An Efficient Symbolic Representation of Periodic Time

Marc Niézette Jean-Marc Stévenne
Université de Liège *

Abstract

We describe a system for storing and processing complex temporal information. It consists in a symbolic layer added above a strong mathematical model known as *"generalized databases"*. This symbolic layer is based on the notions of *calendars* and *slices*. Calendars such as years, months, weeks,... represent everyday's life time structure as well as any more specific structure such as school terms,.... Slices combine calendars to express complex temporal information such as "every Monday of the second week of each month". They offer both the advantage of conviviality with a dramatic reduction in space and computational cost with respect to the purely mathematical formalism.

1 Introduction

For many years, storing and handling temporal information has been a topic of interest in database technology. Since the first models were proposed [Bub77] and a clear semantic of the so called *historical databases* was given [CW83], a number of models for temporal databases have been presented [Ari86, Tan86, NA89, Gad88, TC90, GM91]. Usually, these models extend the well-known relational model in order to include temporal aspects, and give extensions of the relational query languages that handle the temporal dimension of the data. Unfortunately, most of these models do not take into account the infinite nature of time. This means that there is a limit to the temporal extension of predicates represented by the relations in those databases. For instance, one cannot store information that is repeated every month unless knowing (or arbitrarily fixing) upper and lower bounds to the repetition. Even if those limits are known, this representation can involve a large amount of tuples (one for each occurrence).

More recently, a few models have been proposed, that allow finite representation of infinite temporal extension. In [CI88, CI89, CI90, Cho90], the infinite dimension of time is represented using a deductive layer upon an finite historical database. In [KSW90], another extension of the relational model allowing the finite representation of infinite temporal extension is introduced.

*address: Institut Montéfiore, B28; 4000 Liège Sart-Tilman; Belgium.
Email: {niezette,stevenne}@montefiore.ulg.ac.be

The next subsection briefly describes another formalism called *generalized databases*, presented in [KSW90]. We illustrate its expressive power and review some complexity results for this purely mathematical formalism. The expressive power of these various representations is discussed in [BNW91].

In section 1.2 we illustrate some of its drawbacks which are mainly due to the absence of symbolic informations about the structure of the calendar. This advocates a symbolic representation like the one described in [LMF86] which is very close to natural language but for which no general method of evaluation was provided.

We suggest the use of a symbolic layer built upon generalized databases. We define this system in section 2. Finally, in section 3 we give an evaluation method which takes advantage of both the symbolic level and the mathematical base. Indeed the symbolic layer is not only important from the conviviality point of view but it also dramatically reduces both the storage and computational cost of the system.

1.1 Generalized temporal databases

Generalized databases are extensions of the classical relational databases which allow the finite representation of infinite temporal information by generalizing the notion of tuple. A *generalized tuple* represents a possibly infinite set of classical tuples. This is achieved by introducing mathematical expressions called *linear repeating points (lrps)* as values of temporal attribute(s). A linear repeating point is a set of integers represented by a linear equation $c + kn$ where c and k are integer constants and n takes all values from $-\infty$ to $+\infty$ in Z. In addition, conjunction of constraints on the temporal attributes can be associated with each tuple. These constraints are inequalities of the form $X_i \leq c$ or $X_i \geq c$ or $X_i \leq X_j + c$ where X_i and X_j represent attributes values and c is an integer constant.

All relational algebra operations have been extended to support lrps. For *"normalized generalized databases"* all operations can be computed in PTIME under *"fixed-schema complexity"* (that is, for a given schema of the database all operations can be computed in PTIME with respect to the number of tuples). Furthermore, under *"general complexity"* (i.e. when both the number of tuples and the schema of the database can vary) all operations except complementation can be computed in PTIME [KSW91]. *Normalization* consists in replacing each temporal tuple into a set of tuples in which the periods of all temporal attributes are identical. The normalization process can dramatically increase the number of tuples if the periods are not closely related. In practice however this should not be the case. Moreover, normalization is only required before projections and only between attributes sharing a common constraint with the attribute being projected out. But, as we will see in section 1.2, even this PTIME complexity of operations does not mean efficiency.

The following example illustrates the power of representation of this formalism and motivates the use of relations with two temporal attributes for representing intervals.

Example 1.1 The following generalized relation represents the schedule of trains leaving Liège every hour, h, at $h : 02$ and arriving in Brussels at $(h + 1) : 20$ and others trains leaving Liège at $h : 46$ and arriving at $(h + 1) : 50$.

Train		
$02 + \texttt{one\_hour} * n_1$	$80 + \texttt{one\_hour} * n_2$	$X_1 = X_2 - 78$
$46 + \texttt{one\_hour} * n_1$	$110 + \texttt{one\_hour} * n_2$	$X_1 = X_2 - 64$

In the constraints part, X_1 and X_2 refer respectively to the first and second temporal argument. ∎

Notice that by using two temporal attributes the fact that during some intervals there are two trains on the same line entails no confusion. If we were restricted to only one temporal attribute per relation we would use two relations one with the departure times and another with arriving times. But then we would have to use a special trick to correctly pair each departure with the corresponding arrival. Actually, in the above relation this kind of synchronization is performed by the constraint part of each tuple. It seems that such "synchronization constraints" will always occur in practice and thus that a representation in terms of linear repeating (fixed length) intervals (*lris*) instead *lrps* would be more accurate. In [Sté91], an *lris* based representation is advocated instead of an *lrps* one. However, in order to be able to represent the extension of a certain class of temporal predicates, a distinction is introduced between *genuine intervals* which are interpreted as pairs of points and *liquid intervals* interpreted as the set of their included points. This allows representing both predicates like "train" (whose extension is given in the above example), which are classically called *"actions"* in the AI literature[All84, Sho86] and predicates like "the train is in the station", typically called *"facts"*. Actions *occur over genuine intervals* (but not over their subintervals) while *facts hold on liquid intervals* (and all their sub-intervals). Of course different algorithms must be used to compute relational algebra operations for each type of intervals. [Sté91] shows that this representation leads to much more compact relations and faster evaluation.

1.2 Limitations

Using linear repeating intervals (points) for representing time gives us a great expressiveness, but is not convenient for everyday's life timing information.

Example 1.1 is rather simple since it assumes that each hour the same train schedule is repeated. It can be refined by specifying that, actually, this repetition is valid from 6 am to 11 am. In an lri based formalism, this would yield a relation like :

Train	
$n * \texttt{one\_hour} + (02, 80)$	$X.b \geq 6 * \texttt{one\_hour} \wedge X.b \leq 12 * \texttt{one\_hour}$
$n * \texttt{one\_hour} + (46, 110)$	$X.b \geq 6 * \texttt{one\_hour} \wedge X.b \leq 12 * \texttt{one\_hour}$

where $X.b$ represents the beginning points of the repeated intervals.

This relation represents only the situation for one day (say one Monday), but this schedule is actually repeated every day from Monday to Friday every week. Unfortunately, such embedded repetition cannot be handled simply and leads to duplication of tuples with a period of one week :

Train
$n*\texttt{one\_week} + (6 * \texttt{one\_hour} + 02, 6 * \texttt{one\_hour} + 80)$
$n*\texttt{one\_week} + (7 * \texttt{one\_hour} + 02, 7 * \texttt{one\_hour} + 80)$
$\vdots$
$n*\texttt{one\_week} + (11 * \texttt{one\_hour} + 02, 11 * \texttt{one\_hour} + 80)$
$n*\texttt{one\_week} + (1 * \texttt{one\_day} + 6 * \texttt{one\_hour} + 02, 1 * \texttt{one\_day} + 6 * \texttt{one\_hour} + 80)$
$\vdots$
$n*\texttt{one\_week} + (1 * \texttt{one\_day} + 11 * \texttt{one\_hour} + 02, 1 * \texttt{one\_day} + 11 * \texttt{one\_hour} + 80)$
$\vdots$
$n*\texttt{one\_week} + (4 * \texttt{one\_day} + 11 * \texttt{one\_hour} + 02, 4 * \texttt{one\_day} + 11 * \texttt{one\_hour} + 80)$

This last relation contains 60 tuples and represents what is said in two sentences. Moreover, the information relative to the calendar has been lost. For instance, in a query involving the scheduling of train on Saturdays, there is no means to detect that all of the above tuples are out of temporal range and thus all will be involved in join operations although the result will be inevitably empty. With a symbolic representation, knowing that Saturday are disjoined from Mondays, ..., Fridays, we should directly conclude that the result is empty without an expensive computation.

2 Definition of the system

2.1 Calendars

In the previous section we have illustrated the advantages and drawbacks of the generalized database formalism. This suggests the use of a symbolic formalism. But a pure symbolic approach like the one proposed in [LMF86] in not completely satisfying since it seems hard to obtain a complete evaluation procedure. Instead of redefining a whole symbolic system, we propose to build this symbolic layer over a lris (lrps) layer. This would provide us the mathematical basis of lris (lrps), combined with a human-readable formalism, allowing symbolic (and, if needed, algebraic) computation.

Our system relies on the notion of *calendar*.

Definition 2.0.1 *A* calendar *is defined as a* periodic infinite set of consecutive intervals which partition time. *These intervals are referred to by a common name, called the* granularity *of the calendar.*

Example 2.1 "Every week", "every day", "every two days" are calendars. "Every months" is also a calendar, even if the relative duration of a month in terms of days can vary. ∎

In the following, the calendars "every week", "every day",... will be represented by `weeks`, `days`,...,using a `typewriter font`.

Definition 2.0.2 *A calendar* X *is* subcalendar *of a calendar* Y, *written*

$$\mathrm{X} \sqsubseteq \mathrm{Y}$$

if each interval of Y *is exactly covered by an integer number of intervals of* X*. The relation* subcalendar *defines a partial order on the calendars set.*

Example 2.2 `Days` is a subcalendar of `months`, but `weeks` is not a subcalendar of `months`. ∎

The main calendars appearing in everyday's life are `seconds`, `minutes`, `hours`, `days`, `weeks`, `months`, `years`. Each of them can be represented by one or more lris, depending on the chosen tick. `Years`, for instance, which stands for "every year" is a lri if the tick is the *year* (the *duration year*) or the *month*.

$$\begin{array}{rl} n+[1,1] & \text{if the tick is the } \textit{year} \\ 12.n+[1,12] & \text{if the tick is the } \textit{month} \end{array}$$

If the tick is the *day*, `years` is no longer equivalent to a lri, because its duration in terms of days is not a constant. It is a set of lris whose period is four years :

$$\begin{array}{lrcl} & 1461.n & + & [1,\ 366] \\ \cup & 1461.n & + & [367,\ 731] \\ \cup & 1461.n & + & [732,\ 1096] \\ \cup & 1461.n & + & [1097,\ 1461]. \end{array}$$

The only calendars that are lris in terms of all their traditional subunits, are `seconds`, `minutes`, `hours`, `weeks`. The other ones, `months` and `years`, will be sets of lris. The dependencies between the traditional calendars are described by the following graph, where each arrow holds for the *subcalendar* relation.

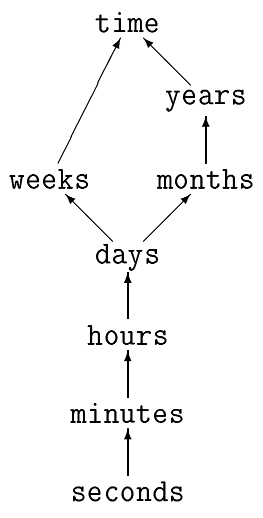


Figure 1: Usual calendar structure

The node `time` on the graph of figure 1 is a particular calendar consisting in one infinite interval including all time points. The terminal nodes of the graph, such as `seconds` in this example, will be declared as *basic*, meaning that they are ticks of the system and cannot be expressed in terms of subunits. We will usually have only one tick. Assuming the existence of `time` and a single tick `second`, the *subcalendar* relation will define a lattice on the calendars set.

The relation between the calendars is described by three main properties :

- the *duration* of a calendar versus one of its subcalendars.
- the *synchronization point* of a calendar versus one of its subcalendars.
- the *period* of a calendar versus one of its subcalendars.

Definition 2.0.3 *If C_0 and C_1 are calendars, with $C_1 \sqsubseteq C_0$,* duration(C_0/C_1) *returns the list of the successive durations of C_0 in terms of the granularity of C_1. This set of durations is periodic.*

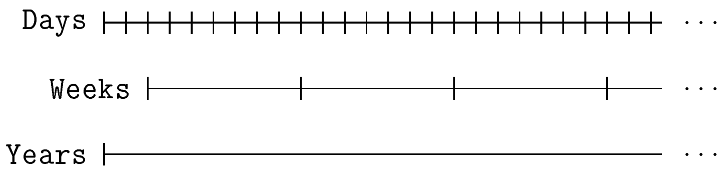
Example 2.3

$$\begin{aligned}\mathrm{duration}(\texttt{years/days}) &= [365, 365, 365, 366]\\ \mathrm{duration}(\texttt{years/months}) &= [12]\\ \mathrm{duration}(\texttt{years/years}) &= [1]\end{aligned}$$

∎

Definition 2.0.4 *The* synchronization point *of a calendar C_0 with respect to a calendar C_1 such as $C_1 \sqsubset C_0$,* synchronization(C_0/C_1)*, is the interval of C_1 which begins at the same time as the first interval of C_0.*

Example 2.4 Consider the three calendars represented in the following figure:



We have that

$$\begin{aligned}\mathrm{synchronization}(\texttt{years/days}) &= 1.\mathit{days}\\ \mathrm{synchronization}(\texttt{weeks/days}) &= 3.\mathit{days}\end{aligned}$$

∎

The *synchronization point* and the *duration* completely define a calendar.

Example 2.5 The `week` will be defined as $\{1.\mathit{day}{:}\mathit{day}{:}[7]\}$, and `years` as $\{1.\mathit{day}{:}\mathit{day}{:}[365, 365, 365, 366]\}$. ∎

We will be able to create any new calendar, and introduce it into this graph, given its exact relation to the other calendars. E.g., adding a calendar `school-terms`, which holds for "every school-term", can be done by adding a node above `months`, with associated information $\{9.months{:}months{:}[4,6,2]\}$.

Moreover, calendars should be created dynamically for computational purposes. An operation involving two calendars requires the expression of one in terms of the other. If these are not related by the relation subcalendar, then a third one must be choosen or possibly created such as the to first are are both subcalendars of this third one. It appears that the creation of a calendar whose periodicity is 28 years long, corresponding to the common periodicity of both `years` and `weeks` will certainly be bound to happen. We will use it later in examples of section 3 under the name of `bigwheels`.

Definition 2.0.5 *The period of a calendar C_0 with respect to the calendar C_1, where $C_1 \sqsubseteq C_0$ is defined as*

$$\mathrm{period(C_0/C_1)} = \sum_{i=1}^{\mathrm{length(duration(C_0/C_1))}} \mathrm{elem}(i, \mathrm{duration}(C_0/C_1)$$

*where elem(i,*list*) returns the i^e element of the* list *and length(*list*) returns the number of elements in the* list.

Example 2.6

$$\begin{aligned} \mathrm{period}(\texttt{years}, \texttt{days}) &= 1461 \\ \mathrm{period}(\texttt{years}, \texttt{months}) &= 12 \end{aligned}$$

∎

The *synchronization point*, *duration* and *period* of a calendar versus a nondirect subcalendar can be computed recursively.

2.2 Slices

Using the calendars as base for our periodic representation of time, we can now define the basic structure we are going to use, the *slice*.

Definition 2.0.6 *A slices is an infinite repeated set of time intervals. It is built according to the following recursive rules. :*

1. *Each calendar is a* slice.
2. *If* R *is a slice,* O *an integer (or a set of integers) and* G *the granularity of an existing calendar, then*

   $$\mathrm{S} \ = \ \mathrm{R} \ + \ \mathrm{O.G}$$

   is a slice, where

- R *is the slice* reference*. It give us a first infinite set of intervals, each of them containing one or more intervals of* S.
- G *is the slice* granularity *and corresponds to the duration of the intervals of* S.
- O *is the slice* offset(s) *of the intervals of* S *with respect to the beginning of each interval of* R*. It fixes the position of the intervals of* S *in each interval of* R*. It can take the special value* all *if it holds for all the intervals on* S *included in each interval of* R*.*

Actually, the temporal relation that holds between the intervals of S and the intervals of R is *during*.

Let us notice that a calendar is a slice which is its own reference and granularity, with an offset of 1.

Example 2.7 The following array illustrates the use of the slices to express temporal extensions.

Meaning	Slice representation
Sundays	$\texttt{week} + 1.days$
The 3^{rd} day of each month	$\texttt{months} + 3.days$
The 4^{th} hour of the 3^{rd} day of each month	$\texttt{months} + 3.days + 4.hours$
The 1^{st} of April 1992	$\texttt{time} + 1992.years + 4.months + 1.days$
Usually we will omit `time` *just keeping:*	$1992.years + 4.months + 1.days$

∎

Definition 2.0.7 *The ordered list of calendars whose granularity appears in the expression of a slice will be called the* hierarchy *of the slice.*

Example 2.8 The hierarchy of

$$\texttt{months} + 3.days + 4.hours$$

is

$$(\texttt{months}, \texttt{days}, \texttt{hours})$$

∎

Definition 2.0.8 *A slice is S such as*

$$(c_1, c_2, \ldots, c_n) = hierarchy(S)$$

is regular *iff*

$$\forall_{i,j\ in[1,\ldots,n]}\ i \leq j\ \supset\ c_j \sqsubseteq c_i$$

Example 2.9 "The first day of the month", represented by

$$\texttt{month} + 1.day$$

is a *regular slice*, while "the first week of the month", i.e.

$$\texttt{month} + 1.week$$

is not since `weeks` is not a sub calendar of `months`. ■

We will use *sets of slices* to represent infinite temporal data, as we represented them by *sets of lris (lprs)* in [KSW90]. Various sets of slices can correspond to a single temporal extension. Particularly, considering a slice S_0 whose hierarchy is

$$(\texttt{c}_0, \texttt{c}_1, \texttt{c}_2)$$

and if there is a calendar c'_1 such as

$$\texttt{c}_2 \sqsubset \texttt{c}_{1'} \sqsubset \texttt{c}_1$$

we can always find a set of slices representing the same extension whose common hierarchy is

$$(\texttt{c}_0, \texttt{c}_1, \texttt{c}_{1'}, \texttt{c}_2)$$

Example 2.10

$$\texttt{years} + 122.days$$

is equivalent to

$$\begin{array}{rl} & \texttt{years} + 5.months + 3.days \\ \cup & \texttt{years} + 5.months + 3.days \\ \cup & \texttt{years} + 5.months + 2.days \\ \cup & \texttt{years} + 5.months + 3.days \end{array}$$

where the computation of the offset of *days* and *months* can be achieved using the information stored in the calendars.

The reverse operation is not always possible without modifying the reference calendar. E.g.,

$$\texttt{years} + 5\ months + 3.days$$

cannot be reduced in terms of `years` and `days`. We have to introduce a new supercalendar to achieve that (as we will see in 3.1.3). ■

The reference calendar of a slice can also be replaced by a slice with one of its supercalendar as reference, the granularity of the old reference calendar as slice granularity and *all* as offset. E.g.,

$$\texttt{months} = \texttt{years} + all.months$$

Particularly, every calendar `cal` can be translated into the slice

$$\texttt{time} + all.cal$$

In the following sections, the slices will be represented under the format

$$\sum_i O_i.C_i,$$

where the O_i are offsets and the C_i granularities of calendars, that is the format

$$\texttt{time} + \sum_i O_i.C_i$$

where the `time` term has been omitted.

3 Evaluation

We can now define the way we are going to evaluate formulas involving slices. Our goals are

- to provide a mechanism that improves the performances of the classical lri evaluation, making use of symbolic simplifications or using the slice representation to optimize the classical algorithm application.
- to keep the more structured representation we can have, for computation purposes as well as for user friendliness.

Improving performances can mainly be achieved at the symbolic level by quick checking of the emptyness of the solution. We should be able to directly check that

$$\texttt{Sundays} \cap \texttt{Saturdays} \equiv \emptyset.$$

More complicated evaluation should be performed using as much as possible the structure of our representation, shifting to real lri (lrp) evaluation in difficult cases. We will restrict here our presentation to the most important operation: intersection.

3.1 Intersection

Intuitively, once two slices X and Y have the same hierarchy their intersection is simply a slice Z with the same hierarchy and whose offsets are the intersections of the corresponding offsets in X and Y.

When the hierarchy of X and Y differs, before intersecting X and Y, we have to rewrite them such as they share the same hierarchy. As we will see, such a rewriting of a X (resp. Y) can result in a union of slices X_i (resp. Y_i). Each X_i must then be intersected with each Y_i. Hence it is important to avoid rewriting as much as possible. For example by analyzing only some *"mutually synchronized"* parts of slices X and Y (that are parts whose hierarchies correspond) it is often possible to conclude that their intersection is empty or at least to reduce the number of rewritten slices.

Before giving the intersection algorithm, we shall see how a slice can be *"left-completed"* that is rewritten into slices with one more calendar at the left of its hierarchy.

3.1.1 Left completion algorithm

The *left completion* of a slice X by a calendar C_0 follows the following rules.

- $X = C_1 + \sum_{i=2}^{n} O_i.C_i$ can be rewritten into $X = C_0 + C_1 + \sum_{i=2}^{n} O_i.C_i$
- $X = \sum_{i=1}^{n} O_i C_i$, where $O_1 \neq all$ can be left-completed by a calendar C_0 such as $C_1 \sqsubset C_0$ and duration$(C_0/C_1) =$ period(C_0/C_1). This results into a set of slices X_i given by the following algorithm :
  1. Let k_1 be the cardinal of the set of offsets O_1 and o_{1i} be the elements of O_1. Let
  $$S = \bigcup_{i=1}^{k_1} (\lfloor o_{1i} \div \text{period}(C_0/C_1) \rfloor + 1, (o_{1i} \bmod \text{period}(C_0/C_1)) + 1)$$
  2. For each s_{2i} such as there is a pair $(s_{1j}, s_{2i}) \in S$, let
  $$O_{0i} = \bigcup_{j} \{s_{1j} | (s_{1j}, s_{2i}) \in S\}$$
  3. Then,
  $$X = \bigcup_{i} X_i = O_{0i}.C_0 + s_{2i}.C_1 + \sum_{j=2}^{n} O_j.C_j$$

3.1.2 Intersection of two regular slices

Let

$$X = O_{X1}.C_{X1} + O_{X2}.C_{X2} + \ldots + O_{Xn}.C_{Xn}, \text{ and}$$
$$Y = O_{Y1}.C_{Y1} + O_{Y2}.C_{Y2} + \ldots + O_{Yn'}.C_{Yn'}$$

be two *regular slices*. Let H_X and H_Y be their respective hierarchy.

The intersection is computed by applying the following rules:

1. If $H_X = H_Y$ then
$$X \cap Y = \sum_{i=1}^{n} (O_{Xi} \cap O_{Yi}).C_i$$
2. If $H_X = L_X H_Y$, then let Y_i be the slices resulting from the left completion of Y with the successive calendars in L_X (from right to left).
$$X \cap Y = \bigcup_{i} (X \cap Y_i)$$
3. If $H_X = LM_X R_X$ and $H_Y = LM_Y R_Y$ with $M_X = (C_{X\ell}, \ldots, C_{Xk}) \wedge M_Y = (C_{X\ell}, \ldots, C_{Yk'}) \wedge$ $C_{X\ell} \neq C_{Y\ell} \wedge C_{X(k-1)} \neq C_{Y(k'-1)} \wedge$ $C_{Xk} = C_{Yk'} \wedge ((k = k' = n) \vee (C_{X(k+1)} = C_{Y(k'+1)}))$, That is X and Y are mutually synchronized until $C_{X\ell}$. They are no longer synchronized between $C_{X\ell}$

and C_{Xk}, and are resynchronized from C_{Xk} (but no necessary till C_{Xn}), then let

$$X_L = \sum_{i=1}^{\ell-1} O_{Xi}.C_{Xi}, \quad X_M = \sum_{i=\ell}^{k} O_{Xi}.C_{Xi}, \quad X_R = \sum_{i=k+1}^{n} O_{Xi}.C_{Xi},$$
$$Y_L = \sum_{i=1}^{\ell-1} O_{Yi}.C_{Yi}, \quad Y_M = \sum_{i=\ell}^{k'} O_{Yi}.C_{Yi}, \quad Y_R = \sum_{i=k'+1}^{n'} O_{Yi}.C_{Yi}.$$

If $(L \neq \emptyset \vee I_L = X_L \cap Y_L \neq \emptyset) \wedge (R \neq \emptyset \vee I_R = X_R \cap Y_R \neq \emptyset)$ then

$$X \cap Y = \bigcup_{i,j} (X_{Li} \cap Y_{Lj}) + I_R$$

where the X_{Li} and Y_{Lj} are respectively the results of the *mutual synchronization* of $I_L + X_M$ and $I_L + Y_M$.

3.1.3 Mutual synchronization of regular slices

Consider two regular slices X and Y of the form:

$$X = \sum_{i=1}^{\ell-1} O_i.C_i + \sum_{j=\ell}^{k} O_{Xi}.C_{Xi}$$
$$Y = \sum_{i=1}^{\ell-1} O_i.C_i + \sum_{j=\ell}^{k'} O_{Yi}.C_{Yi}$$

with $C_{X\ell} \neq C_{Y\ell}, C_{X(k-1)} \neq C_{Y(k'-1)}, C_K = C_{Xk} = C_{Yk'}$.

The purpose of this algorithm is to rewrite X and Y into equivalent forms such as their hierarchy are the same. Here we will chose as target hierarchy the one of X possibly surrounded by $C_0 \sqsupset C_1$.

The algorithm consists in the following steps:

1. Compute or build the calendar C_0 such as

$$\begin{aligned} \text{duration}(C_0/C_k) &= \text{period}(C_0/C_k) \\ &= \text{lcm}(\text{period}(C_1/C_k), \text{period}(C_\ell/C_k), \text{period}(C'_{\ell'}/C_k)) \end{aligned}$$

   where lcm stands for "least common multiple". and

$$\text{offset}(C_0) = \text{offset}(C_1)$$

2. If $C_0 \neq C_1$ then apply the left completion of X and Y by C_0 which gives two sets of slices of the form:

$$\begin{aligned} X_i &= \sum_{j=0}^{\ell-1} O_{ij}.C_j + \sum_{j=\ell}^{k} O_{Xj}.C_{Xj} \\ Y_i &= \sum_{j=0}^{\ell-1} O_{ij}.C_j + \sum_{j=\ell}^{k'} O_{Yj}.C_{Yj} \end{aligned}$$

   Apply the mutual synchronization algorithm[1] to each pair of slices X_i, Y_i.

[1]Actually, execute point 3 of the algorithm for each (X_i, Y_i) renamed as (X, Y).

3. If $C_0 = C_1$ then rewrite Y as

$$Y = \sum_{i=0}^{\ell-1} O_i.C_i + F_{(Y,k',\ell)}(Y).C_{Yk'}$$

where $F_{(Y,\ell,k')}(Y)$ is a translation function described in appendix A which calculates the equivalent offsets $O'_{k'}$ of $C_{Yk'}$ with respect to $C_{Y\ell}$. And finally,

$$Y = \sum_{i=0}^{\ell-1} O_i.C_i + F^{-1}_{X,k,\ell}(Y)$$

Where F^{-1} maps Y into the hierarchy of X by decomposing $O_{k'}.C_{Yk'} = O_{k'}.C_{Xk}$ into $\sum_{i=\ell+1}^{k} O''_i.C_{Xi}$ (see appendix B).

3.2 Intersection of non-regular slices

Intersecting two non regular slices (or one regular and one non-regular) entails no particular problem if both have the same hierarchy or can be transformed to have the same one by left completion. Indeed, in this case intersection still consists in intersecting the respective offsets.

Example 3.1

$$\texttt{months} + 1.\mathit{weeks} \cap \texttt{weeks}$$

`weeks` can be left completed by `months`, so that the expression becomes

$$\texttt{months} + 1.\mathit{weeks} \cap \texttt{months} + \mathit{all}.\mathit{weeks}$$

Theses two non-regular slices are now synchronized, and the result of the intersection give us

$$\texttt{months} + 1.\mathit{weeks}$$

∎

Example 3.2 The intersection between *The first Sunday of each year* and *The first Monday of each month* will be translated into

$$\texttt{years} + 1.\mathit{weeks} + 1.\mathit{days} \cap \texttt{months} + 1.\mathit{weeks} + 2.\mathit{days}$$

As the partial intersection of the rightmost synchronized part of the two slices is empty, the global result is empty. ∎

If the non-regular parts of the slices appear to be in the mutually synchronized parts, then there is no particular problem since the intersection of these parts still consists in intersecting the respective offsets. The serious problems only appear when the non-synchronized parts are not regular. Then in general we will have to use the lri (lrp) equivalence to transform them into regular parts or use a similar technic to the one exposed in the following example.

Example 3.3

$$\begin{array}{ll} & \texttt{months} \ + \ 5.\mathit{days} \ + \ (10,11,12,13,14).\mathit{hour} \\ \cap & \texttt{months} \ + \ 1.\mathit{weeks} \ + \ 2.\mathit{days} \ + \ (8,9,10,11,12).\mathit{hour} \end{array} \tag{1}$$

The intersection of the rightmost synchronized part gives us

$$\mathit{days} \ + \ (10,11,12).\mathit{hour}$$

The left parts must be synchronized. Actually this will consist in suppressing *weeks* from the hierarchy of the second slice and find the offset of *days* with respect to *months*. But here we cannot use the translation function F to achieve this since `weeks` is not a subcalendar of `months`.

However `months` $+\ 1.\mathit{weeks}\ +\ 2.\mathit{days}$ means *"the second days of the first week of each month"*. This *"first week of each month"* is characterized by the fact that *"the first day of these weeks is one of the seven first days of each month"* which can be expressed as:

$$\texttt{months} + (1,2,3,4,5,6,7).\mathit{days} \cap \texttt{weeks} + 1.\mathit{days}$$

This intersection of two regular slices results in the union of regular slices:

$$\begin{array}{ll} & \texttt{bigwheels} \ + \ 1.\mathit{months} \ + \ 1.\mathit{days} \\ \cup & \texttt{bigwheels} \ + \ 2.\mathit{months} \ + \ 4.\mathit{days} \\ \vdots & \\ \cup & \texttt{bigwheels} \ + \ 335.\mathit{months} \ + \ 6.\mathit{days} \\ \cup & \texttt{bigwheels} \ + \ 336.\mathit{months} \ + \ 4.\mathit{days} \end{array}$$

We have now a set of regular slices representing *"the first day of the first week of each month"*. We can transform them by incrementing[2] their offsets of days to *"the second days of the first week of each month"*:

$$\begin{array}{ll} & \texttt{bigwheels} \ + \ 1.\mathit{months} \ + \ 2.\mathit{days} \\ \cup & \texttt{bigwheels} \ + \ 2.\mathit{months} \ + \ 5.\mathit{days} \\ \vdots & \\ \cup & \texttt{bigwheels} \ + \ 335.\mathit{months} \ + \ 7.\mathit{days} \\ \cup & \texttt{bigwheels} \ + \ 336.\mathit{months} \ + \ 5.\mathit{days} \end{array}$$

Finally the intersection (1) can be rewritten as

$$\begin{array}{ll} & \texttt{bigwheels} \ + \ \mathit{all}.\mathit{months} \ + \ 5.\mathit{days} \ + \ 1.\mathit{hour} \\ \cap & \left\{ \begin{array}{ll} & \texttt{bigwheels} \ + \ 1.\mathit{months} \ + \ 2.\mathit{days} \ + \ 1.\mathit{hour} \\ \cup & \texttt{bigwheels} \ + \ 2.\mathit{months} \ + \ 5.\mathit{days} \ + \ 1.\mathit{hour} \\ \vdots & \\ \cup & \texttt{bigwheels} \ + \ 335.\mathit{months} \ + \ 7.\mathit{days} \ + \ 1.\mathit{hour} \\ \cup & \texttt{bigwheels} \ + \ 336.\mathit{months} \ + \ 5.\mathit{days} \ + \ 1.\mathit{hour} \end{array} \right\} \end{array}$$

[2]This incrementation is straightforward since we deal with the first week of months (always containing more than a week). It would not have been the case for the fourth week of the month and in that case we should have returned to a purely lri formalism.

giving:

$$
\begin{aligned}
& \texttt{bigwheels} \ + \ 2.months \ + \ 5.days \ + \ 1.hour \\
& \vdots \\
\cup \ & \texttt{bigwheels} \ + \ 336.months \ + \ 5.days \ + \ 1.hour
\end{aligned}
$$

∎

4 Conclusion and future works

We have presented a temporal system for representing infinite temporal information combining the advantages of both a complete mathematical formalism and an efficient and convivial symbolic treatment. We are currently developing a object oriented system based on the model presented in this paper. This system will allow the "semi-symbolic" treatment of complex temporal expressions. Its potential applications range from temporal databases to scheduling system via the time specialist of an expert system. As future extension we plan to add a deductive layer above the system to grant it a very powerful query processing, as suggested in [BNW91].

A The translation function F

Consider the regular slice[3]

$$S = \sum_{i=0}^{\ell-1} o_i C_i + \sum_{i=\ell}^{k} o_i C_i$$

The translation function $F_{(S,\ell,k)}$ evaluates the offset o'_k of the C_k term such as

$$S = \sum_{i=0}^{\ell-1} o_i C_i + o'_k C_k$$

The following algorithm determines o'_k. We consider that elem(i,*list*) returns the i^e element of the *list*, length(*list*) returns the number of elements in the *list* and mod is the modulo operator.

/* Ω_j represents the global offset of the j^e element of the slice S versus its global reference $o_0.C_0$. The first value, the offset of the global reference: o_0 */
$\Omega_0 := o_0$
/* we are now computing step by step Ω_j for $0 < j \leq k$ */
for $j = 1$ to k do

/* First, we store in aux the offset of C_{i-1} expressed in terms of offset of C_i, by adding the length of the corresponding intervals returned by duration(C_{i-1}/C_i). This operation make the assumption that the em duration function takes care of the synchronization points of the calendars. */

[3]We consider here that all o_i are single offset, this can be view as one branch of the "tree" corresponding to the more general case where the o_i are sets offsets. The following algorithm can be easily extend to handle this case.

$$\text{aux} := \sum_{i=1}^{\Omega_j - 1} \text{elem}(((i-1) \bmod \text{length}(\text{duration}(C_{i-1}/C_i))) + 1, \text{duration}(C_{i-1}/C_i))$$

/* then we adjust the value of the local offset */

$\Omega_j := aux + o_\ell - 1$

/* at the ℓ step, we compute the offset equivalent to aux in terms of C_k */

if $(j = \ell)$ then

$$o'_{\ell-1} = \sum_{i=1}^{aux} \text{elem}(((i-1) \bmod \text{length}(\text{duration}(C_{i-1}/C_i))) + 1, \text{duration}(C_{i-1}/C_i))$$

endif

enddo

/* the result is the difference between the global offset $o'\ell - 1$ of the $C_{\ell-1}$ interm of C_k and the global offset Ω_k C_k term */

$o'_k = \Omega_k - o'_{\ell-1}$

B The translation function F^{-1}

Consider the the two regular slices

$$\begin{aligned} S_1 &= \sum_{i=0}^{\ell-1} o_i.C_i + \sum i = \ell^k o_i.C_i \\ S_2 &= \sum_{i=0}^{\ell-1} o_i.C_i + o'_k.C_k \end{aligned}$$

The application of translation function $F^{-1}_{(S_1,\ell,k)}$ to S_2 calculates the offsets o''_i such that

$$S_2 = \sum_{i=0}^{\ell-1} o_i C_i + \sum_{i=\ell}^{k} o''_i.C_i$$

and returns

$$\sum_{i=\ell}^{k} o''_i.C_i$$

The following algorithm computes the o''_i:

/* first, we compute both the Ω_j and $o'_{\ell-1}$ as in F's algorithm */

$\Omega_0 := o_0$

for $j = 1$ to ℓ do

$$aux := \sum_{i=1}^{\Omega_j - 1} \text{elem}(((i-1) \bmod \text{length}(\text{duration}(C_{i-1}/C_i))) + 1, \text{duration}(C_{i-1}/C_i))$$

/* then we adjust the value of the local offset */
$\Omega_j := aux + o_\ell - 1$
/* at the ℓ step, we compute the offset equivalent to aux in terms of C_k */
if $(j = \ell)$ then

$$o'_{\ell-1} = \sum_{i=1}^{aux} \text{elem}(((i-1) \bmod \text{length}(\text{duration}(C_{i-1}/C_i))) + 1, \text{duration}(C_{i-1}/C_i))$$

endif
end do

/* Knowing o'_k, the offset of C_k with respect to C_ℓ we compute the o''_i values by evaluation of the maximum number of intervals C_i that we can count in o_k making use of the Ω_i values to interpret the duration lists */
for $i = \ell + 1$ to $k - 1$ do
$o''_i = 0$
while $\text{elem}((\Omega_{(i-1)} + o''_i) \bmod \text{length}(\text{duration}(C_\ell/C_k)) + 1, \text{duration}(C_\ell/C_k)) > o''_k$
$o''_k := o''_k - \text{elem}((\Omega_\ell + o_i) \bmod \text{length}(\text{duration}(C_\ell/C_k)) + 1, \text{duration}(C_\ell/C_k))$
$o''_i := o''_i + 1$

$$\Omega_i = \sum_{j=1}^{\Omega_{i-1}-1} \text{elem}(((j-1) \bmod \text{length}(\text{duration}(C_{j-1}/C_j))) + 1, \text{duration}(C_{j-1}/C_j))$$

$\Omega_i = \Omega_i + o''_i$
end while

end do
$o''_k = o'_k$

References

[All84] James F. Allen. Toward a general theory of action and time. *Artificial Intelligence*, 23:123–154, July 1984.

[AM89] Martín Abadi and Zohar Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

[Ari86] G. Ariav. A temporally oriented data model. *tods*, 11(4):499–527, dec 1986.

[Bau89a] Marianne Baudinet. *Logic Programming Semantics: Techniques and Applications.* PhD thesis, Computer Science Department, Stanford University, Stanford, CA, February 1989.

[Bau89b] Marianne Baudinet. Temporal Logic Programming is Complete and Expressive. *16th Annual ACM Symposium on Principles of Programming Languages*, pages 267–280, January 1989.

[BNW91] Marianne Baudinet, Marc Niézette, and Pierre Wolper. On the Representation of Infinite Temporal Data and Queries. Technical report, Computer Science dep. University of Liege, 1991. Proceedings of the 10th ACM Symposium on Principles of Database Systems (PODS91).

[Bub77] Jr. Bubenko, J.A. *The Temporal Dimension in Information Modeling*, pages 93–118. North-Holland Pub. Co., The Netherlands, 1977.

[Cho90] Jan Chomicki. Polynomial time query processing in temporal deductive databases. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 379–391, Nashville Tennessee, 1990.

[CI88] Jan Chomicki and Tomasz Imielinski. Temporal deductive databases and infinite objects. In *Proceedings of the 7th ACM Symposium on Principles of Database Systems*, pages 61–73, Austin Texas, 1988.

[CI89] Jan Chomicki and Tomasz Imieliński. Relational specifications of infinite query answers. In *ACM-SIGMOD International Conference on Management of Data*, pages 174–183, Portland, Oregon, May 1989.

[CI90] Jan Chomicki and Tomasz Imieliński. Finite representation of infinite query answers. Technical Report TR-CS-90-10, Kansas State University, Manhattan, KS, August 1990.

[CW83] J. Clifford and D.S. Warren. Formal semantics for time in databases. *tods*, 8(2):214–254, jun 1983.

[Gad88] S.K. Gadia. A homogeneous relational model and query languages for temporal databases. *tods*, 13(4):418–448, dec 1988.

[GM91] D. Gabbay and P. McBrien. Temporal logic and historical databases. In *vldb*, Barcelona, Spain, sep 1991.

[KSW90] F. Kabanza, JM. Stévenne, and P. Wolper. Handling infinite temporal data. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 392–403, Nashville Tennessee, 1990.

[KSW91] F. Kabanza, JM. Stévenne, and P. Wolper. Handling infinite temporal data. Extended and revised version., sep 1991.

[LMF86] Bruce Leban, David D. McDonald, and David R. Forster. A Representation for Collections of Temporal Intervals. In *AAAI-86*, volume 1, pages 267–371. Fith National Conference on Artifitial Intelligenge, Morgan Kaufmann, August 1986.

[NA89] S. B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49:147–175, 1989.

[Sho86] Yoav Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial intelligence.* PhD thesis, Yale University, Computer Science Department, New Haven, CT, 1986.

[Sté91] Jean-Marc Stévenne. A Model-Checking Approach to Temporal Reasoning. Presented at the second Bar-Ilan Symposium on Foundation of Artificial Intelligence (BISFAI91), jan 1991.

[Tan86] A.U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[TC90] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In *vldb*, Brisbane, Australia, aug 1990.