A language to express time intervals and repetition

Diana Cukierman and **James Delgrande**

School of Computing Science
Simon Fraser University
Burnaby, BC, Canada V5A 1S6
{diana,jim}@cs.sfu.ca

Abstract

We are investigating a formal representation of *time units*, *calendars*, and *time unit instances* as restricted temporal entities for reasoning about repeated activities. We examine characteristics of time units, and provide a categorization of the hierarchical relations among them. Hence we define an abstract hierarchical unit structure (a *calendar structure*) that expresses specific relations and properties among the units that compose it. Specific time objects in the time line are represented based on this formalism, including non-convex intervals corresponding to repeated activities. A goal of this research is to be able to represent and reason efficiently about repeated activities.

1 Introduction

The motivation for this work is to ultimately be able to reason about schedulable, repeated activities, specified using calendars. Examples of such activities include going to a specific class every Tuesday and Thursday during a semester, attending a seminar every first day of a month, and going to fitness classes every other day. Defining a precise representation and developing or adapting known efficient algorithms to this domain would provide a valuable framework for scheduling systems, financial systems, process control systems and in general date-based systems. We search for a more general, and formalized representation of the temporal entities than in previous work. We explore further the *date* concept, building a structure that formalizes dates in calendars.

Schedulable activities are based on conventional systems called *calendars*. We use as a departure point "calendar" in the usual sense of the word. Examples of calendars include the traditional Gregorian calendar, university calendars, and business calendars, the last two groups being defined in terms of the Gregorian. The framework we define concerns a *generic* calendar abstract structure, which subsumes the mentioned calendars, and arguably any system of measures based on discrete units. Calendars can be considered as repetitive, cyclic temporal objects. We define an abstract structure that formalizes calendars as being composed of *time units*, which are related by a

decomposition relation. The decomposition relation is a containment relation involving repetition and other specific characteristics. Time units decompose into contiguous sequences of other time units in various ways. A *calendar structure* is a hierarchical structure based on the decomposition of time units. This structure expresses relationships that hold between time units in several calendars. We refer to the concept of time unit *instances*, and distinguish different levels of instantiation. Whereas "month" refers to a time unit class, "June" is referred to as a *named time unit*. June is one of the 12 occurrences of the notion of *month* with respect to *year*, and viewed extensionally it represents the set of all possible occurrences of June. Finally, "June 1994" is one specific instance of a month.

2 Related work

Our formalism deals with time units, which are a special kind of time interval with inherent durations. Therefore we base our work on *time intervals* as the basic temporal objects [1]. In [21], the *time point algebra* is developed, based on the notion of time point in place of interval. Computation of the closure of pairwise time-interval relations in the full interval algebra is NP-complete, whereas the time point algebra has a polynomial closure algorithm. However the time point algebra is less expressive than the interval algebra: certain disjunctive combinations of relations are not expressible with the time point algebra. Nonetheless, some applications do not require the full expressive power of the interval algebra, and can benefit from efficient (albeit less expressive) representations. Hence it is of interest to study restrictions of the interval algebra.

The present framework deals with restricted kinds of intervals within a hierarchical structure. The intent is that the hierarchy provide a basis for obtaining efficient algorithms for certain operations. (This may be contrasted with [11] which considers a hierarchical structure in the general interval algebra.) Non-convex intervals (intervals with "gaps") are employed in [12, 13] when using time units in a repetitive way or when referring to recurring periods. The time unit hierarchy proposed in our work generalizes [13], in that temporal objects can be represented by any se-

quence of composed time units, as opposed to fixed in Ladkin's approach. Moreover, we are able to *combine* systems of measurement, and so talk about the third month of a company's business year as corresponding with June in the Gregorian calendar. [13] also does not take into account the varying duration of specific time instances, a matter addressed and formalized in our work. In addition we address the varying duration of specific time instances. [18] also elaborate on the notions of non-convex interval relations defined in [12, 13] while [16] proposes a generalization of non-convex intervals. Leban et. al. [15] deals with repetition and time units. This work relies on sequences of consecutive intervals combined into "collections". The collection representation makes use of "primitive collections" (essentially circular lists of integers), and two basic operators, *slicing* and *dicing*, which subdivide an interval and select a subinterval respectively. Poessio and Brachman [19] are mainly concerned with the implementation of algorithms to detect overlapping repeated activities. This work relies on temporal constraint satisfaction results and algorithms [8]. [19] also introduces the concept of using dates as reference intervals to make constraint propagation further efficient. We envision our proposed formalism provides a useful framework to follow this idea. [5] exposes a set theoretic structure for the time domain with a calendar perspective. It formalizes the temporal domain with sets of "constructed intervallic partitions" which have a certain parallel to our decomposition into contiguous sequences of intervals. However, this formalization is more restricted than ours and can not handle the Gregorian nor general calendars.

[3] and [4] propose formalizations that deal with calendars and time units. These two papers are interestingly related to our research, even though they have evolved independently. These works are analyzed and compared with our research in the Section 5.

3 Time units and time unit instances

The central element of our formalism is that of a *time unit*. Time units represent *classes* of time intervals, each with certain commonalties and which interact in a limited number of ways. For example, *year* and *month* are time units. Something common to every *year* is that it decomposes into a constant number of months. A characteristic of *month* is that it decomposes into a non-constant number of days, which vary according to the instance of the month.

Properties that are common to time units determine the time unit class attributes. In [6, 7], the *identifier* of a time unit class and the *(general) duration* are presented. Relative and general durations are compared and the concept of an *atomic time unit* is introduced. Time unit instances and their numbering and naming is described as well. All attributes are formally defined in a functional way. We here present a summary of these concepts with examples from the Gregorian calendar.

Identifier of a time unit The first attribute of the time unit class is a unique *identifier*, a time unit name, for example *year* or *month*. We distinguish time unit names when the time units have the same duration but differ in their origin. For example, *years* in the Gregorian calendar start in January, but *academic years*, in university calendars in the northern hemisphere, start in September. *Year* and *academic year* are two different time units, which have several properties in common.

Duration Time units inherently involve *durations*; a time unit expressly represents a standard adopted to measure periods of time. Different time unit instances of the same time unit class can have different durations. For example different months have different number of days, from 28 to 31. Accordingly, the attribute representing a duration of a *class* is a *range* of possible values. We represent this range by a pair of integers, the extremes of the range of possible lengths any time unit instance can have. Thus, *month* as a class has a duration of $(28, 31)$.

A duration referred to as *general* will be expressed in a *basic* unit, common to all the time units in one calendar. For example, using *day* as a common or basic unit, *month* has a (general) duration of $(28, 31)$ days. A specific month, for example, *February 1994*, has a duration of 28 days. We also define another kind of duration; a *duration relative to another time unit*. An important reason why (general) durations of all time units in a calendar are defined based on the same basic measure unit is to be able to compare them. This notion plays a fundamental role in the partial order among time units in the same calendar or variants of a calendar.

Time units decompose into smaller units up to a finite level at which a time unit is *atomic* — a non-divisible interval. When the time interval is non-decomposable it is called a *time moment*, following [2]. Time units therefore model time in a *discrete fashion*. Time units may be considered atomic in one application and decomposable into smaller time units in other applications, depending on the intended granularity for the application [10].

Instance names and numbers The *name* or *number* of a time unit instance can be expressed in several ways. Similar to durations, the name is relative to another time unit, a *reference time unit*. For example, a *day* instance can be named from *Sunday* to *Saturday* or numbered 1 to 7 if *week* is the reference time unit of *day*. (Names can be thought of as synonyms for the numbers). Instances of time units which do not have any reference time unit are numbered relative to a conventional *reference point* or *zero point* of the calendar, for example the Christian Era for the Gregorian calendar.

The ordered set of *all the possible* names the instances of a time unit can take within a reference can be expressed extensionally, by a sequence of names (if there are names associated to the time unit class), or a pair of numbers representing the range of maximum possible instance numbers of a class. This sequence (or pair of numbers) is an attribute of the time unit class. A particular instance name or number of a time unit within a reference reflects the relative position of the instance within the reference, given a certain *origin* where counting of instance values starts. For example, months are counted from 1 to 12, in a year in the Gregorian calendar, starting from 1. But, in the case of months counted within an *academic year*, the origin of numbering is not month number 1, but rather the $9th$ (or *September*), so the $3rd$ month of a *academic year* is the $11th$ month of all the possible months name sequence (or *November*). *Circular counting* is assumed. More detail appears in [6].

3.1 Decomposition of time units

The primary relation among time units is that of *decomposition*. When A decomposes into B, A will be referred to as the *composed* time unit, and B will be referred to as the *component* unit. For example, a *year* decomposes into *months* and a *month* into *days*. Also a *month* decomposes into *weeks*, a *week* into *days*, etc. Clearly there are different kinds of decompositions: a year decomposes exactly into 12 months, whereas a month decomposes in a non-exact way into weeks, since the extreme weeks of the month may be complete or incomplete weeks. We propose that all these variations in the decomposition relation can be captured with two different aspects of the relation: *alignment* and *constancy*. A time unit may decompose into another in an *aligned* or *non-aligned* fashion. The decomposition is *aligned* just when the composed time unit starts exactly with the first component and finishes exactly with the last component. Consequently, a certain number of *complete* components fit exactly into the composed time unit. Examples of aligned decompositions include *year* into *months*, *month* into *days*, and *week* into *day*. Examples of non-aligned decompositions include *year* into *weeks* and *month* into *weeks*. Figure 1 shows a graphical picture of aligned and non-aligned decomposition.
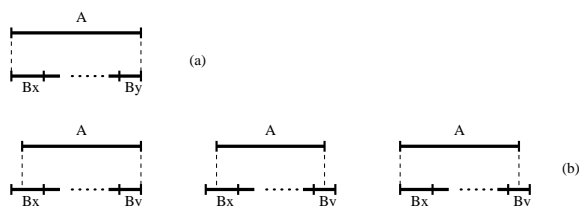

Figure 1: Graphical representation of Alignment

A time unit may decompose into another in a *constant* or *non-constant* fashion. The decomposition is *constant* when the component time unit is repeated a constant number of times for *every* time unit instance. Examples of constant decompositions include *year* into *months* and *week* into *days*. Examples of non-constant decompositions include *month* into *days* and *year* into *days*.

There are four possible combinations resulting from these two aspects of alignment and constancy. These combinations cover examples from the various calendars analyzed. (Arguably) all the relationships of interest in such systems are covered by the variants resulting from combining alignment and constancy of decomposition.

We analyze the composition (or product), intersection (or sum) and inverse of decomposition relations. For example, if two aligned decomposition relations are multiplied, the resulting decomposition relation is also aligned. For example *year* decomposes into *month*, *month* decomposes into *day*. These two (aligned) decomposition relations can be composed (or multiplied) to obtain the (aligned) decomposition relation of *year* into *day*.

It should be noticed that these three operations (composition, intersection and inverse) are defined among decomposition between time unit *classes*. This is to be contrasted to the relational algebras defined in the literature and constraint propagation algorithms, which deal with relations between time intervals in the time line, i.e., at the instance level (for example [1, 18, 14].) A detailed study of these operations appears in [6].

3.2 Calendar Structures: time unit hierarchies

A calendar structure is a pair: a set of time units and a decomposition relation. We obtained the following result:

Theorem 1 (Decomposition) *The decomposition relation is a particular case of containment, and constitutes a partial order on the set of time units in a calendar.*

Therefore, a calendar structure is defined as a containment *time unit hierarchy*. The structure is defined so that *variants* of a specific calendar can be defined in terms of a basic one. Such would be the case of a university or business calendar, based on the Gregorian calendar. Such calendars have the same time units as the basic one, with possible new time units or a different conventional beginning point. For example, many university calendars would have a *semester* time unit, where the academic year begins in September. Since calendar structures are partial orders, they can be represented by a directed acyclic graph. The set of nodes in a calendar structure represents the set of time units. Edges represent the decomposition relation. The intended application of use of the calendar structure determines which level of time units is included.

Chains We develop a categorization based on the subrelations of decomposition, i.e. considering only constant/aligned decompositions, or aligned only, or constant only. *Calendar substructures*, composed of *chains* result from these subrelations. The term "chains" appears in [17], however there chains are defined on time intervals on the time line. In our case we are referring to chains of time unit classes. Nonetheless, chains of time unit classes are directly related to expressions that represent time unit instances, and influence greatly in efficiency matters. To give an example, the operation of converting time unit instances from one time unit to another will be more efficient when the time units intervening in the time unit instance expression decompose in a constant/aligned way; divisions and multiplications can be done, whereas it is necessary to have some iterative process of additions or subtractions when the decomposition is not exact.

A chain is a consecutive linear sequence of time units, such that each one decomposes into any other in the chain in the same way. Hence all time units in an aligned chain decompose in an aligned way, etc. A calendar structure can be organized according to these chains. There is a different subgraph associated to each type of decomposition. For example, Figure 2 shows the Gregorian calendar structure characterized by two aligned chains: *<28-centuries, 4-centuries, century, year, month, day, hour>* and *<28-centuries, week, day, hour>*. In this figure we can observe there are three *common nodes* to both chains: *28-centuries, day* and *hour*.
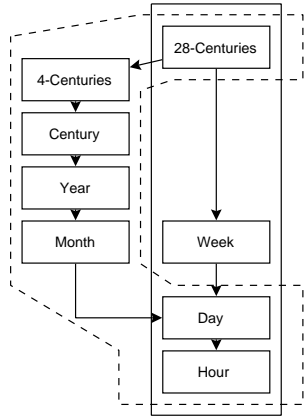

Figure 2: Aligned Gregorian calendar substructure

We refer to the subgraph that results from organizing the calendar structures according to a certain type of chain as a *calendar substructures* of that particular type of decomposition (constant, aligned or constant/aligned). Therefore a chain of a certain type is composed by all those time units in the hierarchy that form a path in the corresponding calendar substructure.

3.3 A language of the set of time units

We define a language of time units based on the fact that calendar structures can be organized in constant/aligned, constant or aligned chains. Together with the organization of calendar structures in chains, we distinguish special time units as *primitive*. The set of primitive time units includes one time unit per chain in a minimal way. Hence, in case that all chains have at least one common node, the primitive set is a singleton. It was proved [6] that any calendar structure can be extended (i.e. added time units) so that there is such unique common time unit to all chains, for any type of chain. For example, if the Gregorian calendar is organized as in Figure 2, there are three minimal sets of primitive time units: *{hour}*, *{day}* or *{28-centuries}*.

The main idea of this language is that all time units in the structure can be recursively constructed starting from a set of primitive time units, decomposing or composing them, with decomposition limited to an aspect (for example only aligned).

The symbols that are used in this language are: A set of primitive time units, PRIM = $\{P, P_1, P_2, \ldots\}$, a set of special symbols, $\{*, \ /, \ (, \ )\}$ a set of (possibly infinitely many) constants, CONS = $\{second, minute, hour, \ldots\}$ and a language to express durations, DUR = $\{d \ | \ d \in N^+\} \cup \{(d_1, d_2) \ | \ d_1, d_2 \in N^+ \ and \ d_1 < d_2\}$. The language is defined:

1. If $P \in$ PRIM, then $P \in$ TUS.
2. If $C \in$ CONS, then $C \in$ TUS.
3. If $T \in$ TUS and $D \in$ DUR then $(T * D) \in$ TUS.
4. If $T \in$ TUS and $D \in$ DUR then $(T/D) \in$ TUS.
5. Those are all the possible elements of TUS.

Briefly, $T = (S * D)$ when T is composed of a contiguous sequence of D S's and $T = (S/D)$ when D contiguous T's compose S. For example, if we organize the Gregorian calendar with aligned chains, the following is a possible interpretation of the language:

PRIM $= \{day\}$; $month = day * (28, 31)$; $year = day * (365, 366) = month * 12$; $hour = day/24$; $twoday = day * 2$; $trimester = month * 3$.

Generally we consider only one time unit as primitive. It is convenient to include constants into the language. We abuse notation in that we name constants, which are part of the alphabet of symbols, by the name of the domain elements. Thus if Φ is an interpretation function, Φ : TUS $\times$ $Calendar \longrightarrow Time\_units\_in\_the\_calendar$. For example $\Phi(month, Gregorian) =$ "*month*", $\Phi(month * 3, University) =$ "*trimester*". As long as it does not lead to ambiguities, we will not make an explicit use of this interpretation function in this paper. Using the same strings for constants and domain elements appears elsewhere, for example [20].

3.3.1 Named time units

As discussed in previous sections, there is more than one level of time unit instantiation. For exam-

ple, a subclass of *month* in the Gregorian calendar could be the fourth month (synonym of *April*). This does not represent a specific month yet. We call *April* a *named-month*, and viewed extensionally, it represents the set of all specific instances of the month "April". A specific instance would be "April 1995". To be able to express specific instances we define *calendar expressions* in the next section.

Named time units are dependent on the time unit, a reference time unit and a position within the reference. Names and numbers are functions defined on the sets of time units, and some involving instances as well. We write $name(T, R, X)$ to represent the Xth "named-T" within the reference time unit R, such that R decomposes into T. If X is outside permissible values, or if the time unit has no associated names with that reference $name(T, R, X)$ represents an inconsistent value ($\perp$). Examples of this function include $name(month, year, 3) = $ "*March*"; $name(month, academic_year, 3) = $ "*November*"; $name(day, week, 8) = \perp$. Numbered-time units are defined in an analogous way. For example $number(month, year, 3) = $ "*3*". A function related to named and numbered time units is the *last possible value*. Some cases, as already explained, will not provide a unique value, but a range of last values. In those cases there exists a unique last value only at the instance level (and not at the class level). For example, $last_name(month, year) = $ "*December*"; $last\_number(day, month) = (28, 31)$; $last\_number\_instance(day, February\ 1994) = 28$.

4 Calendar expressions

A goal of our research is to represent and reason with time unit instances, that is, the temporal counterpart of single or repeated activities occurring in the time line. We want to express these time entities in terms of days, weeks, hours, etc, relative to certain conventional calendars. Calendar structures above defined provide a formal apparatus of units on which to represent a date-based temporal counterpart of activities. Intervals, time points and moments in the time line will be represented in terms of these units. Specific time objects are expressed based on the time units in a calendar structure, via *calendar expressions*. A basic calendar expression is defined by a conventional beginning reference point or *zero point* associated to the calendar, and a finite sequence of pairs: $Z \oplus [(t_1, x_1), \ldots, (t_n, x_n)]$. Each pair (t_i, x_i) contains a time unit t_i from the calendar structure and a numeric expression x_i. Numeric expressions include numbers and variables ranging over the set of integers. The pairs in the sequence are ordered so that any time unit in a pair *decomposes into* the time unit in the *following* pair in the sequence. We also define duration expressions. These are similar to calendar expressions in that they consist of a list of pairs *(time unit,value)*, but have no beginning reference point nor included variables. The operation of adding a duration expression to a calendar expression defines a new calendar expression.

4.1 A language for simple and repetitive time unit instances: calendar expressions

The formal definition of the language of calendar expressions is presented next. The language of duration expressions is not introduced here for space reasons, it follows a similar style as the calendar expressions language. The decomposition relation is used in the definitions. The same kind of decomposition used to define the time units in the calendar structure (and therefore the time units language, TUS) is used in these languages.

The language of calendar expressions $CALXS$, uses the following: A conventional zero point, Z, a set of special symbols $\{\oplus, (,), [,], \cap, \cup, /, Last\}$, a set of time units (TUS), a set of variables (VAR), which will be ranging in the set of positive naturals, a set of duration expressions(DURXS), and can be defined as:

1. If $T \in$ TUS, $X \in N^+ \cup$ VAR then $(Z \oplus [(T, X)]) \in$ CALXS.

2. If $T_1, T_2 \in$ TUS, such that T_1 decomposes into T_2, $X_2 \in N^+ \cup$ VAR $\cup \{Last\}$ and $Z \oplus [CX, (T_1, X_1)] \in$ CALXS, then $(Z \oplus [CX, (T_1, X_1), (T_2, X_2)]) \in$ CALXS.

3. If $CX(\overline{v}) \in$ CALXS, where $\overline{v}$ are the variables in CX, then $(CX(\overline{v})$ **where** $Exp(\overline{v})) \in$ CALXS; $Exp(\overline{v})$ is an expression constraining $\overline{v}$.

4. If CX_1 and $CX_2 \in$ CALXS then $(CX_1 \cup CX_2)$, $(CX_1 \cap CX_2)$, $(CX_1 / CX_2) \in$ CALXS.

5. If $CX \in$ CALXS and $DX \in$ DURXS then $(CX + DX) \in$ CALXS.

6. These are all the possible elements of CALXS.

4.2 Examples of calendar expressions

The following examples express calendar expressions in the Gregorian calendar. The zero point is the beginning of the Christian Era (CE).

1. $CE \oplus [(year, X), (month, 4)]$. The fourth month within any year. (Viewed extensionally, it represents the set of all specific instances of the month "April"). X is a non-quantified variable.

2. $CE \oplus [(year, 1994), (week, 17), (day, 5)]$. The 5th day within the 17th week of the year 1994.

The last time unit in the sequence provides the *precision* of the time unit instance. Thus, in Example 1 above, the precision is *month*. It can also be observed that a calendar expression with *no* variables is a single convex interval; a time unit instance of the last time unit in the calendar expression. Consequently, $Z \oplus [(t_1, x_1), \ldots, (t_n, x_n)]$ is a (single) t_n-instance. Example 2 above is a (single) *day-instance*.

A calendar expression with variables represents a *set* of time unit instances. These sets of intervals are the temporal counterpart of a date-based repeated activity, a specific case of a non-convex interval, as defined in [12] . Thus, $CE \oplus [(year, X), (day, 175)]$ represents a non-convex interval, and the subintervals are the days numbered 175th. As well, when there are variables in the calendar expression, these can be constrained with logic/mathematical expressions. This example can be extended to:

3. $CE \oplus [(year, Y), (day, 175)]$ `where` ($Y >$ 1992 `and` $Y < 1996$). The 175th day from the beginning of each year after 1992 and before 1996. Hence starting and ending times of non-convex intervals can be expressed straightforwardly.

Another extension to the language of calendar expressions consists of applying set operations (union, intersection and difference) to combine non-convex intervals, viewing non-convex intervals as sets of subintervals. (limit cases could produce an empty set of intervals, which we consider a limit case of calendar expression). The following illustrate this possibility:

4. $CE \oplus [(year, 1994), (month, 11), (day, X)] \cap CE \oplus [(year, 1994), (week, W), (day, Tuesday)]$. Tuesdays of November 1994.

5. $CE \oplus [(year, 1994), (month, 11), (day, X)]$ / $(CE \oplus [(year, 1994), (month, W), (day, Y)]$ `where` ($Y = 7 \ or \ Y = 1$)). Days of November 1994 which are not Saturdays nor Sundays, i.e., weekdays of November 1994. Saturday and Sunday are abbreviations of certain days numbers as numbered within *week*. / represents set difference.

Finally we also allow calendar expressions to be moved (or displaced) by adding a certain (convex) interval. For example, *October 1994* plus one month is *November 1994*. We use *duration expressions* to express moved expressions. For example:

6. $CE \oplus [(year, 1994), (month, 11), (day, 5)] + [(day, 5)]$. The 5th day of November is moved 5 days, resulting in the 10th day of November 1994.

7. $CE \oplus [(year, 1994), (month, X), (day, 5)] + [(day, 5)]$ The 10th day of every month in 1994.

It is worth noticing that $[(month, 1)]$ is a duration expression representing one month. On the contrary, $CE \oplus [(year, Y), (month, 1)]$, stands for the non-convex interval of all January's. Named time units are precluded as a valid duration.

The following examples show how to represent slightly more elaborated periodicity patterns.

8. $CE \oplus [(year, 1994), (week * 2, W), (day, 1)]$. Every other Monday of 1994.

9. $CE \oplus [(year, 1994), (month, M), (day, Last)]$. Last day of every month. "*Last*" is based on the *last_number* function above presented.

Examples 1, 2 and 8 above are accounted by rules 1 and 2 of the language. Rule 3 allows to have calendars with variables that are constrained, as in example 3. Examples 4 and 5 are covered by rule 4. Finally examples 6 and 7 correspond to rule 5.

4.3 Semantics of calendar expressions

A set-based semantics is proposed for these languages. Interpretation of duration expressions and calendar expressions is based on an interpreted calendar structure. Recall that we conventionally name constants in the language of the time units (TUS) as the elements in the domain, i.e. as the time units in the calendar. Therefore, for the sake of a more clear presentation we will omit the application of an interpretation function when referring to interpreted time units whenever this does not lead to ambiguities. Examples will be made within the Gregorian calendar. Hence the zero point Z will be interpreted as year zero of the Christian Era (CE).

A duration expression is interpreted as a convex interval in the time line. Comparison, addition and subtraction of duration expressions are defined. Addition and subtraction can be viewed as translations or displacements in the time line. Limit cases, circular counting, difference in precision, etc, are taken into account. This is not presented in this paper.

Calendar expressions are interpreted as *sets of intervals* in the time line. Let Ψ be a function interpreting calendar expressions. (We omit here the specification of the calendar to simplify this presentation, and again will provide examples from the Gregorian Calendar), then $\Psi : CALXS \longrightarrow 2^{time \ intervals}$.

1. $\Psi(Z \oplus [(T, X)]) =$

a. If $X \in N^+$, the singleton with the Xth occurrence of the interval T starting from $\Psi(Z)$. For example, $\Psi(Z \oplus [(year, 1995)]) = \{year \ 1995\}$.

b. If X is a variable, the set of time intervals T starting from $\Psi(Z)$, i.e. $\bigcup_{X=1}^{\infty} \Psi(Z \oplus [(T, X)])$. For example, $\Psi(Z \oplus [(year, Y)]) =$ set with every year since $CE = N^+$.

2. $\Psi(Z \oplus [CX, (T_1, X_1), (T_2, X_2)]) =$

a. If $X_2 \in N^+$, the set of X_2^{th} subintervals T_2 within each interval in the set $\Psi(Z \oplus [CX, (T_1, X_1)])$. For example, $\Psi(Z \oplus [(year, Y), (month, April)]) = \{April \ 1, \ April \ 2, \ ...\}$.

b. If $X_2 = Last$, the set of Lth subintervals T_2 within each interval in the set $\Psi(Z \oplus [CX, (T_1, X_1)])$, where $L = last\_number(T_2, T_1)$ as defined in Section 3.3.1 above. For example $\Psi(Z \oplus [(year, Y), (month, Last)]) = \{ December \ 1, \ December \ 2, \ ...\}$. In case T_1 decomposes into T_2 in a non-constant way, L depends on X_1 and possibly on CX. For example $\Psi(Z \oplus [(year, 95), (month, M), (day, Last)]) = \{January \ 31 \ 1995, \ February \ 28 \ 1995, \ ...\}$.

c. If X is a variable, the set of all time intervals T_2 within each interval in the set $\Psi(Z \oplus [CX, (T_1, X_1)])$, i.e. $\bigcup_{X=1}^{L} \Psi(Z \oplus [CX, (T_1, X_1)])$, where $L = last\_number(T_2, T_1)$. For example, $\Psi(Z \oplus$

$[(year, 1995), (month, M)]) = \{ January\ 1995,\ February\ 1995,\ \ldots,\ December\ 1995\ \}$.

3. $\Psi(CX(\overrightarrow{v})$ where $Exp(\overrightarrow{v})) =$
$\bigcup_{\overrightarrow{v}\ |\ Exp(\overrightarrow{v})} \Psi(CX(\overrightarrow{v}))$.

For example, $\Psi(Z \oplus [(year, 95), (month, M)]$ where $(M > 3$ and $M < 6)\ ) = \{April\ 1995,\ May\ 1995\}$.

4. $\Psi(CX_1 \cup CX_2) = \Psi(CX_1) \cup \Psi(CX_2)$
$\Psi(CX_1 \cap CX_2) = \Psi(CX_1) \cap \Psi(CX_2)$
$\Psi(CX_1/CX_2) = \Psi(CX_1)/\Psi(CX_2)$

See examples in Section 4.2 above.

5. $\Psi(CX + DX) =$
$\bigcup T_{\Psi(DX)}(Interval),\ \forall\ Interval \in \Psi(CX)$

Addition of a duration expression is interpreted as a translation (T) or displacement of all the subintervals denoted by the calendar expression, eventually a single one $(\Psi(CX))$, by a distance defined by the duration expression $(\Psi(DX))$. See examples in Section 4.2 above.

5 Alternative proposals dealing with calendars

[3] by Chandra et.al. present a language of "calendar expressions" to define, manipulate and query what they call "calendars". In terms of terminology used, their and our proposals coincidentally refer to "calendars" and "calendar expressions". The notions however are of a different nature; the two researches have evolved completely independently.

We refer to "calendar" in the usual sense, where the Gregorian calendar or a university calendar are possible examples. On the other hand, their "calendars" are structured collections of intervals, as defined in [15]. The difference between both "calendar expressions" is more subtle. Both calendar expressions define, in different ways, lists of points and intervals in the time line. However, our expressions are of a more declarative nature, whereas their expressions are closer to a procedure to obtain such intervals. For example, we express the collection of Fridays as: $CE \oplus [(year, Y), (week, W), (day, 5)]$. The calendar expressions in [3] are meant as a way of creating and manipulating their "calendars" (i.e., collections of intervals). *[5]/DAYS:during:WEEKS* expresses the Fridays collection in their language. There are two operations involved here: *selection* and the *strict foreach* operation (or *dicing*, as defined in [15]) with the operator *during*.

We have continued further a formal definition of the domain and languages. On the other hand, Chandra et al.'s work includes a description of an algorithm to parse their expressions and generate a procedural plan to produce an evaluation plan, implemented in Postgress. As well, they outline how to incorporate calendar expressions to temporal rules in a temporal data base. We have not dealt with temporal databases; however it is perceived as an area in which to apply our framework.

Ciapessoni et al. [4] define a many sorted first order logic language, augmented with temporal operators and a metric of time. The language is an extension of a specification language TRIO [9]. We will focus the present discussion on the semantics of their system, specially in the extension they provide to embed time granularities in the language. At this stage of our research, this is where we see the two works as being comparable.

The semantics of the language in [4] includes a temporal universe which consists of a finite set of disjoint and differently grained "temporal domains". Each temporal domain contains temporal instants expressed in the corresponding granularity. Time domains relate with a granularity (or coarseness) relation and a disjointedness relation.

Very briefly, our "time units classes" are related to their "time domains"; our "decomposition relation" to their "granularity relation"; our "repetition factor" to their "conversion factor", our "aligned decomposition" to their "disjointedness" relation. Another similar characteristic in both proposals is the distinction between time moments and durations (or displacements in their terminology).

We deal with a more general concept of repetition factor. In fact we extensively address non-constant decomposition. We emphasize how to express temporal entities based on dates with the conventions of the Gregorian and other calendars. At the present stage we specially address the formal representation and reasoning about repetitive temporal terms. As well, we believe that our abstraction of decomposition with only two characteristics (constancy and alignment) defines the relations of interest between the time unit classes in a more general way, as compared to the distinction of different relations in [4].

6 Discussion

The goal of this work is to be able to represent and reason efficiently about repeated activities using the formalism. We have defined a hierarchical structure of time units, emphasizing on decomposition among the time units. We take into consideration the distinction between *time unit classes*, *named time units* and specific *time unit instances*, (for example "*month*", "August", and "August 1994"). A characterization of the hierarchical structure considers subrelations of decomposition, according to the different aspects of decomposition: aligned, constant, and constant/aligned. A path in such a substructure is referred to as a *chain*, where all time units decompose in the same way. Chains constitute the basis we use to define a formal language to characterize time units.

We also introduce a language to represent time unit *instances*, called *calendar expressions*. These ex-

pressions provide for a straightforward way of representing the temporal counterpart of repeated activities. We envision that reasoning with calendar expressions based on time units in one chain will produce very efficient operations. Constant/aligned chains are expected to be particularly efficient.

Currently we are working on further formalizing useful operations between calendar expressions. A set based semantics has been provided for such expressions. We are investigating alternative semantic characterizations. We are also considering adding more expressive power to the language to consider for example expressions that add uncertainty, as for example "twice a week". Important future research includes studying algorithms that would best fit with this formalism, so that we may obtain efficient inferences when reasoning about repeated activities. Algorithms developed for qualitative and/or quantitative temporal constraint satisfaction problems, or variations, could be considered in this matter. A language which allows one to define time units of a variant calendar, such as a university calendar, in terms of a basic calendar, such as the Gregorian, is under study also. Finally, we believe that the formalism defined represents a generic approach, appropriate to ultimately reason efficiently with repeated events within any measurement system based on discrete units that relate with a repetitive containment relation, such as the Metric or Imperial systems.

References

[1] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] J. F. Allen and P. J. Hayes. Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5:225–238, 1989.

[3] R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars and temporal rules in next generation databases. In *Proc. of the International Conference on Data Engineering, ICDE-94*, pages 264–273, 1994.

[4] E. Ciapessoni, E.Corsetti, A. Montanari, and P. San Pietro. Embedding time granularity in a logical specification language for synchronous real-time systems. *Science of Computer Programming*, 20:141–171, 1993.

[5] J. Clifford and A. Rao. A simple, general structure for temporal domains. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal aspects of information systems*, pages 17–28. Elsevier Science Publishers B.V. (North-Holland), 1988.

[6] D. Cukierman. *Formalizing the temporal domain with hierarchical structures of time units*. M.Sc. Thesis. Simon Fraser University, Vancouver, Canada, 1994.

[7] D. Cukierman and J. Delgrande. Hierarchical decomposition of time units. In *Workshop of Temporal and Spatial reasoning, in conjunction with AAAI-94*, pages 11–17, Seattle, USA, 1994.

[8] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[9] C Ghezzi, D. Mandrioli, and A. Morzenti. Trio, a logic language for executable specifications of real-time systems. *Journal of Systems and Software*, 12(2), 1990.

[10] J. Hobbs. Granularity. In *Proc. of the Ninth IJCAI-85*, pages 1–2, 1985.

[11] J. A. G. M. Koomen. Localizing temporal constraint propagation. In *Proc. of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 198–202, Toronto, 1989.

[12] P. Ladkin. Time representation: A taxonomy of interval relations. In *Proc. of the AAAI-86*, pages 360–366, 1986.

[13] P. B. Ladkin. Primitives and units for time specification. In *Proc. of the AAAI-86*, pages 354–359, 1986.

[14] P. B. Ladkin and R. D. Maddux. On binary constraint problems. *Journal of the ACM*, 41(3):435–469, 1994.

[15] B. Leban, D. D. McDonald, and D. R. Forster. A representation for collections of temporal intervals. In *Proc. of the AAAI-86*, pages 367–371, 1986.

[16] G. Ligozat. On generalized interval calculi. In *Proc. of the AAAI-91*, pages 234–240, 1991.

[17] S. A. Miller and L. K. Schubert. Time revisited. *Computational Intelligence*, 6(2):108–118, 1990.

[18] R. A. Morris, W. D. Shoaff, and L. Khatib. Path consistency in a network of non-convex intervals. In *Proc. of the IJCAI-93*, pages 655–660, 1993.

[19] M. Poesio and R. J. Brachman. Metric constraints for maintaining appointments: Dates and repeated activities. In *Proc. of the AAAI-91*, pages 253–259, 1991.

[20] R. Reiter. Towards a logical reconstruction of relational database theory. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modeling*, pages 191–238. Springer-Verlag, 1984.

[21] M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. of the AAAI-86*, pages 377–382, 1986.