are all interrelated via a concept network similar to a semantic net. While this work is of pioneering importance in the temporal database literature by having explored so many aspects of the temporal dimension of data, it did not study the properties of periodic sets of times, nor did it address the issue of a language for expressing queries about the periodic nature of data.

The model presented in [19] mentions a technique for "the recording of periodic events." It does so by allowing the user to associate multiple valid-time time stamps with a given fact. For example, a tuple such as $< john, d5, d9, h8, h12 >$ on relation scheme GUARDS(NAME, Fdate, TDate, Fhour, Thour) is said to represent that "$John$ works as a guard during the hours $h8-h12$ ($h12$ excluded), for *every* day in the interval $[d5, d9)$." However, the semantics of such relations with multiple valid-time timestamps is not fully explored. Moreover, the query language does not support queries about periodic events.

In [13], a classification of relations based on the types of relationships allowed between the transaction-time and valid-time timestamps of "items" in the database is presented. Among the varieties of relations types discussed are several that rely on a notion which is called "regularity." For example, a temporal relation is said to be "*transaction time event regular* with time unit $\Delta t \geq 0$" if the transaction times of any two items stored in the database is always an integral multiple of some specified duration Δt. Variations of this type of consideration lead to some additional characterizations of relations related to temporal regularity. As the authors point out, regularity is related to, but is different from, periodicity. In fact, as we shall see, regularity can be defined in terms of our notion of periodicity by starting with a periodic set of times and subtracting any arbitrary, but recursively defined, set of times from it. Finally, the issues of a language for querying such databases with properties of regularity, or for expressing these regularity constraints on the temporal dimensions, are not considered.

Classical work in temporal logic [18, 20] is concerned with truth and validity of temporal logic formulae, which are defined in terms of *temporal structures* [18] specifying how propositions or predicates change over time. As was observed by [34], classical temporal logic does not address the issue of periodicity. In particular, [34] showed that the temporal logic with *next* and *until* operators cannot define the set of even numbers, which forms one of the basic periodic sets. As a remedy, Wolper proposed the language ETL that extended the standard temporal logic with additional temporal connectives [34]. The extended language could express periodic events. However, to get the full expressive power of ETL we would need infinitely many temporal connectives. To remedy this problem Vardi proposed the temporal fixpoint calculus μTL [33].

The language μTL, as defined in [33], is a propositional temporal calculus, although Vardi points out that it can be easily extended to the predicate case [33]. Calculus μTL is obtained from the propositional temporal logic with operators **next**, and two types of "previous" operators, $\mathbf{previous}_1$ and $\mathbf{previous}_2$[1] by the addition of the operators *least* and *greatest fixpoint*. The formula $\mu X.\phi(X)$ (respectively $\nu X.\phi(X)$) denotes the least (respectively greatest) fixpoint solution to the equation $X \equiv \phi(X)$, where X is a temporal proposition.

Calculus μTL can express temporal operators **until** and **since** and can also define periodic sets. For example, it can define the set of even numbers as

[1]The difference between them is that $\mathbf{previous}_1$ is always true and $\mathbf{previous}_2$ is always false when applied to an arbitrary formula at time $t = 0$.

$$\mu X.(init \vee \textbf{previous}_2\, \textbf{previous}_2 X)$$

where *init* is defined as $\textbf{previous}_1$ **false** and is equal to true at time $t = 0$ and false elsewhere. Vardi [33] studies the expressive power of μTL and shows that it a quite expressive language: it has the expressive power of ω-regular languages [29]. This means that μTL is "too powerful" to express periodicity *alone*: intuitively, μTL does not capture the "essence" of periodicity, but supports other concepts *in addition* to periodicity. Since in this paper we are interested in the question of what mechanism is needed to add periodicity, and only periodicity, to existing temporal query languages and calculi, we want to study languages that are less powerful than μTL.

Periodicity has also been studied in the context of infinite temporal databases in [16, 7] and surveyed in [5]. In [16, 5] a *generalized relation* is defined as a set of generalized tuples of the form

$$\{t_1, \ldots, t_m, d_1, \ldots, d_l \mid t_1 \in (a_1 n_1 + b_1), \ldots, t_m \in (a_m n_m + b_m) \text{ and constraints on } (t_1, \ldots, t_m) \text{ are satisfied}\}$$

where $t_1, \ldots, t_m$ are temporal variables, $d_1, \ldots, d_l$ are non-temporal constants, and the constraints imposed on $t_1, \ldots, t_m$ are linear. The expressions $a_i n_i + b_i$ are called *linear repeating points (LRPs)*, and they form periodic sets. If we restrict the notion of a generalized tuple to only one attribute and make it temporal, then a generalized relation would consist of a set of LRP's with some linear constraints imposed on them. This representation would correspond to the first notion of periodicity above, i.e., "having periods or repeated cycles." The union of LRP's would also capture the second notion of periodicity, i.e., "happening or appearing at regular intervals," but only to *some* extent. For example, as will be shown later in the paper, a finite union of LRP's cannot represent the set of "visits of the head of a state" described in the introduction. We will call the sets defined by unions of LRP's *strongly periodic* sets to distinguish them from the more general notion of *nearly periodic.* We will show in the paper that the concept of near periodicity subsumes strong periodicity and captures the second meaning of periodicity, i.e. "happening or appearing at regular intervals."

Also, certain aspects of periodicity are studied in [7] in the context of Datalog_{1s} (and therefore Templog [5]) programs[2]. For example, the set of even points can be defined in Datalog$_{1s}$ (and hence in Templog) as $q(0) \leftarrow$, $q(T+2) \leftarrow q(T)$. This program says that q is true at time $t = 0$, and that if q is true at some moment of time T, then it is also true at time $T+2$. It is shown in [5] that the (infinite) temporal database defined by a Templog (or a Datalog$_{1s}$) program is ultimately periodic (i.e. becomes periodic starting from some time). Although [7] deals with periodicity issues within the context of Datalog$_{1s}$, this work does not study periodicity per se: it does not directly address the questions presented in the introduction: i.e., what periodicity *is*, and how it can be added in a *practical* way to the commercial temporal query languages, including the TSQL2 proposal.

In summary, the work of [16, 7, 5] studies the theory of temporal deductive databases and in the process examines certain theoretical aspects of periodicity. However, it considers only what in this paper we call *strong* periodicity, and does not consider a more general concept of *near* periodicity that will be introduced below. Furthermore, it does not address the pragmatic issues of how periodicity can be added to existing temporal SQL proposals and to the TSQL2 standard in particular.

Still another work related to periodicity is the work on *calendars*, as presented in [26, 23, 6]. The objective of this work is to define not only the standard Gregorian calendar but other calendars as well, such as Lunar, Meso-American, Russian and ultimately an arbitrary user-defined calendar. However, to achieve this goal, the researchers studying calendars take different approaches.

In [6], calendars are defined with a calendar expression language. To define a particular user-defined calendar, a script is written in this language. For example, to define the calendar "the last day of every month; but if this day is a holiday, then the preceding business day," [6] proceeds as follows. First, it specifies calendars "last day of every month" and "American holidays," and then defines the calendar "business days" by subtracting "American holidays" from week-days. Then it adjusts "last day of every month" based on whether the last day of a month belongs to "business days" or not.

In [23], calendars are defined as periodic infinite sets of consecutive intervals which partition time, such as `years`, `weeks`, `months`, etc. Also, [23] imposes the partial order *subcalendar* on calendars. For example, `days` is a subcalendar of `weeks` and of `months`. In

[2]It is shown in [5] that the two languages are equivalent.

addition, [23] defines *slices* of calendars that are non-consecutive repeated intervals of time. Examples of slices are "Sundays," "the 4th hour of the 3rd day of each month," and "from 3 am to 6 pm each 3rd day of each month."

Soo and Snodgrass [26] were among the first to study calendars, and their work, as described in [26], was subsequently incorporated into the TSQL2 standard [25]. In [26], Soo and Snodgrass define a calendric system with a set of properties that are contained in a property table which is activated with the `set properties` command. The system presented in [26] supports the temporal data types *events*, *intervals*, and *spans* that correspond to moments of time, periods of time, and durations of time. Unlike [23] and [6], where it is possible to select "slices" of a calendar (non-consecutive repeated intervals of time), [26] considers only whole calendars and does not provide operators that "slice" and "dice" them into pieces. Therefore, [26] apparently does not support such periodic sets of time as "the first Monday of each month," or "the last non-holiday day of every month."

Note that the calendars, as defined in [23] and [6], support such periodic events as "every Sunday" and "every first Monday of each month." However, as we show in the paper, periodicity is not limited to calendars. It is based on a more general concept of "regularity" and encompasses arbitrary recursive sets of time, elements of which exhibit certain "regularity" that we define later in the paper.

As explained earlier, we follow the methodology used in [5], when studying periodicity, and consider both *data* and *query* expressiveness. In the next section, we study the data expressiveness issues of periodicity by considering formalisms for defining periodic sets of time. Then in Section 3 we address the issue of query expressiveness by considering formalisms for defining periodic queries.

2 Periodic Sets of Time

In this section, we consider just the temporal domain, in isolation from the value domain, and will describe the notion of periodic sets of times for that domain. In particular, we are concerned with the data expressiveness of various formalisms that define periodic sets to see how well they capture the intuitive meanings of periodicity described in Section 1.

Note that all the dictionary meanings of the word periodicity discussed in Section 1, including the third one, assume that each periodic event has a successor. This means, among other things, that sequences of periodic events are infinite. In practice, however, databases are finite, and therefore infinite periodic sequences are only abstractions.

To begin with, we have to specify the model of time. The most general model represents time as an arbitrary set with a partial order imposed on it. With additional axioms, we can introduce other models of time, e.g., time can be treated as discrete or dense, bounded or unbounded, linear or branching [32]. In this paper, we assume that time is discrete, linear, unbounded in the future and bounded in the past, because this is the model of time generally considered by historical and temporal data models ([21, 27]). This model of time is isomorphic to the set of natural numbers [32]. For this reason, we assume throughout most of the paper that time is represented with natural numbers.

In the next section, we consider one type of periodicity that captures the first meaning of this term described in the introduction. Then in Section 2.2, we consider a more general meaning of this term.

2.1 Strongly Periodic Sets of Time

Periodic sets of numbers have been studied before and are relatively well understood objects. In particular, Enderton describes these sets in [10] within the context of mathematical logic and [16, 5] in the context of infinite temporal databases. To make the paper self-contained, we review in this section the standard concepts of periodicity, as presented in these references and elsewhere.

A *linear repeating point (LRP)* [16, 5] is a set

$$\{x \in N \mid (\exists k)(k \in N \wedge x = ak + b)\}$$

where $a, b \in N$[3]. LRPs are usually denoted as $an + b$. Also, an *eventual linear repeating point* is defined as

$$M \cup \{x \mid (\exists k)(x > c \Rightarrow x = ak + b)\}$$

where M is a finite set of natural numbers and $a, b, c \in N$.

[3]Note that [16, 5] assume that a, b x and k range over integers. However, we follow Enderton [10] in this paper who considers only the set of natural numbers.

Note that the definition of a linear repeating point corresponds to the first intuitive meaning of periodicity (having periods or repeated cycles) as presented in [1] and discussed in the introduction. Furthermore, if we combine various LRPs, we will get the following concept of strong periodicity.

Definition 1 *A set of natural numbers is* strongly periodic *if it can be defined as a finite union of eventual linear repeating points.*

As we said already, strong periodicity was studied before under different names in [10] and [16] (we use the term "strong periodicity" to distinguish it from "near periodicity" to be defined below). In the rest of this section, we will refer to strongly periodic sets simply as "periodic sets" if no confusion arises (however, we will call them "strongly periodic" again in the rest of the paper).

Example 1 Consider the set of times that define first dates of each month over the years. This set is strongly periodic. To see this, assume that there are no leap years[4] and that the beginning of time starts with 1 (so that January 1 of the first year corresponds to 1). Then this set can be defined as

$$365n + 1 \cup 365n + 32 \cup 365n + 60 \cup 365n + 91 \cup \ldots \cup 365n + 305 \cup 365n + 335$$

where the union has 12 terms, and each term corresponds to the first day of the corresponding month.

□

It is easy to see that strongly periodic sets are closed under union, intersection, and complementation. Also strong periodicity corresponds to the second intuitive meaning of periodicity (happening or appearing at regular intervals) as presented in [1] and discussed in the introduction. For example, consider the set consisting of the union of even numbers and multiples of 3. Note that the elements of this set do not occur in a cycle, as is required in the first intuitive meaning of periodicity, but they occur "regularly enough" so that they can be represented as a union of LRPs.

We next consider different first-order logics(s) that can define strongly periodic sets of numbers. As a starting point, we consider the first-order language $\mathcal{R}_L = (N, 0, S, <)$ with

[4]It is easy to see that this set is still strongly periodic if this restrictive assumption is removed. However, the removal of this assumption will require more complex expressions to define this periodic set of times.

equality, where N is the set of natural numbers, 0 is a constant symbol (intended to denote the number 0), S is the successor function $S : N \to N$ (i.e. $S(n) = n + 1$), and $<$ is a linear ordering relation on N. For instance, if we use $t+1$ as a shorthand for $S(t)$, then the formula $(\exists t')(t + 1 < t' \wedge t' > 100)$ is an example of a well-formed formula in $\mathcal{R}_L$.

Enderton shows [10][Corollary 32C] that a set of numbers is definable in $\mathcal{R}_L$ if and only if it is either finite or has finite complement. It follows immediately from this that strongly periodic sets of time cannot be expressed in $\mathcal{R}_L$ except for some degenerate cases. For example, the set of even (or odd) numbers cannot be expressed in $\mathcal{R}_L$. Therefore, we have to extend $\mathcal{R}_L$ to be able to define periodic sets.

A natural way of extending $\mathcal{R}_L$ to support periodic sets would be to add the one-place predicate *periodic* to $\mathcal{R}_L$ (actually, we define a class of such predicates, one for each pair of p and s):

$$periodic_{p,s}(t) \overset{\triangle}{=} t \equiv s \bmod p \text{ or as } (\exists k)(t = k * p + s) \tag{1}$$

We denote the resulting language as $\mathcal{R}_{Lp}$. Clearly, we can define some periodic sets in $\mathcal{R}_{Lp}$. For example, the set of even and odd numbers can be defined using $\mathcal{R}_{Lp}$ as $\{n \mid periodic_{2,0}(n)\}$ and $\{n \mid periodic_{2,1}(n)\}$ respectively.

Since predicate *periodic* is defined above in terms of congruence ($\equiv$) or addition ($+$) operators, we can also define two languages $\mathcal{R}_{L\equiv}$ and $\mathcal{R}_{L+}$ [10] by adding, respectively, *congruence* and *addition* operators to $\mathcal{R}_L$. For example, the set of odd numbers can be defined using $\mathcal{R}_{L\equiv}$ as $\{n \mid n \equiv 1 \ mod \ 2\}$, and as $\{n \mid (\exists m)(n = m + m + 1)\}$ using $\mathcal{R}_{L+}$. Clearly, the languages $\mathcal{R}_{L\equiv}$ and $\mathcal{R}_{L+}$ are at least as expressive as $\mathcal{R}_{Lp}$.

Furthermore, Enderton shows in [10][Theorem 32F] that

Theorem 2 *A set of natural numbers is definable in $\mathcal{R}_{L+}$ if and only if it is strongly periodic.*

Intuitively, this theorem holds because any $\mathcal{R}_{L+}$ formula with quantifiers can be converted to an equivalent $\mathcal{R}_{L\equiv}$ formula without quantifiers. Furthermore, the following corollary immediately follows from the proof of this theorem.

Corollary 3 *Languages $\mathcal{R}_{Lp}$, $\mathcal{R}_{L\equiv}$, and $\mathcal{R}_{L+}$ have the same expressive power and define the class of strongly periodic sets.*

In summary, the three languages $\mathcal{R}_{Lp}$, $\mathcal{R}_{L\equiv}$, and $\mathcal{R}_{L+}$ have the same expressive power, and define *exactly* the class of strongly periodic sets.

As we mentioned in Section 1.1, generalized relations, as defined in [16, 5], are related to the concept of strong periodicity: if we restrict generalized tuples to only one attribute and make it temporal, then a generalized relation would be strongly periodic.

In [23], Niezette and Stevenne took the concept of linear repeating points and applied it to the definition of calendars. The resulting language can support not only the Gregorian calendar, but also other types of calendars, such as Lunar, Meso-American, and various user-defined calendars. In [23], they also show that their concept of calendars can be reduced to linear repeating intervals, which is an extension of linear repeating points to the case of intervals. Thus they view calendars as a user-friendly representation of linear repeating intervals. Therefore, the concept of a calendar, as defined in [23], does not take us beyond strongly periodic sets, and thus such sets of times as "the first day of each month," or "American holidays" can be defined in strongly periodic terms.

It follows from the discussion of strong periodicity, that strong periodicity can capture a rich class of the sets which are intuitively "periodic." In particular, it captures the first intuitive meaning of periodicity, as described in the introduction. Furthermore, it captures, to some extent, the second intuitive meaning of periodicity since such non-equal-distance but "regular" sets as "the first date of each month" can be expressed in strongly periodic terms (Example 1). However, it is not clear if strong periodicity is a sufficiently powerful concept to capture *all* the aspects of the second meaning of "periodicity," or if we need a more powerful concept for that purpose. We address this question in the next section.

2.2 Nearly Periodic Sets of Time

A strongly periodic set of times defines a union of eventually periodic points, i.e. of points that are eventually equally distanced from each other. However, the second concept of periodicity, as described in the introduction, is intuitively broader than that. All that this concept says is that "periodic" points have to be "regularly" distanced from each other, i.e., within a certain limit from each other; but they *don't* have to be *equally* distanced though. For example, consider the set of once-a-week trips that the head of a state makes, as described in Section 1. Intuitively, this is a "periodic" set because the trips are *regularly* made (once a week). However, as we will formally show below, this set cannot be represented as a finite union of eventually periodic sets. Intuitively, this is the case because the times of the week when the person travels are very unpredictable (randomized).

To capture the second type of periodicity, we start with an arbitrary recursive set of times S. Then we impose a restriction on the set S that there is a strongly periodic set of points S' and a bijective mapping between points in S and S' such that each point in S must be "close" to the corresponding point in S'. In other words, S and S' are not "too much away" from each other.

Formally, we proceed as follows.

Definition 4 *A recursive set of natural numbers* $\{q_1, q_2, \ldots, q_n, \ldots\}$ *is* essentially nearly periodic *with period* p *if there exists an eventual LRP defining a periodic sequence* $p_1, p_2, \ldots, p_n, \ldots$ ($p_k = k * p + s$ *for some* p, s, *and* M, *and for* $k > M$) *and a number* n, $n < p/2$, *such that* $|q_k - p_k| < n$ *for all* $k > M$.

Note that it immediately follows from this definition that the set of numbers $\{q_1, q_2, \ldots, q_n, \ldots\}$ forms a monotonic sequence, i.e. $q_k < q_{k+1}$ for $k > M$. The following proposition immediately follows from this definition and the definition of a recursive set and provides an alternative characterization of nearly periodic sets.

Proposition 5 *A set of natural numbers* $\{q_1, q_2, \ldots, q_n, \ldots\}$ *is essentially nearly periodic with period* p *if and only if there exists an eventual LRP defining a periodic sequence* $p_1, p_2, \ldots, p_n, \ldots$ ($p_k = k * p + s$ *for some* p, s, *and for* $k > M$ *for some* M), *a recursive function* f, *and a number* n, $n < p/2$, *such that*

- $q_{k+1} = f(q_k)$, *for* $k = 1, 2, 3, \ldots$
- $q_k < q_{k+1}$, *for* $k > M$
- $|q_k - p_k| < n$, *for* $k > M$

Definition 6 *A set* D *of natural numbers is* nearly periodic *if it can be defined as a finite union of essentially nearly periodic sets.*

We will sometimes use a slightly different, though equivalent, characterization of nearly periodic sets based on the following concept of nearly periodic predicates.

Definition 7 *A monadic predicate Q is* nearly periodic *with period p if it can be defined by some essentially nearly periodic sequence of numbers with period p, i.e., there exists an essentially nearly periodic sequence $q_1, q_2, \ldots, q_n, \ldots$ such that $Q(t)$ is true if and only if there exists k such that $Q(t) = q_k$.*

Then a set of natural numbers D is nearly periodic if and only if it can be expressed as

$$D = \{t \mid \bigvee_{i=1}^{k} Q_i(t)\} \tag{2}$$

where $Q_1, \ldots, Q_k$ are nearly periodic predicates.

Example 2 Consider a disk backup policy under which computer disks are backed up once every week but highly irregularly. In particular, the set of backup days is defined with the predicate *backup* as:

$$\{7 * n + irregular(n) \mid n \in N\} \tag{3}$$

where the function *irregular* is a recursive function that (i) maps each natural number to a natural number between 0 and 6, and (ii) has the property that the sequence of numbers $\{x_k\}_{k \in N}$, such that $x_k = irregular(k)$, does not have a periodic subsequence of the same number[5]. In other words, $\{x_k\}_{k \in N}$ does not have a subsequence $\{x_{k_i}\}_{i \in N}$ such that, for all i, $k_{i+1} - k_i = k_i - k_{i-1}$ and $x_{k_i} = x_{k_{i+1}}$. For example, such function can be defined with a pseudo-random recursive function simulating the uniform distribution on the interval [0, 6]. It follows from the definitions that this set of backup dates is nearly periodic.

Similarly, we can define the set of once-a-week visits of the head of a state described in the introduction and at the beginning of this section with the expression (3). Again, this set is nearly periodic.

□

[5]Since expression (3) consists of the "periodic" component $7*n$ and the "random" component *irregular(n)*, it may appear that predicate *backup* can be defined in terms of strong periodicity and *temporal indeterminacy* [9]. However, this is not the case because the function *irregular()* is *recursive* and not a random function. Therefore, there is no indeterminacy involved in defining predicate *backup*.

It follows from the definitions of strongly and nearly periodic sets that a strongly periodic set is also nearly periodic. It is also important to know if the inverse is true, and the next proposition provides the answer to this question.

Proposition 8 Backup *and* head_of_state_visits *do not define strongly periodic sets.*

Proof: The proof follows from the definition of strong periodicity and from the fact that sets *backup* and *head_of_state_visits* do not have any periodic subsets. □

The next corollary follows from this proposition and from the fact that a strongly periodic set is also nearly periodic.

Corollary 9 *The class of strongly periodic sets of time is properly contained in the class of nearly periodic sets.*

We next consider the restrictions that have to be imposed on the nearly periodic predicates in order to make them strongly periodic. Note that nearly periodic predicates are defined with arbitrary recursive functions as long as they satisfy the restrictions specified in Definition 4 (or equivalently Proposition 5). In contrast to this, it follows from Theorem 2 that the set of natural numbers is strongly periodic if and only if it is expressible in $\mathcal{R}_{L+}$. Therefore, a necessary and sufficient condition for a nearly periodic set to be strongly periodic is that $\bigvee_{i=1}^{k} Q_i(t)$ in (2) be expressible in $\mathcal{R}_{L+}$; this condition delineates the boundary between nearly and strongly periodic sets. For instance, if the recursive predicate *irregular* from Example 2 is not expressible in $\mathcal{R}_{L+}$, then the resulting sets *backup* and *head_of_state_visits* are nearly but *not* strongly periodic.

3 Formalization of Periodic Queries

In Section 2, we considered periodic *sets of time* and the data expressiveness of periodic extensions of $\mathcal{R}_L$. In this section, we study periodic *queries* and their *query expressiveness* [5]. This means that we will consider arbitrary temporal relations [14] in addition to periodic sets of time and study formalisms that capture the two intuitive meanings of periodicity, described in the introduction, with respect to queries on a database.

Our starting point is the standard ungrouped temporal calculus **TC** [31, 8] that serves as a formal basis for the ungrouped temporal query languages. The calculus **TC** is based on two-sorted first-order logic where one of the sorts is linearly ordered (by a relation denoted $<$), and is interpreted as the set of times. Furthermore, all the predicates can have at most one temporal argument. Calculus **TC** can be thought of as the language $\mathcal{R}_L$ extended with temporal relations. Several temporal relational data models, such as TQuel [24], support two times per relation, typically interpreted as "start time" when the tuple was added to the relation and "end time" when it was removed. This temporal data model can be represented with the calculus $\mathbf{TC}_2$ [16] that is very similar to **TC** except that each predicate has two times instead of one. As an example of language **TC**, the query finding names of all the employees that received a cut in their salaries can be expressed in **TC** as

$$\{name \mid (\exists t)(\exists t')(EMPL(name, sal, t) \wedge EMPL(name, sal', t') \wedge t < t' \wedge sal > sal')\}$$

Since **TC** is based on $\mathcal{R}_L$, it cannot express the query **even** that returns the set of even time points, and thus, like $\mathcal{R}_L$, it is not powerful enough to support periodicity. In [8] we considered some temporal query languages that have the expressive power equal to, or less than, **TC**. Therefore, these languages also cannot express periodic queries. Furthermore, as will be explained in Section 5, periodic queries are not supported in TSQL2 either. Therefore, we have to extend **TC** (and these query languages) to support periodicity.

The language **TC** can be extended to support periodicity by adding strongly or nearly periodic predicates to it. In particular, the *strongly periodic* language $\mathbf{TC}_p$ is obtained from the language **TC** by adding the class of monadic predicates $periodic_{ps}$ defined by (1) to the temporal sort of **TC**.

Clearly, the language $\mathbf{TC}_p$ can express periodic queries of the first type defined in the introduction, i.e. queries dealing with "periods or repeated cycles." The following examples illustrate this point.

Example 3 The query "Find the names and attendance times of people who attended any of the Tuesday meetings of the Computer Resources Committee (CRC)" can be expressed in $\mathbf{TC}_p$ as

$$\{< empl, t > \mid ATTEND(empl, meeting, t) \;\wedge\; meeting = \text{"CRC"} \;\wedge\; periodic_{7,2}(t)\}$$

where `ATTEND` is a relationship specifying the times when employees attended meetings, and

7 corresponds to seven days of a week, and 2 to Tuesday.

□

Example 4 The query "Find the people who attended all the Tuesday meetings of the CRC committee in the past two months" can be expressed in $\mathbf{TC}_p$ as

$$\{empl \mid (\forall t)((periodic_{7,2}(t) \ \wedge \ now \leq t+60 \ \wedge \ t < now) \Rightarrow ATTEND(empl, \text{"CRC"}, t))\}$$

assuming that $t+60$ is a shorthand for the successor function S applied 60 times to t.

□

Moreover, strongly periodic queries can express some of the periodic queries of the second type that deal with the sets of events that are not strongly periodic. For example, the query "Find the people who attended the CRC meetings *once a week*" can be expressed as

$$\{empl \mid (\forall t)(periodic_{7,0}(t) \Rightarrow (\exists t')(ATTEND(empl, \text{"CRC"}, t') \wedge t \leq t' < t+7 \wedge (\forall t'')(t \leq t'' < t+7 \wedge t'' \neq t' \Rightarrow \neg ATTEND(empl, \text{"CRC"}, t''))))\}$$

Note that the "once-a-week" events themselves may not be strongly periodic. However, the query that checks if events are held once a week is expressed in strongly periodic terms in the previous example, i.e. is expressed in $\mathbf{TC}_p$.

The next proposition, that immediately follows from the fact that periodic sets of time cannot be defined in $\mathcal{R}_L$ (see Section 2.1), shows that the predicate *periodic* adds extra expressive power to $\mathbf{TC}$.

Proposition 10 $\mathbf{TC}_p$ *has more expressive power than* $\mathbf{TC}$.

In addition to strongly periodic predicates, we can add nearly periodic predicates to $\mathbf{TC}$. The resulting query language will be called *nearly periodic* and will be denoted as $\mathbf{TC}_{np}$.

For instance, if `head_of_state_visits` is a monadic predicate defined in Example 2 specifying the times when the head of a state is scheduled to make visits and `ACTUAL_VISITS` is the predicate specifying the actual visits he or she made, then the query "Did the head of the state make all the visits as they were scheduled for him/her (with predicate `head_of_state_visits`)?" can be expressed in $\mathbf{TC}_{np}$ as a yes/no query

$(\forall t)(\text{head\_of\_state\_visits}(t) \Rightarrow \text{ACTUAL\_VISITS}(t))$

The language $\mathbf{TC}_{np}$ differs from $\mathbf{TC}_p$ in the following important aspect. $\mathbf{TC}_p$ is obtained from **TC** by adding a family of periodic predicates $periodic_{p,s}$ defined with equation (1). In contrast to this, $\mathbf{TC}_{np}$ is obtained from **TC** by adding all the nearly periodic predicates to **TC**. Since there are infinitely many such predicates that, unlike $\mathbf{TC}_p$, are defined in infinitely many ways, this means that all of them cannot be added to **TC** in practice. However, in practice, the user will add *finite* libraries of nearly periodic predicates to **TC** to support near-periodicity. We will elaborate on this further in Section 5.

The following proposition follows immediately from the definitions of $\mathbf{TC}_p$, $\mathbf{TC}_{np}$, and from Corollary 9:

Proposition 11 $\mathbf{TC}_{np}$ *has more expressive power than* $\mathbf{TC}_p$.

4 Relationship of $\mathbf{TC}_p$ and $\mathbf{TC}_{np}$ to Other Formalisms

Periodic queries can be expressed in some other formalisms besides $\mathbf{TC}_p$ and $\mathbf{TC}_{np}$. For instance, the $\mathbf{TC}_p$ query from Example 3 can also be expressed as:

$$\{< empl, t > \mid ATTEND(empl, meeting, t) \wedge meeting = \text{“CRC”} \wedge (\exists t')(t = 7 * t' + 2)\} \quad (4)$$

in an extension of **TC** that supports addition[6].

Since addition is a very fundamental mathematical operation and since some of the strongly periodic queries can be expressed using addition, we consider the extension of **TC** that supports addition, $\mathbf{TC}^+$, in the next section.

4.1 Calculus $\mathbf{TC}^+$

The calculus $\mathbf{TC}^+$ is defined as follows. It is identical in its syntax and semantics to **TC**, except that it supports one extra function *addition* (+) *only* for its temporal sort, and this function is defined in the standard way. The query specified in (4) provides an example of the question expressed in $\mathbf{TC}^+$.

[6]It is important to note that the multiplication in this example is really not a true multiplication; it can be replaced with 6 additions of t'.

As the following theorem shows, $\mathbf{TC}^+$ is more expressive than $\mathbf{TC}_p$.

Theorem 12 *The class of* $\mathbf{TC}_p$ *queries is properly contained in the class of* $\mathbf{TC}^+$ *queries.*

Proof: The containment follows from the fact that periodic predicates can be expressed in $\mathbf{TC}^+$. To prove the proper containment, consider the following $\mathbf{TC}^+$ query:

$$\{t \mid P(t) \wedge (\exists t')(\exists t'')(P(t') \wedge Q(t'') \wedge t + t' = t'')\} \tag{5}$$

We claim that this query cannot be expressed in $\mathbf{TC}_p$. To see this, assume that it can be expressed in $\mathbf{TC}_p$ as $\{t \mid \psi_{P,Q}(t)\}$. Consider the interpretations for P and Q that make these predicates true throughout. Then ψ is equivalent to a $\mathcal{R}_{L+}$ formula and hence is eventually periodic (Theorem 2) with some period p and some starting point M. Furthermore, it follows from the proof of Theorem 2 that if ψ has a periodic predicate $period_{q,s}(t)$ then q divides p.

Take the largest constant N among all the constants appearing in $\psi_{P,Q}$, and let t_0 be a multiple of p such that $max\{N, M, 1\} \ll t_0$ (e.g., let $t_0 = 100(M + N + 1)$). Consider an interpretation for Q such that Q is true in that interpretation at only one time, c, and false elsewhere, and that c is also a multiple of p and is much larger than t_0, $t_0 \ll c$ (e.g., let $c = 100t_0$).

Then the original $\mathbf{TC}^+$ query with this interpretation for Q is equivalent to $\gamma(t) \equiv P(t) \wedge (\exists t')(P(t') \wedge t + t' = c) \equiv P(t) \wedge P(c - t)$. Furthermore, $\psi_{P,Q}(t)$ with this interpretation of Q can be replaced by $\phi_P(t)$, where $\phi_P(t)$ is obtained from $\psi_{P,Q}(t)$ by replacing all predicate instances $Q(t')$ in it with the expression $t' = c$. Since we assume that the original $\mathbf{TC}^+$ query is equivalent to $\psi_{P,Q}$, it means that $\gamma(t)$ must be equivalent to $\phi_P(t)$, i.e., they must be equal at all times for all the interpretations for P.

Now consider the following two interpretations for P. The first interpretation, I_P, is true for all moments of time except $t_1 = c - t_0 - p$, and the second one, I_P^{+p}, is also true for all moments of time except for $t_1' = t_1 + p = c - t_0$ (see Figure 1). Clearly, $\gamma(t_0)$ is true in I_P, and $\gamma(t_0)$ is false in I_P^{+p}. Since γ_P and ϕ_P are equivalent, it should be the case that $\phi_P(t_0)$ is true in I_P and $\phi_P(t_0)$ is false in I_P^{+p}. However, we claim that if $\phi_P(t_0)$ is true in I_P then $\phi_P(t_0)$ must also be true in I_P^{+p}, and the resulting contradiction proves the theorem.

To see that if $\phi_P(t_0)$ is true in I_P then $\phi_P(t_0)$ must also be true in I_P^{+p}, we do the

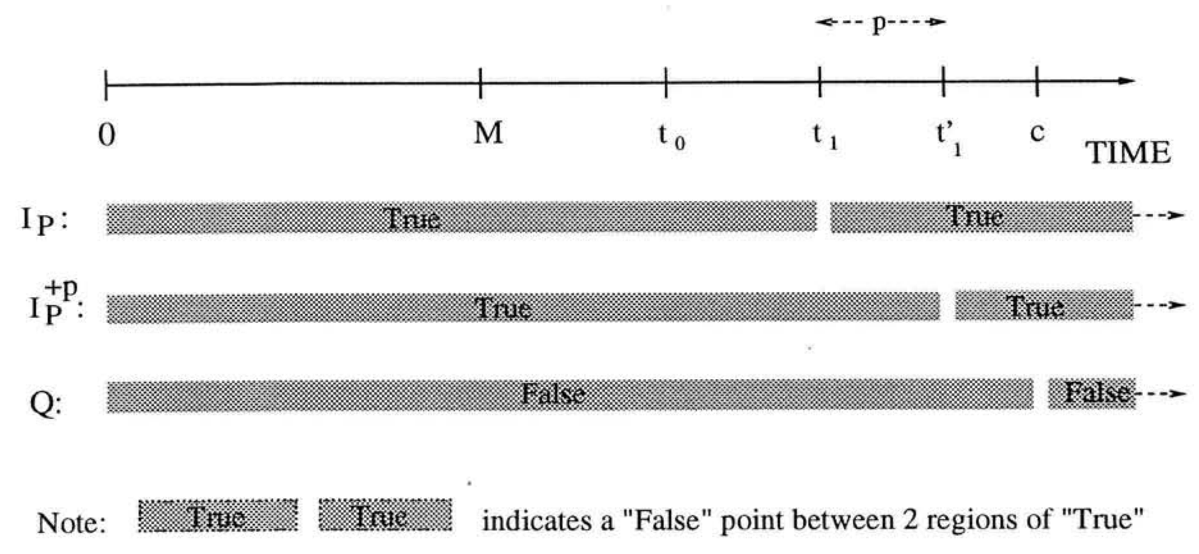

Figure 1: Theorem 9 Illustration

following. For interpretation I_P, we replace $\phi_P(t_0)$ with $\phi_1(t_0)$ obtained from $\phi_P(t_0)$ by replacing each instance of $P(t)$ with $t \neq t_1$. Similarly for interpretation I_P^{+p}, we replace $\phi_P(t_0)$ with $\phi_2(t_0)$ obtained from $\phi_P(t_0)$ by replacing each instance of $P(t)$ with $t \neq t_1 + p$. Note that ϕ_1 and ϕ_2 no longer contain any temporal predicates. The only difference between ϕ_1 and ϕ_2 is that whenever the first contains an expression of the form $t \neq t_1$, the second contains an expression of the form $t \neq t_1 + p$. It can be shown that if $\phi_1(t_0)$ is true then $\phi_2(t_0)$ is also true. Intuitively, this is the case because t_1 is located "far away" from the other constants appearing in the formula (based on our choice of t_1), and because for any periodic predicate $periodic_{q,s}(t)$ in ϕ_1 or ϕ_2, q divides p (therefore, periodic predicates do not "notice" the difference between $\phi_1(t_0)$ and $\phi_2(t_0)$).

□

Note that $\mathbf{TC}^+$ is more expressive than $\mathbf{TC}_p$, whereas $\mathcal{R}_{L+}$ and $\mathcal{R}_{Lp}$ have the same expressive power. Intuitively, this happens because the language $\mathcal{R}_{L+}$ admits quantifier elimination (that is how Theorem 2 is proved in [10]), whereas $\mathbf{TC}^+$ does not. From another point of view, $\mathbf{TC}^+$ is more expressive than $\mathbf{TC}_p$ because temporal variables appearing in different predicates can be added together in $\mathbf{TC}^+$, as is done in query (5). Note that if we just added to $\mathbf{TC}$ purely temporal relations defined in $\mathcal{R}_{L+}$, then the resulting language would have had the expressive power of $\mathbf{TC}_p$ because of Theorem 2. This observation shows that the issue of query expressiveness of periodic extensions of $\mathbf{TC}$ cannot be trivially reduced to studying periodic sets of time.

The next theorem shows that $\mathbf{TC}_{np}$ and $\mathbf{TC}^+$ are unrelated in the sense that neither one contains the other.

Theorem 13 *There are* $\mathbf{TC}_{np}$ *queries that cannot be expressed in* $\mathbf{TC}^+$*, and* $\mathbf{TC}^+$ *queries that cannot be expressed in* $\mathbf{TC}_{np}$.

Proof: To prove the first part, consider the following query:

$$\{t \mid (\exists k)(t = k * p + (-1)^k \frac{1}{k})\}$$

where p is an integer greater than 1. Clearly, this query belongs to $\mathbf{TC}_{np}$. However, this set is not strongly periodic. Therefore, it cannot be expressed in $\mathcal{R}_{L+}$ (Corollary 3) and thus in $\mathbf{TC}^+$.

The proof of the second part of the query proceeds as the proof of Theorem 12. We start with the same $\mathbf{TC}^+$ query (5) and assume that it can be expressed in $\mathbf{TC}_{np}$ as $\{t \mid \psi_{P,Q}(t)\}$. Then we use the same argument reducing $\psi_{P,Q}$ to ϕ_1 and ϕ_2. However, the constants t_0, t_1, and c have to be selected now differently from the way they were selected in the proof of Theorem 12. First, we select the point t_0 that is "much larger" than any constant appearing in $\psi_{P,Q}$. Then we select the points t_1 and t_2, such that $t_0 < t_1 < t_2$, and all the nearly periodic functions in $\psi_{P,Q}$ are either simultaneously true or false at t_1 and t_2. Then we select the point $c = t_0 + t_2$ and set $p = t_2 - t_1$. Then the rest of the argument in the proof of Theorem 12 goes through. □

It follows from Theorem 12 that $\mathbf{TC}^+$ can express periodic queries. Since $\mathbf{TC}^+$ is a very fundamental, intuitive, and convenient language for expressing periodicity, it would be very nice to use it for that purpose. However, it also follows from Theorem 12 that $\mathbf{TC}^+$ is "too powerful" for defining periodicity, i.e., it captures more than the concept of periodicity alone. Therefore, if we want to use $\mathbf{TC}^+$ for expressing periodic and only periodic queries, we should impose certain syntactic conditions on $\mathbf{TC}^+$ that would restrict it to only periodic queries.

As we said already, the main reason why $\mathbf{TC}^+$ is more expressive than $\mathbf{TC}_p$ is that in $\mathbf{TC}^+$ we can add two *different* temporal variables, whereas in $\mathbf{TC}_p$, periodicity can be defined only in terms of the addition of a variable to itself. For example, $\mathbf{TC}^+$ formulae containing expressions $t + t' = t''$ cannot always be expressed in $\mathbf{TC}_p$ as the proof of Theorem 12 shows. However, if a $\mathbf{TC}^+$ formula containing expressions $t + t' = t''$ does not contain temporal relations, then it can be expressed in $\mathbf{TC}_p$ (by eliminating quantifiers, as explained in Theorem 32E in [10], and converting the resulting eventually periodic expression into $\mathbf{TC}_p$ expressions).

This discussion suggests the following approach to restricting $\mathbf{TC}^+$ to the class of formulae expressible in $\mathbf{TC}_p$. Intuitively, if ϕ is a $\mathbf{TC}^+$ expression, then we want to "separate" all of its subformulae containing expressions of the form $t + t'$ into purely temporal expressions that do not contain any temporal relations. Then we can remove $t + t'$ expressions through the quantifier elimination process (as described in the proof of Theorem 32E in [10]) thus producing expressions containing only congruence relations that can be easily expressed in $\mathbf{TC}_p$.

Formally, we proceed as follows.

Definition: A $\mathbf{TC}^+$ expression ϕ is *essentially periodic* if it is equivalent to another $\mathbf{TC}^+$ expression ϕ' having the following property:

> If ϕ' contains an atomic formula defined with a relational operator $\theta(t_1, t_2, \ldots, t_n)$ (θ is $<, =, \leq$, etc.) that depends on distinct variables $t_1, t_2, \ldots, t_n$, and $n > 1$, then for all variables t_i, except one, the subexpression of ϕ' of the form $(\mathbf{Q}t_i)\psi$, where $\mathbf{Q}$ is a quantifier, does not contain any temporal predicates.

For example, the formula

$$(\forall t)(\exists t')(P(t) \wedge t + t' = c)$$

is essentially periodic because it is equivalent to $(\forall t)(P(t) \wedge (\exists t')(t + t' = c))$, and its subexpression $(\exists t')(t + t' = c)$ does not contain any predicates. Therefore, the quantifier in the latter subexpression can be eliminated by replacing the entire subexpression with $t \leq c$. Thus, the main formula is equivalent to the $\mathbf{TC}$ (and hence $\mathbf{TC}_p$) formula

$$(\forall t)(P(t) \wedge t \leq c)$$

As another example, the formula

$$(\forall t)(\exists t')(P(t) \wedge Q(t') \wedge t + t' = c)$$

is *not* essentially periodic. To see that this is so, consider its atomic subformula $t + t' = c$. Both the subexpressions $(\exists t')(P(t) \wedge Q(t') \wedge t + t' = c)$ and $(\forall t)(\exists t')(P(t) \wedge Q(t') \wedge t + t' = c)$ contain a temporal predicate ($P(t)$ and $Q(t')$) inside. Furthermore, this formula cannot be converted to an equivalent formula that satisfies this property.

We denote the class of essentially periodic queries as $\mathbf{TC}_e^+$.

Proposition 14 *Classes of queries* $\mathbf{TC}_p$ *and* $\mathbf{TC}_e^+$ *have the same expressive power.*

Proof: Clearly, any $\mathbf{TC}_p$ query can be expressed in $\mathbf{TC}_e^+$. To prove the inverse, consider an essentially periodic query ϕ. Transform it into the equivalent query ϕ' as described above. Then for each relational operator $\theta(t_1, t_2, \ldots, t_n)$ having more than one distinct variable in it ($n > 1$) take the subexpression $(\mathbf{Q}t_i)\psi$ containing the scope of the outermost quantifier (i.e. quantifiers of all other variables t_j, except one are contained inside ψ). By the definition of an essentially periodic query, ψ does not contain any predicates. Using the quantifier elimination procedure, as described in [10, Theorem 32E], convert it into a subexpression that contains only one variable from the set $\{t_1, t_2, \ldots, t_n\}$ and the congruence operator (in other words, the expression ψ containing quantifiers over temporal variables is replaced by an equivalent expression containing the congruence operator and no quantifiers). Repeat this process inductively for all other relational operators $\theta(t_1, t_2, \ldots, t_n)$ having more than one distinct variable. As a result of this procedure, the addition operator can appear only in terms having only a single variable, and these terms can be defined with periodic predicates. $\square$

In summary, we have considered language $\mathbf{TC}^+$ in this section and have shown that it is more expressive than $\mathbf{TC}_p$ and is unrelated to $\mathbf{TC}_{np}$. Furthermore, we have defined a restricted version of $\mathbf{TC}^+$, $\mathbf{TC}_e^+$, that is equivalent to $\mathbf{TC}_p$, and thus captures strongly periodic queries and only them. These results are summarized in Figure 2, where arrows mean "proper inclusion."

Since languages $\mathbf{TC}_{np}$ and $\mathbf{TC}^+$ are unrelated to each other, it is important to know if there is a language that supports periodicity and at the same time contains both $\mathbf{TC}_{np}$ and $\mathbf{TC}^+$. We address this issue in the next section.

$$\begin{tikzcd}
 & & \mathrm{TC}_{np} \\
\mathrm{TC} \arrow[r] & \mathrm{TC}_p = \mathrm{TC}_e^+ \arrow[ur] \arrow[dr] & \\
 & & \mathrm{TC}^+
\end{tikzcd}$$

Figure 2: The Relationship Between Calculi $\mathbf{TC}$, $\mathbf{TC}_p$, $\mathbf{TC}_e^+$, $\mathbf{TC}_{np}$, and $\mathbf{TC}^+$ (Arrows Indicate Inclusion).

4.2 Calculus $\mathbf{TC}^{+*}$

In this section we define the calculus $\mathbf{TC}^{+*}$ and show that it is more expressive than each of the calculi $\mathbf{TC}^+$ and $\mathbf{TC}_{np}$. The calculus $\mathbf{TC}^{+*}$ is obtained from $\mathbf{TC}^+$ by supporting the *multiplication* operator for the temporal sort. For example, the query "find the people who attended CRC meetings on non-prime days and report their attendance dates" can be expressed in $\mathbf{TC}^{+*}$ as

$$\{< empl, t > \mid ATTEND(empl, meeting, t) \wedge meeting = \text{"CRC"} \wedge (\exists t')(\exists t'')(t = t' * t'' \wedge t' \neq 1 \wedge t' \neq t)\}$$

As the following propositions show, $\mathbf{TC}^{+*}$ is more expressive than $\mathbf{TC}_{np}$ and $\mathbf{TC}^+$.

Proposition 15 $\mathbf{TC}^{+*}$ *is more expressive than* $\mathbf{TC}^+$.

Proof: Immediately follows from Corollary 32G in [10] that says that multiplication cannot be defined in $(N, 0, S, <, +)$. For example, the $\mathbf{TC}^{+*}$ query $\{t \mid (\exists s)(s \in N \wedge t = s * s)\}$ does not define a strongly periodic set and therefore is not expressible in $\mathbf{TC}^+$. □

Proposition 16 $\mathbf{TC}^{+*}$ *is more expressive than* $\mathbf{TC}_{np}$.

Proof: By definition, the nearly periodic predicates are recursive. Then Theorem 34A from [10] says that they are expressible in $(N, 0, S, <, +, *, E)$ (where E stands for exponentia-

$$\begin{tikzcd}
 & & \mathrm{TC}_{np} \arrow[dr] & \\
\mathrm{TC} \arrow[r] & \mathrm{TC}_p = \mathrm{TC}_e^+ \arrow[ur] \arrow[dr] & & \mathrm{TC}^{+*} \\
 & & \mathrm{TC}^+ \arrow[ur] &
\end{tikzcd}$$

Figure 3: The Relationship Between Calculi **TC**, $\mathbf{TC}_p$, $\mathbf{TC}_e^+$, $\mathbf{TC}_{np}$, $\mathbf{TC}^+$, and $\mathbf{TC}^{+*}$ (Arrows Indicate Inclusion).

tion). Then Theorem 37C in [10] says that they are also expressible in $(N, 0, S, <, +, *)$. Furthermore, it follows from Theorem 13 and Proposition 15 that the inclusion is proper. □

4.3 Discussion

Propositions 15 and 16 complete the picture that relates the various periodic calculi described in this paper. This relationship is presented in Figure 3.

Figure 3 shows that the languages $\mathbf{TC}_p$ and $\mathbf{TC}_e^+$ are equivalent and are the least powerful periodic query languages. They define *exactly* the class of strongly periodic queries.

The query language $\mathbf{TC}_{np}$ that defines the class of nearly periodic queries is strictly more expressive than the class of periodic queries. However, if the user wants to formulate a nearly periodic query, he or she has to specify nearly periodic predicate(s), such as *head_of_state_visits*, that are part of the query. This means that the nearly periodic predicates must be stored in a library that is either a part of the DBMS or is created by the end-user or the systems administrator. Furthermore, the user has to deal with the situations when the nearly periodic predicates he or she needs are not in the library.

Therefore, we also studied some of the other formalisms for expressing periodicity that do not have to deal with libraries of predicates, such as languages $\mathbf{TC}^+$ and $\mathbf{TC}^{+*}$. The language $\mathbf{TC}^+$ is strictly more expressive than $\mathbf{TC}_p$ and is very simple: it differs from **TC** in that it also supports addition for the temporal sort. However, it does not express all

$\mathbf{TC}_{np}$ queries. Finally, the language $\mathbf{TC}^{+*}$ can support both periodic and nearly periodic queries (and is strictly more expressive than $\mathbf{TC}^{+}$). However, it may be the case that $\mathbf{TC}^{+*}$ is overly expressive, i.e. it supports various other concepts in addition to periodicity; but the study of the exact characterization of the expressive power of $\mathbf{TC}^{+*}$ is beyond the scope of this paper.

5 Adding Periodicity to TSQL2

In the previous sections of the paper we have formalized the two intuitive concepts of periodicity borrowed from [1] and presented in the introduction. We have also considered various periodic extensions of **TC** that support the two types of periodicity, and we have established the relationships among these extensions. We are now ready to discuss how periodicity can be added to end-user-oriented temporal query languages and to the TSQL2 standard [25] in particular. We concentrate in this section on TSQL2 because it is an important and highly influential temporal query language. However, our discussions of how to add periodicity to this language are more general in nature and can be applied to various other ungrouped temporal query languages, such as TQuel [24] or TSQL [22]. Moreover, we believe that some of these ideas can be extended to the grouped temporal query languages, such as L_h [8], and Gadia's [11] and Tansel's [28] languages.

To make the discussion in this section concrete, we need to consider a specific relation embodying some data that can support periodic queries. For this purpose consider a relation which records attendance by employees at various company meetings. The TSQL2 command to create such a table might be the following:

```
CREATE TABLE ATTEND-2 (EMPL_NAME CHARACTER ( 20 ) NOT NULL,
    MEETING_NAME CHARACTER ( 20 ))
VALID AS INTERVAL
```

Figure 4 shows the example instance of the **ATTEND-2** relation which we will use in this section to illustrate periodic queries.

In its present form, the TSQL2 proposal does not support periodic queries for the following reason. TSQL2 is based on the interval-based temporal logic and supports such Allen's interval-based operators as *precedes*, *contains*, *overlaps*, etc. [2]. Since temporal

ATTEND-2			
EMPL_NAME	*MEETING_NAME*	*start-time*	*end-time*
Tom	CRC	1/4/93 10:30	1/4/93 13:00
Tom	CRC	1/11/93 9:30	1/11/93 12:00
Tom	CRC	1/18/93 10:30	1/18/93 13:00
Tom	CRC	1/25/93 13:30	1/25/93 15:00
Tom	CRC	2/1/93 9:30	2/1/93 12:00
Tom	CRC	2/8/93 10:30	2/8/93 13:00
Tom	CRC	2/15/93 10:30	2/15/93 13:00
Tom	CRC	2/22/93 13:30	2/22/93 15:30
Susan	CRC	1/18/93 11:30	1/18/93 13:00
Susan	CRC	1/3/94 11:00	1/3/94 13:00
Susan	CRC	1/4/94 11:30	1/4/94 13:30
Susan	P&T	9/18/93 14:30	9/18/93 16:00
Susan	P&T	10/20/93 14:00	10/20/93 16:30
Susan	P&T	3/21/94 14:00	3/21/94 16:30
Susan	P&T	4/13/94 10:30	4/13/94 13:30

Note that the following dates in this table are Mondays: 1/4/93, 1/11/93, 1/18/93, 1/25/93, 2/1/93, 2/8/93, 2/15/93, 2/22/93, 1/3/94, 3/21/94.

Figure 4: Interval Relation **ATTEND-2**

logic, including the interval-based one, cannot support periodic queries [34], it follows that TSQL2 cannot support these queries either.

Based on the theoretical discussions presented in Sections 3 and 4, periodic queries can be added to TSQL2 (or to any other ungrouped temporal query language) in one of the following ways:

1. by adding additional temporal operators that support periodicity to the temporal operators already existing in TSQL2;

2. by allowing explicit references to time in TSQL2, including temporal variables and quantification over them, and by supporting arithmetic on the temporal domain by using such operators as $+$, $\equiv$, and $*$;

3. by supporting a set of periodic functions, including "calendar" functions.

The first alternative can be implemented by adding congruence operators to temporal logic operators, as has been done for the single temporal attribute case in *timed* temporal logic [3]. However, since in this paper we considered periodicity within the framework of first-order logic with explicit references of time, this proposal to extend temporal logic is beyond the scope of this paper.

The second alternative can support periodicity by allowing the following additions to TSQL2. In its current form, TSQL2 allows references to the initial and final endpoints of a temporal interval as **BEGIN**(<*tuple-variable*>) and **END**(<*tuple-variable*>). However, it does not allow any temporal variables and quantification over these variables, as **TC** does. Periodicity can be supported in TSQL2 by allowing such variables and quantifications over them, *and* by allowing arithmetic operators such as addition $(+)$, congruence $(\equiv)$ (or predicate *periodic*), and multiplication $(*)$.

Example 5 Consider the query from Example 3 "Find the names and attendance times of people who attended any of the Monday meetings of the Computer Resources Committee (CRC)." This can be expressed in such an extension of TSQL2 as

EMPL_NAME	*start-time*	*end-time*
Tom	1/4/93 10:30	1/4/93 13:00
Tom	1/11/93 9:30	1/11/93 12:00
Tom	1/18/93 10:30	1/18/93 13:00
Tom	1/25/93 13:30	1/25/93 15:00
Tom	2/1/93 9:30	2/1/93 12:00
Tom	2/8/93 10:30	2/8/93 13:00
Tom	2/15/93 10:30	2/15/93 13:00
Tom	2/22/93 13:30	2/22/93 15:30
Susan	1/18/93 11:30	1/18/93 13:00
Susan	1/3/94 11:00	1/3/94 13:00

Figure 5: Answer to the "Monday-CRC" Query.

SELECT EMPL_NAME
FROM ATTEND-2 A
WHERE MEETING = 'CRC' **AND** periodic(week,Monday,T)
AND BEGIN(A) $\leq$ T $<$ END(A))

This query would return the result in Figure 5.

□

In this query, `periodic(week,Monday,T)` is the periodic predicate defined in (1), BEGIN(A) and END(A) are TSQL2 functions that specify the initial and final endpoints of a temporal interval of the tuple referenced by A. Note that the variable T in the **WHERE** clause is implicitly existentially quantified, as is the standard practice in relational calculi. Its purpose in the query is to check whether the lifespan of the tuple referenced by A contains Mondays.

It follows from the theoretical considerations described in Sections 3 and 4 that different types of arithmetic operators added to TSQL2 support different types of periodicity. For instance, if we add only the predicate *periodic* or the congruence operator ($\equiv$) to TSQL2, then we can express only strongly periodic queries in it. If we add the addition operator ($+$), then we can support strongly periodic queries and more (as discussed in Section 4), but we cannot support nearly periodic queries. To be able to express nearly periodic queries in TSQL2, we have to add addition and multiplication to the language. However, by adding these two operators, we can express more than nearly periodic queries in TSQL2 (Proposition 16). Note that the advantage of adding to TSQL2 temporal variables and arithmetic operators over them lies in its simplicity: we can express both types of periodicity in it using only a few operators (one or two operators depending on the specific choice made).

EMPL_NAME	*start-time*	*end-time*
Tom	1/4/93 10:30	1/4/93 13:00
Tom	2/1/93 9:30	2/1/93 12:00
Susan	1/3/94 11:00	1/3/94 13:00

Figure 6: Answer to the "First-Monday-of-each-month" Query.

However, this approach can be quite user unfriendly in some cases. For example, the definition of predicate `first_date_of_each_month`, as presented in Example 1, requires addition of 12 terms. Such long and cumbersome expression can confuse the user and can result in specification errors. Therefore, designers of TSQL2 may consider the third approach to adding periodic queries to the language.

The third approach addresses some of the deficiencies of the second approach by supporting *user-defined* periodic functions, including extensions to TSQL2 calendars. These functions can be either strongly or nearly periodic and can be divided into the following types:

- the set of basic periodic functions provided by the vendors, such as *every_week*, *every_month*, *first_day_of_each_month*, and other periodic functions that the vendor finds the most commonly used in practice;
- arbitrary user-defined periodic functions[7].

Example 6 For example, if the predicate *first_Monday_of_each_month* is located in a user-defined library of periodic predicates, then the query "Give me the names and attendance times of people who attended any of the CRC meetings held on the first Monday of each month" can be expressed as

SELECT EMPL_NAME
FROM ATTEND-2 A
WHERE MEETING = 'CRC' **AND** (first_Monday_of_each_month(T)
AND BEGIN(A) $\leq$ T $<$ END(A))

This query would return the result in Figure 6.

□

[7]By "user" we mean here either the end-user of the temporal DBMS or a systems administrator. Both of them can define periodic functions.

In order for users to define arbitrary periodic functions, such as `head_of_state_visits` or `third_day_of_each_month`, TSQL2 should support a language for that purpose. Such a language can include the operators of the *calendar expression language* of [6] or *slices* of [23]. For example, the function `third_day_of_each_month` can be defined as `months + 3.days` in the language of [23], where `months` and `days` are calendars, and the expression `3.days` selects the third day in a month. However additional constructs may also be added to this language in order for it to support arbitrary user-defined periodic functions in an efficient manner. Moreover, as we demonstrated in Section 4 (and in particular in Proposition 16), this language should also have the operators of addition and multiplication for the temporal sort.

One obvious advantage of this approach is that it is more user-friendly than the first one. However, its limitation is that users many need large libraries of user-defined (and vendor-supplied) functions to support their needs. Also, if we restrict these user-defined periodic functions to nearly-periodic functions, then, as it was shown in Proposition 16, this approach does not take us beyond the second proposal of adding arithmetic operators and temporal variables to the query language because it does not add any expressibility to periodic queries.

To address deficiencies of the second and third approaches of adding periodicity to the TSQL2 standard, we propose a *combination* of the two approaches. In other words, TSQL2 can support explicit references to time and arithmetic operators on the temporal domain, as was advocated in the second approach, *and* also support libraries of user-defined periodic functions. By combining the two approaches, TSQL2 would be able to express any nearly periodic query and would also have the user-friendliness that comes with the libraries of periodic functions. Although this approach "over-supplies" the user with various periodic operators, it gives him or her the ability to express periodic queries concisely. Therefore, we would advocate this approach as the approach to adding periodicity to the TSQL2 standard.

Finally, we note that this proposal for integrating periodicity into TSQL2 fits quite smoothly with other of its extended temporal aspects, such as temporal aggregates [17]. This is so because temporal aggregation in TSQL2 is applied *after* the TSQL2 temporal selection and projections are performed. In other words, periodicity is treated here essentially as part of the temporal selection process, in that the periodic predicates and operators determine

EMPL_NAME	*COUNT(T)*
Tom	8
Susan	2

Figure 7: Result of COUNT Aggregate Query

whether or not a particular time-stamped "fact" in a relation is eligible to participate in the answer to a query; thus they appear in the WHERE clause of TSQL2. The aggregation process occurs only *after* any periodic predicates and operators are satisfied, and hence this process mashes nicely with this treatment of periodicity.

In [17] it is noted that, as in ordinary (non-temporal) aggregation, it is useful to distinguish two principal types of temporal aggregation, viz. "aggregation via selection" and "aggregation via computation." We illustrate an example of each of these two types in the context of a query involving periodicity.

Example 7 The query "How many Monday meetings of the CRC did each employee attend" can be expressed in TSQL2 as

```
SELECT SNAPSHOT  EMPL_NAME ,COUNT(*)
FROM             ATTEND-2 A
WHERE            MEETING_NAME = 'CRC' AND
                 periodic(week,Monday,T) AND
                 BEGIN(A) ≤ T < END(A))
GROUP BY         EMPL_NAME
```

This query would return the result in Figure 7, and does the temporal aggregation by *computing* (counting) the number of meetings each employee attended. Therefore, [17] calls this type of temporal aggregation "aggregation via computation."

□

Example 8 The query "Who attended the first Monday meeting of the CRC" can be expressed in TSQL2 as

EMPL_NAME
Tom

Figure 8: Result of MIN Aggregate Query

```
SELECT SNAPSHOT  EMPL_NAME
FROM             ATTEND-2 A
WHERE            MEETING_NAME = 'CRC' AND
                 periodic(week,Monday,T) AND
                 BEGIN(A) ≤ T < END(A)) AND
                 T = (
          SELECT          MIN(T1)
          FROM            ATTEND-2 A2
          WHERE           MEETING_NAME = 'CRC' AND
                          periodic(week,Monday,T1) AND
                          BEGIN(A2) ≤ T1 < END(A2) )
```

This query would result in the relation in Figure 8 which is obtained by *selecting* the time of the first Monday meeting of the CRC committee and then returning the names of the persons present at that meeting. Therefore, [17] calls this type of temporal aggregation "aggregation via selection."

□

6 Summary

As discussed in Section 1, periodicity, despite its importance, has not been studied in a comprehensive manner in the temporal database literature. One result of this is that periodicity is not currently supported in the commercially oriented temporal query languages, including the TSQL2 standard. To address this omission, we have explored in this paper the meaning of the notion of periodicity in the context of temporal databases, and have demonstrated how it can be supported in a practical way in end-user-oriented temporal query languages, such as TSQL2.

In order to understand what periodicity means in temporal databases, we consulted a dictionary [1] and identified two different intuitive meanings of periodicity that are related to temporal databases, i.e. "having periods or repeated cycles" and "happening or appearing at regular intervals." To define these concepts formally, we started with the languages $\mathcal{R}_L$ and $\mathbf{TC}$ for representing sets of time and temporal queries, respectively, and considered various periodic extensions of these languages. We examined how well these extensions capture the two types of periodicity described above. This means that for the extensions of $\mathcal{R}_L$ we dealt with the *data expressiveness* of these extensions and for the extensions of $\mathbf{TC}$ we dealt with the *query expressiveness* of the extensions.

At the data expressiveness level, we identified two types of periodicity, i.e. *strong* and *near* periodicity. Strong periodicity can be captured by periodic predicates, by the congruence relation, or by the addition operator. Near periodicity is a broader concept and can be captured by various nearly periodic predicates. At the query expressiveness level, we considered periodic extensions $\mathbf{TC}_p$, $\mathbf{TC}_{np}$, $\mathbf{TC}^+$, $\mathbf{TC}^+_e$, and $\mathbf{TC}^{+*}$ that are obtained from $\mathbf{TC}$ by adding different types of periodic operators described above. The relationship among these languages was summarized in Figure 3.

Having studied on a theoretical level what the concept of periodicity means in the database context, it becomes feasible to add periodicity in a comprehensive way to commercially oriented temporal query languages. In particular, we considered various ways of adding periodicity to TSQL2 and concluded that the most practical and theoretically sound method would be to allow temporal variables, arithmetic operations over the temporal domain, and libraries of vendor- and user-defined periodic functions, including calendar functions. Although these periodic functions are not strictly necessary (they can be implemented with arithmetic operations), they add user-friendliness to the language.

Although this paper focused on the issue of querying the database with respect to the periodicity of the data that it contains, there are other aspects of periodicity which can be of interest in the database context. For example, there is the notion of being able to *schedule* something to occur at every time in some previously defined periodic set of times. In an active database or in a conceptual modeling language one might want to schedule a particular update to be made automatically "every Wednesday," or a particular report to be automatically generated "the first day of each month." Such functionality has been added to the active database Ode [12] and to the specification language Templar [30]. For example, the rule "If a person is a member of a club, he or she must attend the club meetings held on Fridays before his or her club membership expires" can be expressed in Templar as

```
when      every Friday
before    membership_expiration_date(person)
if        club_member(person)
then-do   attend_club_meeting(person)
```

Also, as pointed out in [13], there are many interesting issues related to the interrelationships between the two different dimensions of time, *valid time* and *transaction time*, which have been proposed as the primary temporal dimensions of data ([27, 14]). Issues related to how periodicity, either strong or near, might be interrelated in one or both of these dimensions, or of how it could be incorporated into the schema, query, or active components of a DBMS, are all subjects for further research. Finally, we plan to work on the implementation issues and on the query processing and the optimization strategies for periodic queries.

Acknowledgments

The authors would like to thank the reviewers for their valuable comments which have helped to improve the presentation of this paper.

References

[1] *The American Heritage Dictionary.* Houghton Mifflin Company, second edition, 1985.

[2] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[3] R. Alur and T. A. Henzinger. A really temporal logic. In *Proceedings of the ACM Symposium on the Foundations of Computer Science*, 1989.

[4] T. L. Anderson. Modeling time at the conceptual level. In P. Scheuermann, editor, *Proceedings of the International Conference on Databases: Improving Usability and Responsiveness*, pages 273–297, Jerusalem, Israel, June 1982. Academic Press.

[5] M. Baudinet, J. Chomicki, and P. Wolper. Temporal deductive databases. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases.* Benjamin/Cummings, 1993.

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

[7] J. Chomicki and T. Imielinski. Finite representations of infinite query answers. *ACM Transactions on Database Systems*, 18(2):182–223, 1993.

[8] J. Clifford, A. Croker, and A. Tuzhilin. On completeness of historical query languages. *ACM Transactions on Database Systems*, 19(1), 1994.

[9] C. E. Dyreson and R. T. Snodgrass. Temporal indeterminacy in TSQL2. Commentary, TSQL2 Design Committee, September 1994.

[10] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972. New York.

[11] S. K. Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[12] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In *Proceedings of the ACM SIGMOD Conference*, pages 81 – 90, 1992.

[13] C. S. Jensen and R. Snodgrass. Temporal specialization and generalization. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):954–974, 1994.

[14] C.S. Jensen, J. Clifford, S.K. Gaida, A. Segev, and R.T. Snodgrass. A glossary of temporal database concepts. *ACM SIGMOD Record*, 21(3), September 1992.

[15] Jensen, C., Clifford, J., et al. A consensus test suite of temporal database queries. In R. Snodgrass, editor, *The TSQL2 Language Specification.* Tucson, AZ, October 1993.

[16] F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling infinite temporal data. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 392–403, 1990.

[17] Kline, N., Snodgrass, R. T. Aggregates in TSQL2. Available by anonymous ftp from **cs.arizona.edu**, June 1994.

[18] F. Kroger. *Temporal Logic of Programs.* Springer-Verlag, 1987. EATCS Monographs on Theoretical Computer Science.

[19] N.A. Lorentzos and R.G. Johnson. Extending relational algebra to manipulate temporal data. *Information Systems*, 13(3):289–296, 1988.

[20] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[21] E. McKenzie and R. Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.

[22] S. B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49:147–175, 1989.

[23] M. Niezette and J.M. Stevenne. An efficient symbolic representation of periodic time. In *Proceedings of the 1st International Conference on Information and Knowledge Management*, Baltimore, Maryland, 1992.

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

[25] R. T. Snodgrass, editor. *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, 1995.

[26] M.D. Soo and R.T. Snodgrass. Multiple calendar support for conventional database management systems. Technical Report TR 92-07, University of Arizona, Department of Computer Science, 1992.

[27] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases.* Benjamin-Cummings, 1993.

[28] A.U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[29] W. Thomas. A combinatorial approach to the theory of ω-automata. *Information and Control*, 48:261–283, 1981.

[30] A. Tuzhilin. Templar: A knowledge-based language for software specifications using temporal logic. *ACM Transactions on Information Systems*, 13(3), 1995.

[31] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In *Proceedings of the International Conference on Very Large Databases*, pages 13–23, 1990.

[32] J.F.A.K. van Benthem. *The Logic of Time.* D. Reidel Publishing Company, 1983.

[33] M. Y. Vardi. A temporal fixpoint calculus. In *Proceedings of the Fifteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 250–259, 1988.

[34] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–99, 1983.