# A lattice of classes of user-defined symbolic periodicities

Lavinia Egidi and Paolo Terenziani

Dipartimento di Informatica
Università del Piemonte Orientale *A. Avogadro*
Spalto Marengo, 33 - 15100 Alessandria AL - Italy
{lavinia.egidi, paolo.terenziani}@mfn.unipmn.it

## Abstract

*User-defined calendars and periodicities are gaining an increasing relevance in AI and DB theory and applications. Thus several representation languages have been introduced to model them. Even focusing the attention on* symbolic *languages only, several different proposals can be chosen, and the comparisons between them are not trivial at all. In this paper, we propose a lattice of properties about periodicity and use it in order to classify different symbolic approaches in the literature. We then propose a language which covers all the properties in the lattice.*

**Keywords:** user-defined periodicity, symbolic languages, classification

## 1. Introduction

Calendars and periodicities are an intrinsic part of the human way of approaching reality. They play a fundamental role in many applications, spanning from financial trading to scheduling, from manufacturing and process control to office automation and data broadcasting. In many practical applications, supporting a "standard" calendar (i.e., the Gregorian calendric system) does not suffice, since *user-defined* periodicities need to be used. For instance, Soo and Snodgrass [13] emphasized that the use of a calendar depends on the cultural, legal and even business orientation of the user, and listed many examples of different calendric systems. Both in Databases (DB) and in Artificial Intelligence (AI), many different approaches (see, e.g., the survey in [15]) have been devised to deal with periodicity and its related notions of *granularity* and *calendric systems* [2, 13].

Roughly speaking, we can classify these approaches into three mainstreams: *general-purpose (logical)* languages, with (minimal) extensions to cope with periodicity (e.g., [4] and approaches in classical temporal logics), *constraint-based* languages (terminology from [1]), using mathematical formulae (and constraints on variables) to model periodicity (e.g., Kabanza's *linear repeating points* [8]), and *symbolic* languages.

In this paper, we focus only on *symbolic* languages to deal with *user-defined* periodicities. Symbolic languages usually provide a limited set of "high-level" and "user-friendly" operators to let users define the periodicities in a "commonsense" and incremental way.

One of the main goals of our work is that of designing a symbolic language expressive enough to cover a wide range of user-defined periodicities, such as those in the following examples:

**Examples.**

1. "WorkingHours", defined as the intervals from 8 a.m. to 18 p.m, each day,
2. "Mary's and Tom's shifts" (e.g., Mary: from 8 to 12 and from 14 to 18 -intended as a unique interval of time with a two-hour gap in it; Tom: the same as Mary)
3. "*January 2, 2003, from 13 to 14*, plus *Mary's and Tom's shifts*, grouped by months"

Notice that dealing with the above examples means providing a language powerful enough to express periodicities having a wide range of different properties. For instance the periodicity in the first example consists of non consecutive intervals; in the second periodicity, non-consecutive intervals have internal gaps, and some of them overlap (notice that repetition, i.e., multiple occurrences of the same interval, is a special case of overlap); in the third periodicity we add further complexity with respect to the second one, since it also involves an aperiodic part, and "structure", in the sense that time intervals are now viewed as grouped month by month.

Different symbolic languages have been recently presented in the literature, covering (at least partially) several of the above properties.

Unfortunately, however, despite their claimed simplicity, symbolic languages in the literature usually lack a clear semantics for their operators, which sometimes seem to have unexpected "side-effects". [1]

Also, in some cases, it is not easy to understand the extent of their expressiveness. [2]

One of the main goals of our overall approach is that of designing an expressive symbolic language overcoming such limitations. Specifically, the contribution of the approach we describe in this paper is twofold:

- We provide a homogeneous framework and an expressiveness criterion that we use in order to compare and classify different approaches. We do this by identifying a set of orthogonal properties that characterize user-defined periodicities; based on such properties we define a lattice of sets of periodicities (see Section 2). The properties are defined mostly following homogeneity considerations, that turn out to agree with intuition. The aim is to select periodicities that have a clear semantical meaning.

  In Section 3, we use the lattice in order to classify different symbolic languages in the literature, on the basis of the properties they can deal with, and show that none can handle all the aspects we take into account. (We discuss further approaches in the conclusions, Sec. 5.)

- We propose (in Section 4) a symbolic language which covers the top of the lattice. In other words, our language can represent periodicities for which all of the properties may hold. For example, our approach is (to the best of our knowledge) the only one coping with all forms of overlaps, including multiple identical intervals (which we call *repetitions*, in the following).

  We tailored our language so that each operator corresponds exactly to one of the properties (with no "side effect"). This enhances clarity, although it might not be of practical use as it is. It should be taken mostly as a proof of concept.

In a related paper [6], we propose a mathematical characterization of the semantics of our language (in terms of Presburger Arithmetics), showing that it covers *eventually periodic sets* [7].

## 2. The properties and the lattice

As in many approaches in the literature, we chose to focus only on the *temporal* properties of periodic events, disregarding others (e.g., agent, location). We represent it via *order-n collections* of time intervals (see the definition below and notice that, differently from Leban et al.[10], we adopt multisets).

We adopt *discrete*, *linearly ordered* and *unbounded* time; *time points* are the basic temporal primitives. The basic structures on time points are *time intervals*, i.e. non-empty sets of time points. A *convex* time interval is the set of all points between two endpoints; a *gap* interval is a non-convex set of time points. Time intervals may be *left* and/or *right-infinite*.

For uniformity of exposition we sometimes refer to intervals as *order-0* collections.

**Definition 1 (Order-$n$ collections)** *For each $n > 0$, an* order-$n$ collection *is a multiset of order-$(n-1)$ collections.*

Intuitively, Order-$n$ collections, with $n > 1$, allow one to *structure* temporal data by grouping them. For instance, "days gouped-by months grouped-by years" can be modeled (e.g., in the approach by Leban et al. [10]) by an Order-3 collection.
*Remark* (Homogeneity) Notice that in order-$n$ collections, braces are not annotated, i.e. no semantics can be attached to them. Therefore operators can act on order-$n$ collections with $n > 1$, only if they disregard any grouping. This implies that grouping can be viewed as added last, which is in tune with the intuition that the user first collects raw data and then organizes it according to some temporal scheme.

We anticipate that these considerations influence our choices for the definitions of the properties according to which we classify periodicities, and for the kinds of periodicities we choose to treat.

*Periodicities* are order-$n$ collections, for $n > 0$. A *periodic* event is an order-$n$ collection that can be defined giving a non-empty repeating pattern RP, and a

1 For instance, in Leban et al.'s approach [10], the *"first Monday of each month"* can be easily defined on the basis of *"Monday"* and *"month"*, by grouping Mondays into months (through the *Dicing* operator) and by selecting (through the *Slicing* operator) the first Monday in each group. This definition can yield, counterintuitively, the collection of one-element collections each one containing a Monday (instead of the "plain" collection of Mondays).

2 For instance, in the approach [10] of Leban et al, one can obtain only limited forms of *overlapping* intervals: one can define the collection of weeks grouped by months in which each week spanning over two months is repeated in each group (so that identical weeks overlap in the overall collection), but not, e.g., Example 2 in Section 1.

positive period specifying the spacing of different occurrences of RP. An *eventually periodic* event is (using the terminology of [7, 14]) a generalization of a periodic one, in which we admit the existence of a finite aperiodic part, and of left (or right) bounds.

We define the *extent* of an order-$n$ collection ($n \geq 0$) as the interval whose endpoints are the minimum and maximum point belonging to the collection [2]. We single out the following properties on periodicities:

G *Gaps:* the periodicity $P$ has *Gaps* if it contains gap intervals;

EP *Eventually Periodic: $P$* is *Eventually Periodic* if it can't be expressed giving a repeating pattern and a positive period (this definition is meant to capture the difference between eventually periodic events and a purely periodic one);

S *Structure: $P$* has *Structure* if it is an *order-$n$ collection*, with $n > 1$;

O *Overlaps: $P$* has *Overlaps* if the extents of some of the intervals nested in the same order-1 subcollection have non-empty intersection (*repetitions* are the special case of *exact* overlaps);

NA *Non Adjacent Intervals:* a periodicity $P$ has *Non Adjacent Intervals* if its interval closure (we call *interval closure* of $P$ the periodicity whose intervals are the extents of intervals in $P$) can't be split in a finite number of periodicities, each one covering the whole extent of $P$.

(If there are no overlaps, $NA$ captures the intuition that the ending point of each interval is immediately followed by the starting point of another one. The definition we give extends such an intuition to the general case (with overlaps), with the aim of keeping properties orthogonal.)

The five properties are orthogonal with each other.

We don't consider overlaps of collections of order-$n$, with $n > 0$; although syntactically possible they have no clear meaning, as follows from the Remark above. Based on similar arguments, the most complex periodicities we consider are basic eventually periodic events, with structure added later on.

**Definition 2 (Basic eventually periodic events)** *A (non degenerate)* basic eventually periodic event *is an order*-1 *collection that can be split in a finite portion and one or two (left/right bounded) periodic parts.*

(Without loss of generality, the three parts can be viewed as non overlapping.)

A more liberal definition of eventually periodic periodicities as order-$n$ collections, for arbitrary $n$, would include non-homogeneous objects, in which the braces in the three (finite and purely periodic) parts might have a different semantics.

Thus, we can classify the different (user-defined) periodicities according to the five properties above.

Following Leban et al. [10], we call *calendars* the periodicities that are defined as a periodic partition of the timeline, such as "minutes" or "months". Notice that calendars don't have any of the above properties. We call $Cal$ the class of calendars and build from it richer classes based on the properties above.

**Definition 3 (Classes of periodicity)** *Let $Cal$ denote the class of calendars, together with the null event. $Cal^{\pi_1,\dots,\pi_k}$ is the class of calendars together with all periodicities that may satisfy only properties in $\{\pi_1, \dots, \pi_k\}$.*

Classes of periodicities can be ordered with respect to set inclusion:

**Claim 1 (Lattice of classes of periodicity)** *The set of classes of periodicities partially ordered with respect to set inclusion, is a lattice.*

The bottom of the lattice is the set of calendars $Cal$, and the top denotes the broadest class of periodicities that we consider, i.e., $Cal^{G,EP,S,O,NA}$.

Therefore, if a periodicity belongs to a class $Cal^{\pi_1,\dots,\pi_k}$, it also belongs to all classes $Cal^{\pi_1,\dots,\pi_{k+i}}$ in the lattice, that contain the former. However, in the rest of the paper, unless explicitly stated, we stress membership to the *smallest class* containing the periodicity.

For instance, a periodicity $P$ that only has the property $NA$ (like our Example 1, of Sect. 1—see below) is contained in all classes $Cal^{NA,\pi_1,\dots,\pi_k}$, for any choice of $\{\pi_1, \dots, \pi_k\}$, including the empty set. The smallest class containing $P$ is $cal^{NA}$.

To exemplify the concept of classes, we consider again the examples of periodicities we proposed in the introductory section, and show the smallest classes containing each one of the periodicities:

**Examples.**

1. "WorkingHours" has non adjacent intervals, but no gap intervals, is not bounded, has no aperiodic part, no overlaps and no structure; therefore, the smallest class that contains it is $Cal^{NA}$.
2. Analogously, the smallest class containing "Mary's and Tom's working periods" is $Cal^{NA,G,O}$;
3. The smallest class containing "*January 2, 2003, from 13 to 14* plus *Mary's and Tom's working periods*, grouped by months" is $Cal^{G,EP,S,O,NA}$.

Our lattice is useful to classify in a uniform way the languages that appear in the literature, by associating to each language the smallest class in the lattice which contains all the periodicities that can be defined in the language.

## 3. Classification

We review four of the main symbolic approaches to user-defined periodicities from the literature and we classify them according to our lattice.

### 3.1. Collection formalism

Leban et al. [10] introduced a high-level symbolic language to define *order-n* collections. *Calendars* are used as the basis of the construction.

Two classes of operators, *dicing* and *slicing*, are defined in order to operate on collections. The *dicing* operators provide means to further divide each interval in a collection within another collection. For example, $Days : during : Months$ breaks up months into days.

*Slicing* operators provide a way of selecting intervals from collections. For instance, $[1]\backslash Days : during : Months$ selects the first day in each month.

Collection expressions can be arbitrarily built by using a combination of these operators.

**Claim 2 (Collections in the lattice)** *$Cal^{S,O,NA}$ is the minimal class in the lattice covering all periodicities definable in [10] (but not all forms of overlap are definable in Leban et al.'s formalism).*

**Proof (sketch)** No gaps are possible, and eventually periodic collections with a non empty aperiodic portion *cannot* be defined. Structure with arbitrary nesting can be defined, via the *dicing* operators. Moreover, also *f/C slicing* operators can affect structure. Periodicities with overlaps can be defined to a *limited extent*, via the *dicing* operators. For instance, repetitions are not definable. *Slicing* operators can be used in order to select non adjacent intervals. □

### 3.2. Slice formalism

Also in the approach proposed by Niezette and Stevenne [11] the basic units are *calendars*. From calendars, periodicities can be built by defining (via an ordered list of *selector.calendar* pairs) the starting point of *slices*, and (via a separate *number.calendar* pair) their uniform duration. For instance, $all.Months + 1.Days \triangleright 3.Days$ defines the *order-1* collection of three-day-long time intervals at the beginning of each month. Finally, although not explicitly stated, we understand that (set-theoretic) *union* is defined on the slices.

**Claim 3 (Slices in the lattice)** *$Cal^{O,NA,EP}$ is the minimal class in the lattice covering all periodicities definable in [11] (but not all forms of overlaps are definable in Niezette et al.'s formalism).*

**Proof (sketch)** Slices have no gaps. It is possible to define eventually periodic structures (if union belongs to the language). No structure is definable. Overlaps can be introduced by the union operator. Limited forms of overlaps can also be generated by the other slice constructors. Notice, however, that no repetitions are possible. Periodicities with non adjacent intervals can be defined, by selecting only part of the starting points (i.e., with selectors different from *all*) and/or using a short-enough duration. □

### 3.3. Gap-collection formalism

Bettini et al. [3] extended [10]'s approach to deal also with non-convex (gap) intervals, modifying the primitive for defining calendars in order to indicate the units (basic granules) constituting each non-convex interval. For example, the collection consisting of a Monday, a Wednesday and a Saturday, treated as a single gap time interval, is defined as $Generate(5, Days, 7, \langle[1, 1], [3, 3], [6, 6]\rangle)$. Moreover, [3] re-defined dicing operators in order to consider non-convex intervals.

**Claim 4 (Gap-collections in the lattice)** *$Cal^{G,O,NA,S}$ is the minimal class in the lattice covering all periodicities definable in [3] (but not all forms of overlap are definable in Bettini et al.'s formalism).*

**Proof (sketch)** The same classification we proposed for [10] is still valid, except that the class is enriched of all periodicities that contain gap intervals. □

### 3.4. Unstructured-collection formalism

Terenziani [14] proposed a language which is basically a revision of [10]'s one in order to generate only *order-1* collections (this makes the language more suitable for representing the *validity times* of tuples in temporal relational databases). In Terenziani's approach, periodicities are generated starting from *calendars*, and applying the *Select-Periods* operator, which is is basically a combination of Leban's *slicing* and *dicing* operators. For instance, *Select-Periods(2,Days,during,Weeks)* denotes the collection of Mondays (American calendar interpretation). Moreover, also the set-theoretic operators of *union, intersection, difference* and *complement* are introduced, as well as an explicit operator to introduce (upper and/or lower) *bounds.*

**Claim 5 (Unstruct'd-collections in the lattice)** *The minimal class in the lattice covering all periodicities definable in [14] is $Cal^{O,NA,EP}$ (but not all forms of overlap are definable in Terenziani's formalism).*

**Proof (sketch)** Intervals have no gaps. It is possible to define eventually periodic structures (using union and bounds). No structure is definable. Overlaps can be introduced by the union operator. Notice, however, that no repetitions are possible. Periodicities with non adjacent intervals can be defined, by using the *Select-Periods* operator. □

## 4. Top language

As a result of the study above, we propose a new symbolic language which is expressive, clean and intuitive in the following sense:

- it can define all periodicities in the top class of our lattice;
- operators are orthogonal: two of them define the bottom class, each one of the rest introduces one property and doesn't affect any other;
- operators either specifically modify intervals (*CalDef* and *Drop*), or treat them as atomic entities (yielding a clear operational semantics with no side-effects);
- operators are designed to act in an intuitive way on structures.

But no syntactical sugar is used. Thus our language is not necessarily practical and should be considered mainly as a proof of concept.

We use the following two relational operators:

**Definition 4 ( SDur, NSDur)** *Let I (resp. I') be the extent of interval J (resp. J'). J is* strictly during *J' (J* **SDur** *J') if I' contains I and the endpoints of I are both different from the endpoints of I'; J is* non strictly during *J' (J* **NSDur** *J') if I' contains I.*

In the following, we occasionally assume an ordering on intervals. We use the lexicographic order on interval endpoints.

The basic operators for *Cal* are the classical ones. We briefly comment all the others as we introduce them.

- **Operators defining *Cal***
*Chronon_Calendar:*
*No input;*
*Output:* an order-1 collection right and left infinite, containing adjacent one-element no-gap intervals.
*CalDef:*
*Input:* a calendar $C$ in *Cal*, an 'anchor' time point $p$ which is the start point of an interval in $C$, and a list of positive natural numbers $n_1, \ldots n_k$;
*Output:* a calendar, built by grouping cyclically $n_1, \ldots n_k$ intervals of $C$ starting from $p$ (both directions), i.e. the union over $I\!N$ of the intervals $I_h$ defined as:

$$\bigcup_{\substack{j=1,\ldots,k \\ s \in I\!N}} \left\{ I_s \middle| \; l(h) + \sum_{i=1}^{j-1} n_i < s \leq l(h) + \sum_{i=1}^{j} n_i \right\},$$

where $\{I_s\}$ is the set of intervals in $C$, ordered according to the natural ordering, $l(h) = ap + h \cdot \sum_{i=1}^{k} n_i$ and $ap$ is the index of the interval starting at the anchoring point.
**Example.** Here and in the following examples, we assume $Hours = Chronon\_Calendar$, that the hour 0 is 00:00 of Jan 1st, 2003, and that the definition of "Days" has been given. The calendar "Work&RestHours" splits each day into three parts: morning rest hours (say, the first eight hours of each day: 0-8), work hours (the following ten hours: 8-18) and the evening rest hours (the remaining six hours: 18-24).

$$\text{Work\&RestHours} = CalDef(\text{Hours}, 0, (8, 10, 6))$$

- **Select (for non adjacency)**
*Input:* $N = \{n_1, ...., n_k\}$, $n_i \in I\!N$, order-1 collections $C_1$ and $C_2$;
*Output:* An order-1 collection (or the empty set):

$$\bigcup_{n_i \in N, j \in C_2} \{n^{th}(\{i \in C_1 | i \text{ NSDur } j\}, n_i)\}$$

where $n^{th}(C, i)$ selects the $i$-th interval in the collection $C$, if it exists, it is the empty set otherwise.
*Description.* Select$(N, C_1, C_2)$ modifies collection $C_1$ based on $C_2$ and $N = \{n_1, \ldots, n_k\}$. For each interval $j \in C_2$, consider the ordered set of intervals of $C_1$ contained in $j$: $\{i \in C_1 | i \text{ NSDur } j\}$. The elements of positions $n_1, \ldots, n_k$ in this set

are selected and included in the output periodicity $C_{new}$. Notice that $C_{new}$ is a (not necessarily proper) subset of $C_1$.

Select can only introduce NA since it only drops intervals.

**Example.** "WorkingHours"(WH), as in Example 1 in the Introduction, can be defined as follows (on the basis of the definition of "Work&RestHours"):

$$\text{WH} = \mathit{Select}(2, \text{Work\&RestHours}, \text{Days})$$

- **Drop (for gaps)**
*Input:* $N = \{n_1, ...., n_k\}$, $n_i \in I\!N$, order-1 collections $C_1$ and $C_2$;
*Output:* An order-1 collection:

$$\bigcup_{i \in C_1} \left\{ i \setminus \bigcup_{n_i \in N, j \in C_2} n^{th}(\{j \in C_2 | j \text{ SDur } i\}, n_i) \right\}$$

where $\setminus$ is set minus, and $n^{th}(C, i)$ selects the $i$-th interval in the collection $C$, if it exists, it is the empty set otherwise.

*Description.* Drop is very similar to Select, except that it modifies intervals in the periodicity. Drop$(N, C_1, C_2)$ modifies intervals in $C_1$ based on $C_2$ and $N = \{n_1, \ldots, n_k\}$. Consider, for each interval $i \in C_1$, the ordered set $\{j \in C_2 | j \text{ SDur } i\}$ of all intervals of $C_2$ that are strictly during $i$. The points contained in the $n_1$-th, $n_2$-th,..., $n_k$-th of these intervals are dropped from $i$, thus generating gaps. The new periodicity output consists of all intervals of $C_1$ with the holes (potentially) introduced by the procedure described.

Since Drop defines new intervals dropping portions of input ones, the use of SDur guarantees that NA can't be added.

**Example.** "DayShift", from 8 to 12 and from 14 to 18, intended as unique time intervals with a gap in it (as in Example 2 in the Introduction), can be defined as follows:

$$\text{DayShift} = \mathit{Drop}(4, 5, \text{WH}, \text{Hours}).$$

Notice that since Drop is defined using SDur (and not NSDur), the first hour in the set $\{j \in \text{Hours} \mid j \text{ SDur } i\}$, for $i \in \text{WH}$, is the hour from 9 to 10.

- **Union (for overlaps)**
*Input:* two order-1 collections $C_1$ and $C_2$;
*Output:* the order-1 collection

$$\bigcup_{i \in C_1} \{i\} \cup \bigcup_{j \in C_2} \{j\},$$

where $\cup$ and $\bigcup$ denote *multiset union.*

*Description.* Union is a multiset union.

It is defined only over order-1 collections, consistently with the Remark on homogeneity.

**Example.** "Mary's and Tom's shifts" ("MTShifts" for short; see Example 2) can be defined as follows:

$$\text{MTShifts} = \mathit{Union}(\text{DayShift}, \text{DayShift})$$

- **Replace (for eventual periodicity)**
*Input:* Order-1 collections $C_1, C_2$ and $C_3$ (where $C_1$ and $C_3$ can be the empty set, but not $C_2$) and time points $p_1, p_2$ such that $p_1 < p_2$ and if $p_1$ belongs to some intervals of $C_1$ or $C_2$, it is their right endpoint, and similarly for $p_2$ relative to $C_2$ and $C_3$;
*Output:* An order-1 collection:

$$\{i \in C_1 | i \text{ NSDur}(-\infty, p_1]\} \bigcup \bigcup \{i \in C_2 | i \text{ NSDur}[p_1 + 1, p_2]\} \bigcup \bigcup \{i \in C_3 | i \text{ NSDur}[p_2 + 1, \infty)\}.$$

*Description.* This operator corresponds to the intuition that a complex periodicity consists of bounded portions of basic periodicities. Replace$(C_1, p_1, C_2, p_2, C_3)$ builds a periodicity $C_{new}$ consisting of three non-overlapping portions. The first portion, up to and including $p_1$, is identical to $C_1$; the second, starting on the point $p_1 + 1$ and up to $p_2$, included, is identical to $C_2$, and the remaining one, starting from $p_2 + 1$ is identical to $C_3$.

If $C_1$ (resp. $C_3$) is empty, $C_{new}$ is left (resp. right) bounded. $C_2$ can't be empty, otherwise Replace might introduce $NA$ as well as $EP$. The constraints on $p_i$ have the same purpose and also ensure that intervals need not be split—the only operators that modify intervals are CalDef and Drop. The arguments must be order-1 collections in order to avoid building non-homogeneous structures.

**Example.** "Mary and Tom's working plan" (MTPlan) meaning "*January 2, 2003, from 13 to 14*, plus *MTShifts*" can be defined as follows:

$$\text{MTPlan} = \mathit{Repl}(\text{MTShifts}, 37, \text{EH}, 38, \text{MTShifts}),$$

where EH $= \mathit{Select}(14, \text{Hours}, \text{Days})$ is the periodicity consisting of the hour 14-15, every day.

- **Group-by (for structure)**
*Input:* An order-$n$ collection $C_1$ and a calendar $C_2$,

such that for each interval $i$ of $C_1$, there is an interval $j \in C_2$ containing it: $i$ NSDur $j$;
*Output:* An order-$(n+1)$ collection:

$$\bigcup_{j \in C_2} \left\{ \bigcup_{i \in C_1} \{ i \in C_1 | i \text{ NSDur } j \}_{C_1} \right\},$$

where the subscript $C_1$ indicates that the structure of $C_1$ is preserved.

*Description.* Group-by$(C_1, C_2)$ groups $C_1$'s intervals according to the intervals in $C_2$

The constraints on $C_1$ are motivated by homogeneity requirements (see the remark after Definition 1). The constraint relating intervals of $C_1$ with intervals of $C_2$ ensures that each interval enters either a whole or not at all in a group (and it enters in no more than one).

If, for some $k$, $C_2$ refines the structure defined by order-$k$ subcollections of $C_1$, and in turn order-$(k-1)$ subcollections refine $C_2$, then the interpretation of the requirement "the structure of $C_1$ is preserved" is intuitive, and we don't state it precisely. If order-$k$ subcollections of $C_1$ (for $k > 0$) are not a refinement of $C_2$ nor the other way around, the requirement that "the structure of $C_1$ is preserved" can be managed in two different ways: see [10] and [11].

**Example.** The periodicity in Example 3 in the introduction can be defined by applying the Group_by operator to "MTPlan" (given the definition of Months):

$$\textit{Group\_by}(\text{MTPlan}, \text{Months}).$$

We briefly notice that the language we propose has the claimed expressiveness:

**Theorem 1** *The language proposed above is sufficient in order to define all periodicities in the top class of the lattice.*

**Proof (sketch)** The operators for *Cal* are classical and well understood. Calendars can be appropriately modified using Drop and Select in order to obtain any sorts of gap intervals and of non adjacency of intervals. Union provides all the overlaps that we chose to admit. An application of Replace yields the most general periodicity structure available. Finally structure can be added to it, using Group-by. □

## 5. Conclusions

In this paper, we focus on symbolic languages for user-defined periodicity, and propose a unifying approach. We introduce a lattice of classes of temporal properties regarding periodicity, and use it in order to classify different symbolic approaches in the literature, thus providing an homogeneous framework to compare them. Moreover, we also propose a symbolic language which covers the top of the lattice (i.e., all the properties).

Of course, the approaches we considered in this paper are not the only symbolic approaches in the literature. For instance, Cukierman and Delgrande [5] introduced a very expressive high-level language to model periodicities, considering also intersection, union, difference and universal and existential quantification.

Set operators were also provided by Ning et al [12], who also proposed a set of algebraic operators to combine and select granules of a granularity to form new granularities.

Moreover, within the AI community, starting from Ladkin's pioneering work [9], a lot of attention has been devoted to the treatment of qualitative temporal constraints (e.g., "before") between events which repeat in time.

Tuzhilin and Clifford [15] proposed an overview of many approaches to user-defined periodicity, including symbolic, logical and constraint-based ones. However, the focus of their paper is on a formal logical-based definition of *strongly periodic* and *nearly periodic* events and of a *query* language coping with them, and the corresponding extension of temporal relational DB query languages.

Alternatively, Bettini and De Sibi [3] proposed a mathematical characterization of granularity, and used it in order to compare Leban et al.'s and Niezette et al.'s formalisms, considering the properties of admitting *bounds* (which is implied by our EP property) and/or *gaps*. The property NA was not taken explicitly into account, since it is common to both approaches. On the other hand, O, EP, and S were not considered since they are ruled out by their reference mathematical framework. Notice that, in [3], Bettini and De Sibi also introduced the symbolic language we discussed in Section 3.

To conclude, we deliberately tailored our language so that each operator corresponds to a property. This enhances clarity from the theoretical point of view, and gives a solid basis for the design of semantically clear languages. However, in order to enhance the user-friendlyness of such languages, we are currently investigating the possibility of adding some "syntactic sugar", and/or decomposing some of our operators (e.g., "CalDef" might be further decomposed into different operators for "combining" periodicities, along the lines suggested by Ning et al [12]). Moreover, as future work, we also envision the adoption of the language we introduced in this paper in order to deal with symbolic user-defined periodicities in relational temporal databases (at the level of both *data* and *query* language [14]).

## References

[1] M. Baudinet, J. Chomicki, P. Wolper, Temporal Databases: Beyond Finite Extensions, *Proc. Int'l Workshop on an Infrastructure for Temporal Databases*, 1993.

[2] C. Bettini, C. Dyreson, W. Evans, R. Snodgrass, X. Wang, A Glossary of Time Granularity Concepts, in *Temporal Databases: Research and Practice*, Springer Verlag, 1998.

[3] C. Bettini, R. De Sibi, Symbolic Representation of User-defined Time Granularities, *Proc. TIME'99*, IEEE Computer Society, 17-28, 1999.

[4] J. Chomicki, T. Imielinsky, Finite Representation of Infinite Query Answers, *ACM ToDS* 18(2), 181-223, 1993.

[5] D. Cukierman, J. Delgrande, Expressing Time Intervals and Repetition within a Formalization of Calendars, *Computational Intelligence* 14(4), 563-597, 1998.

[6] L. Egidi, P. Terenziani, A mathematical framework for the semantics of symbolic languages representing periodic time, *Proc. TIME'04*, IEEE Computer Society, 2004.

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

[8] F. Kabanza, J.-M. Stevenne, P. Wolper, Handling Infinite Temporal Data, *Journal of Computer and System Sciences* 51, 3-17, 1995.

[9] P. Ladkin, Time Representation: A Taxonomy of Interval Relations, *AAAI'86*, 360-366, 1986.

[10] B. Leban, D.D. McDonald, D.R. Forster, A representation for collections of temporal intervals, *AAAI'86*, 367-371, 1986.

[11] M. Niezette, J.-M. Stevenne, An Efficient Symbolic Representation of Periodic Time, *Proc. first Int'l Conf. Information and Knowledge Management*, 1992.

[12] P. Ning, X.S. Wang, S. Jajodia, An Algebraic Representation of Calendars, in *Annals of Mathematics and Artificial Intelligence* 36(1-2), 5-38, 2002.

[13] M. Soo, R. Snodgrass, Multiple Calendar Support for Conventional Database Management Systems, *Proc. Int'l Workshop on an Infrastructure for Temporal Databases*, 1993.

[14] P. Terenziani, Symbolic User-defined Periodicity in Temporal Relational Databases, *IEEE TKDE* 15(2), 489-509, 2003.

[15] A. Tuzhilin J. Clifford, On Periodicity in Temporal Databases, *Information Systems* 20(8), 619-639, 1995.