Annals of Mathematics and Artificial Intelligence **36:** 5–38, 2002.

An algebraic representation of calendars

Peng Ning [a], Xiaoyang Sean Wang [b] and Sushil Jajodia [b]

[a] *Department of Computer Science, North Carolina State University, Raleigh, NC 27695-7534, USA*
E-mail: ning@csc.ncsu.edu
[b] *Department of Information & Software Engineering, George Mason University, 4400 University Drive, Fairfax, VA 22030-4444, USA*
E-mail: {xywang, jajodia}@gmu.edu

This paper uses an algebraic approach to define temporal granularities and calendars. All the granularities in a calendar are expressed as algebraic expressions based on a single "bottom" granularity. The operations used in the algebra directly reflect the ways with which people construct new granularities from existing ones, and hence yield more natural and compact granularities definitions. Calendar is formalized on the basis of the algebraic operations, and properties of calendars are studied. As a step towards practical applications, the paper also presents algorithms for granule conversions between granularities in a calendar.

Keywords: temporal granularity, calendar, calendar algebra, granule conversion

AMS subject classification: 68T30, 68U35, 68W30

1. Introduction

System support for time has long been recognized to be important. Time is often represented in terms of closely related granularities (e.g., `year`, `month`, `day`) that are organized into calendars (e.g., Gregorian calendar). Reasoning and processing of time are usually performed on these representations. The reasoning processes usually require knowledge not only about the involved granularities but also the relationships among them. Moreover, for applications that allow user-defined granularities and calendars (e.g., scheduling applications), it is critical for the representation mechanisms to be natural and flexible as well. This paper presents such a representation mechanism that accommodates these requirements.

Natural representation is important not only for the ease of use. In many cases, it also allows more compact representations. As an example, consider the specification of leap years. A year is a leap one if the year (i.e., its number) is divisible by 4, but not divisible by 100 unless it's divisible by 400. A direct method of "coding" the leap year information is to have the above rule embedded in the definition of the granularity `year`. Unfortunately, it seems that all current proposals of granularity symbolic representations adopt an "explicit" method, namely give a separate definition to each year in a 400-year period. Using such a method, the system cannot easily "rediscover" the leap-year rule and take advantage of it in a reasoning system. Moreover, such a method is not scalable

to granularities with large periods. In particular, the enumeration will take more storage, and manipulating large periods may result in poor performance.

In this paper, we develop an algebraic representation for time granularities, which we call the *calendar algebra*. Each time granularity is defined as a mapping from its index set to the subsets of the time domain [1]. We assume that there exists a "bottom" granularity known to the system. Calendar algebra operations are designed to generate new granularities from the bottom one or recursively, from those already generated. Thus, the relationship between the operand(s) and the resulting granularities are encoded in the operations.

The central issue to be resolved is to design a set of operations that capture the characteristics of calendars both naturally and expressively. On the basis of how the human calendars are usually formed, we come up with a set of operations that conform to the human perspective of granularities. For example, granularity `month` can be generated on the basis of granularity `day` by several calendar algebra operations. The first operation generates a granularity by partitioning all the days into 31-day groups, the second operation shrinks the second group of every 12 groups (which corresponds to February) by 3 days, the third step shrinks the fourth group of every 12 ones (which corresponds to April) by 1 day, etc. To define `month` on the basis of `day` including all the leap year information, we only need nine such operations (see section 3 for details) without explicit enumeration of all the months in a period of 400 years (i.e., 4,800 months).

We formalize the notion of calendar on the basis of granularities generated by calendar algebra. Informally, a calendar is a collection of granularities generated from a single bottom granularity and the ways in which the non-bottom granularities are generated. For example, we may have a calendar that has two granularities `second` and `minute`, where `second` is the bottom granularity and `minute` is generated by grouping every 60 seconds. We study the high-level relationships, especially the dependency, between the granularities in the same calendar. It turns out that the relationships among the granularities in the same calendar have some nice properties.

The process of finding some granules in one granularity that has a particular relationship with a set of given granules in another granularity is called *granule conversion*. An example is to find all the business days in a given month. Granule conversion is essential to many applications such as automatic evaluation of user queries, mixed granularities and multiple calendars support, and rolling up along a time hierarchy in time series analysis or OLAP applications. We develop a generic method to solve the general granule conversion problem.

The above mapping viewpoint of granularity represents granules using indices, e.g., integers. However, people are used to relative representations. For example, a particular day is usually represented in the form of, for example, *January 3*, *2001*, which denotes the 3rd day in the first month in year 2001. To formalize such representations, we develop label mappings on the basis of granule conversions. Then the aforementioned day *January 3*, *2001* can be represented as a label (2001, 1, 3) on the given set of granularities `year`, `month` and `day`.

The relative representation and granule conversion combined give rise to some interesting computation capabilities. For example, to find the week day of *January 3, 2001*, we only need to convert the day to a relative representation $(x, 4)$, where x is the index of the week of *January 3, 2001*, based on the two granularities `week` and `day`. Indeed, since it is the 4th day of week x, it is a Wednesday (counting from Sunday as usually done in the US). Other examples include finding the date of the first Monday of September 2001 and finding the week day that is the 200th day of year 2001.

The rest of this paper is organized as follows. Section 2 defines some preliminary concepts that will be used throughout this paper. Section 3 presents the algebraic operations. Section 4 formalizes the notion of calendar on the basis of the algebraic operations and studies its properties. Section 5 discusses granule conversion between granularities within a calendar, and presents the notion of vector label, which provides a relative representation for time granules. Section 6 discusses the related work, and section 7 concludes the paper.

2. Preliminaries

We adopt some notions about temporal granularity from [1,2].

Definition 1. A *time domain* is a pair $(T, \leqslant)$, where T is a non-empty set of *time instants* and $\leqslant$ is a total order on T.

A time domain is the set of primitive temporal entities used to define and interpret time-related concepts. The temporal entities in the set are ordered by a relationship, $\leqslant$, on these entities. Integers $(\mathbb{Z}, \leqslant)$, natural numbers $(\mathbb{N}, \leqslant)$, rational $(\mathbb{Q}, \leqslant)$, and real numbers $(\mathbb{R}, \leqslant)$ are all examples of time domains with their natural total orders. In this paper, we assume that there is a fixed time domain without loss of generality.

Definition 2. A *granularity* is a mapping G from the integers (the *index set*) to the subsets of the time domain such that (1) if $i < j$ and $G(i)$ and $G(j)$ are non-empty, then each element of $G(i)$ is less than all the elements of $G(j)$, and (2) if $i < k < j$ and $G(i)$ and $G(j)$ are non-empty, then $G(k)$ is non-empty.

Each non-empty set $G(i)$ is called a *granule* of granularity G.

The first condition states that the time instants of different granules do not interleave and their index order is the same as their time domain order, i.e., the mapping must be *monotonic*. The second condition disallows an empty set to be the value of a mapping for an index value if both a lower index and a higher index are mapped to nonempty sets.

To simplify the algebra, we use an extended notion of granularity.

Definition 3. A *labeled granularity* is a pair $(\mathcal{L}, G)$, where $\mathcal{L}$ is a subset of the integers, and G is a mapping from $\mathcal{L}$ to the subsets of the time domain such that (1) if $i < j$, where i and j are in $\mathcal{L}$, and $G(i)$ and $G(j)$ are non-empty, then each element in $G(i)$ is

less than all the elements of $G(j)$, and (2) if $i < k < j$, where i, j, and k are in $\mathcal{L}$, and $G(i)$ and $G(j)$ are non-empty, then $G(k)$ is non-empty. $\mathcal{L}$ is called the *label set* of G.

When $\mathcal{L}$ is exactly the integers, we call the granularity *full-integer labeled*. Note that full-integer labeled granularities are exactly "regular" granularities defined in definition 2 since labels coincide with indices. However, in general, the label set $\mathcal{L}$ can be an arbitrary subset of (possibly noncontiguous) integers and these labels are used instead of indices to identify granules.

The reason that we allow the label set to be possibly noncontiguous is due to the need for computational efficiency. For example, to union two granularities (e.g., `Sunday` and `Saturday`, see section 3), we have to make sure the granules of the two operand granularities are properly aligned in the resulting granularity. It is much easier to have the labels of the operand granularities already aligned than to reorder the granules from both granularities and reassign the labels. This implies that we cannot always have integers as the label sets of granularities. For example, if both `Sunday` and `Saturday` have integers as the label sets, then we will have to reassign the labels to the granules of the union of `Sunday` and `Saturday` since the granules of these two granularities actually interleave with each other. Nevertheless, using noncontiguous integers as the label set may be unintuitive and confuse human users. We address this problem by introducing a relative representation mechanism called *vector labels* in section 5.6.

We will still use G to denote labeled granularities and refer to labeled granularities as granularities when no confusion arises.

2.1. Relationships between labeled granularities

To facilitate the description of calendar algebra operations, we now extend some relationships between granularities that have the same time domain to labeled granularities. For detailed information about the original relationships, please refer to [1,2].

Group into A labeled granularity G *groups into* a labeled granularity H if for each label i of H, there exists a subset S of the label set of G such that $H(i) = \bigcup_{j \in S} G(j)$. For example, each week consists of seven days, so granularity `day` groups into granularity `week`.

Finer than A labeled granularity G is *finer than* a labeled granularity H if for each label i of G, there exists a label j of H such that $G(i) \subseteq H(j)$. For example, every day is in a certain month, so `day` is finer than `month`.

Subgranularity A labeled granularity G is a *subgranularity* of a labeled granularity H if for each label i of G, there exists a label j of H such that $G(i) = H(j)$. For example, `Sunday` is a subgranularity of `day`. Note that G is a subgranularity of H if and only if H groups into G and G is finer than H.

Label-aligned subgranularity A labeled granularity G is a label-aligned subgranularity of a labeled granularity H if for each label i of G, i is also a label of H and $G(i) = H(i)$. For instance, if each Sunday has the same label as the corresponding day, then `Sunday` is a label-aligned subgranularity of `day`.

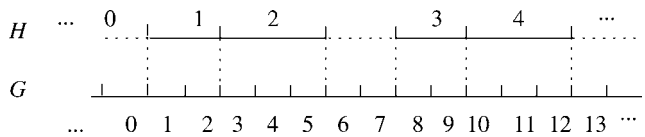


Figure 1. The *groups periodically into* relationship.

Note that the difference between the label-aligned subgranularity relationship and the "usual" subgranularity relationship is that here each granule of G_1 uses exactly the same label as the one used by the corresponding granule in G_2.

Shift equivalent Full-integer labeled granularities G and H are shift equivalent if there exists an integer k such that $G(i) = H(i + k)$ for all $i \in \mathbb{Z}$. For example, if each hour of `GMT-Hour` has a label that is larger than the corresponding hour of `USEastTime-Hour` by 5, then `GMT-Hour` and `USEastTime-Hour` are shift equivalent.

Partition A labeled granularity G *partitions* a labeled granularity H if G groups into H and G is finer than H. For example, `day` partitions `week`, since `day` both groups into and is finer than `week`.

Groups periodically into A labeled granularity G *groups periodically into* a labeled granularity H if G groups into H and there exist positive integers R, P, where R is less than the number of granules of H, such that (1) for each label i of H, $i + R$ is a label of H if there is a label of H that is greater than or equal to $i + R$, and (2) for each label i of H, if $H(i) = \bigcup_{r=0}^{k} G(j_r)$ and $H(i + R)$ is a non-empty granule of H then $H(i + R) = \bigcup_{r=0}^{k} G(j_r + P)$. Figure 1 shows two such granularities G and H, where G groups periodically into H with P and R being 7 and 2, respectively.

3. Calendar algebra

In this section, we present a symbolic representation of granularities. The design of the representation scheme starts with the observation that granularities used in a calendar are not isolated but closely related. We thus design our symbolic representation based on some algebraic operations that capture these relationships. The symbolic representation is called the *calendar algebra*.

The calendar algebra consists of two kinds of operations: *grouping-oriented* and *granule-oriented operations*. The grouping-oriented operations combine certain granules of a granularity together to form the granules of the new granularity, while the granule-oriented operations do not change the granules of a granularity, but rather make choices of which granules should remain in the new granularity. Certain calendar operations will only work on full-integer labeled granularities, while others will be more easily defined and implemented using more flexible labeling as in a labeled granularity.

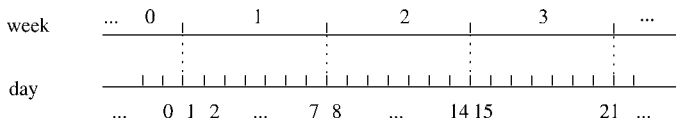


Figure 2. Grouping operation.

3.1. The grouping-oriented operations

3.1.1. The grouping operation

Let G be a full-integer labeled granularity, and m a positive integer. The grouping operation $Group_m(G)$ generates a new granularity G' by partitioning the granules of G into m-granule groups and making each group a granule of the resulting granularity. More precisely, $G' = Group_m(G)$ is the full-integer labeled granularity such that for each integer i, $G'(i) = \bigcup_{j=(i-1)m+1}^{im} G(j)$.

Figure 2 shows an example of the grouping operation, where granularity `week` is defined by `week` $= Group_7($`day`$)$. Note here we assume that the day labeled 1 starts a week.

3.1.2. The altering-tick operation

Let G_1, G_2 be full-integer labeled granularities, and l, k, m integers, where G_2 partitions G_1, and $1 \leqslant l \leqslant m$. The altering-tick operation $Alter^m_{l,k}(G_2, G_1)$ generates a new full-integer labeled granularity by periodically expanding or shrinking granules of G_1 in terms of granules of G_2. Since G_2 partitions G_1, each granule of G_1 consists of some contiguous granules of G_2. The granules of G_1 can be partitioned into m-granule groups such that $G_1(1)$ to $G_1(m)$ are in one group, $G_1(m+1)$ to $G_1(2m)$ are in the following group, and so on. The altering-tick operation modifies the granules of G_1 so that the lth granule of each group has $|k|$ additional (or fewer when $k < 0$) granules of G_2. For example, if G_1 represents 30-day groups (i.e., $G_1 = Group_{30}($`day`$)$) and we want to add a day (i.e., $k = 1$) to the 5th one in each group of 12 months (i.e., $l = 5$ and $m = 12$), we may have $Alter^{12}_{5,1}($`day`$, G_1)$. Intuitively, this makes *May* have 31 days.

More specifically, for all $i = l + mn$, where n is an integer, $G_1(i)$ denotes the granule to be shrunk or expanded. The granules of G_1 are split into two parts at $G_1(0)$. When $i > 0$, $G_1(i)$ expands (or shrinks if $k < 0$) by taking in (or pushing out) later granules of G_2, and the effect is propagated to later granules of G_1. When $i \leqslant 0$, $G_1(i)$ expands (or shrinks if $k < 0$) by taking in (or pushing out) earlier granules of G_2, and the effect is propagated to earlier granules of G_1.

The altering-tick operation can be formally described as follows. For each integer i such that $G_1(i) \neq \emptyset$, let b_i and t_i be the integers such that $G_1(i) = \bigcup_{j=b_i}^{t_i} G_2(j)$. (The integers b_i and t_i exist because G_2 partitions G_1.) Then $G' = Alter^m_{l,k}(G_2, G_1)$ is the granularity such that for each integer i, let $G'(i) = \emptyset$ if $G_1(i) = \emptyset$, and otherwise let

$$G'(i) = \bigcup_{j=b'_i}^{t'_i} G_2(j),$$

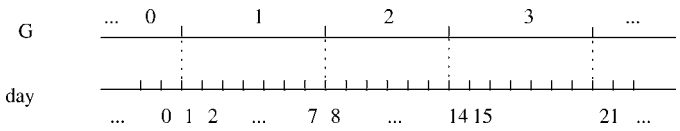


Figure 3. Altering-tick operation.

where

$$b'_i = \begin{cases} b_i + (h-1)k, & \text{if } i = (h-1)m + l, \\ b_i + hk, & \text{otherwise,} \end{cases}$$
$$t'_i = t_i + hk,$$

and

$$h = \left\lfloor \frac{i-l}{m} \right\rfloor + 1.$$

Figure 3 shows an example of an altering-tick operation. Suppose we want to have a granularity called G that has six days in the second granule in every three-granule group (starting from the second one) and has seven days in the other granules. That is, $G(1)$ has 7 days, $G(2)$ has 6 days, and then $G(3)$ has 7 days, and so on. Then G can be defined by $G = \mathit{Alter}^3_{2,-1}(\texttt{day}, \texttt{week})$.

Note that the grouping operation is a special case of the altering-tick operation. Indeed, $\mathit{Group}_m(G) = \mathit{Alter}^1_{1,m-1}(G, G)$, i.e., combining every m granules together is the same as expanding every granule by $m-1$ granules. However, we keep the grouping operation because of its conceptual simplicity.

An extension of the operation is also used: when the parameter m is infinity (∞), the altering-tick operation $\mathit{Alter}^\infty_{l,k}(G_2, G_1)$ means only altering the granule $G_1(l)$. For example, to add a leap second to the last minute of 1998, we may use $\mathit{Alter}^\infty_{x,1}(\texttt{second}, \texttt{minute})$, where x is the label of the last minute of 1998.

3.1.3. Shifting operation

Let G be a full-integer labeled granularity, and m an integer. The shifting operation $\mathit{Shift}_m(G)$ generates a new full-integer labeled granularity G' by shifting the labels of G by m positions. For each integer i, the granule $G'(i)$ has the same time instants as granule $G(i-m)$. More formally, $G' = \mathit{Shift}_m(G)$ is the granularity such that for each integer i, $G'(i) = G(i-m)$. Note that G is shift equivalent to G'.

The shifting operation can easily model time differences. Suppose granularity USEast-Hour stands for the hours of US Eastern Time. Since the hours of US Pacific Time are 3 hours later than those of US Eastern Time, the hours of US Pacific Time can be generated by $\texttt{USPacific-Hour} = \mathit{Shift}_{-3}(\texttt{USEast-Hour})$.

Note. The grouping, altering-tick and shifting operations are collectively called *basic operations*. These basic operations are restricted to operate on full-integer labeled gran-

ularities (i.e., "regular" granularities), and the granularities generated by these operations are still full-integer labeled ones.

3.1.4. Combining operation

Let G_1 and G_2 be granularities with label sets $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. The combining operation $Combine(G_1, G_2)$ generates a new granularity G' by combining all the granules of G_2 that are included in one granule of G_1 into one granule of G'. More formally, for each $i \in \mathcal{L}_1$, let $s(i) = \emptyset$ if $G_1(i) = \emptyset$, and otherwise let $s(i) = \{j \in \mathcal{L}_2 \mid \emptyset \neq G_2(j) \subseteq G_1(i)\}$. Then $G' = Combine(G_1, G_2)$ is the granularity with the label set $\mathcal{L}_{G'} = \{i \in \mathcal{L}_1 \mid s(i) \neq \emptyset\}$ such that for each i in $\mathcal{L}_{G'}$, $G'(i) = \bigcup_{j \in s(i)} G_2(j)$.

For instance, given granularities `business-day` (which represents the business days) and `month`, the granularity for business months can be generated by `business-month` = *Combine*(`month`, `business-day`). That is, each business month is a union of all the business days in the month.

3.1.5. Anchored grouping operation

Anchored grouping operation is a variation of the grouping operation. Let G_1 and G_2 be granularities with label sets $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, where G_1 is a full-integer labeled granularity, and G_2 is a label-aligned subgranularity of G_1. The anchored grouping operation $Anchored\text{-}group(G_1, G_2)$ generates a new granularity G' by combining all the granules of G_1 that are between two granules of G_2 into one granule of G'. More precisely, $G' = Anchored\text{-}group(G_1, G_2)$ is the granularity with the label set $\mathcal{L}_{G'} = \mathcal{L}_2$ such that for each $i \in \mathcal{L}_{G'}$, $G'(i) = \bigcup_{j=i}^{i'-1} G_1(j)$, where i' is the next label of G_2 after i.

Granularity G_2 is called the *anchor granularity* of G_1 in this operation. The granules of G_2 divide the granules of G_1 into groups, and each group is made into a granule of the resulting granularity (by the anchored grouping operation). For example, suppose each fiscal year at a company begins in October and ends in the next September. Then the granularity corresponding to the fiscal years can be generated by `FiscalYear` = *Anchored-group*(`month`, `October`).

3.2. Granule-oriented operations

3.2.1. Subset operation

The subset operation is designed to generate a new granularity by selecting an interval of granules from another granularity.

Let G be a granularity with label set $\mathcal{L}$, and m, n integers such that $m \leqslant n$. The subset operation $G' = Subset_m^n(G)$ generates a new granularity G' by taking all the granules of G whose labels are between m and n. More formally, $G' = Subset_m^n(G)$ is the granularity with the label set $\mathcal{L}_{G'} = \{i \in \mathcal{L} \mid m \leqslant i \leqslant n\}$, and for each $i \in \mathcal{L}_{G'}$, $G'(i) = G(i)$. For example, given granularity `year`, all the years after the 20th century can be generated by `FutureYear` = $Subset_{2001}^{\infty}$(`year`).

Note that G' is a label-aligned subgranularity of G, and G' is not a full-integer labeled granularity even if G is. We also allow in the above the extensions of setting $m = -\infty$ or $n = \infty$ with semantics extended in a usual way.

3.2.2. Selecting operations

The selecting operations are all binary operations. They generate new granularities by selecting granules from the first operand in terms of their relationship with the granules of the second operand. The result is always a label-aligned subgranularity of the first operand granularity.

There are three selecting operations: *select-down, select-up* and *select-by-intersect*. To facilitate the description of these operations, we introduce a notation for subsets of a given set of integers. Suppose S is a set of n integers. Let $S = \{j_1, j_2, \ldots, j_n\}$, where $j_1 < j_2 < \cdots < j_n$. For each $i \leqslant 0$, let j_i be an arbitrary integer less than j_1, and for each $i > n$, let j_i be an arbitrary integer greater than j_n. Given two integers k and l, where $k \neq 0$ and $l > 0$, $\Delta_k^l(S)$ denotes the subset of S defined as follows.

$$\Delta_k^l(S) = \begin{cases} S \cap \{j_k, \ldots, j_{k+l-1}\}, & \text{if } k > 0, \\ S \cap \{j_{(n+k+2)-1}, \ldots, j_{(n+k+2)-l}\}, & \text{if } k < 0. \end{cases}$$

Therefore, if $k > 0$, $\Delta_k^l(S)$ consists of the l (or less than l if the range determined by k and l is out of S) integers in S starting from the kth one from the beginning of the list. For example, $\Delta_3^2(\{1, 2, 3, 4, 5, 6, 7\}) = \{3, 4\}$, since it should include two labels ($l = 2$) starting from the third label ($k = 3$). If $k < 0$, $\Delta_k^l(S)$ consists of the l (or less than l if the range determined by k and l is out of S) integers in S starting from the $|k|$th one from the end of the list counting backward. For example, $\Delta_{-7}^3(\{1, 2, 3, 4, 5, 6, 7\}) = \{1, 2, 3\}$, since it should consist of three labels ($l = 3$) starting from the seven-th label ($k = -7$) from the end.

Let G_1 and G_2 be granularities with label sets $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. In the following, we describe the selecting operations using the $\Delta_k^l(\cdot)$ operator.

Select-down operation. For each granule $G_2(i)$, there may exist a set of granules of G_1 contained in $G_2(i)$. The operation *Select-down*$_k^l(G_1, G_2)$, where $k \neq 0$ and $l > 0$ are integers, selects granules of G_1 by picking up l granules starting from the kth one in each set of granules of G_1 contained in one granule of G_2. More formally, $G' =$ *Select-down*$_k^l(G_1, G_2)$ is the granularity with the label set $\mathcal{L}_{G'} = \bigcup_{i \in \mathcal{L}_2} \Delta_k^l(\{j \in \mathcal{L}_1 \mid \emptyset \neq G_1(j) \subseteq G_2(i)\})$, and for each $i \in \mathcal{L}_{G'}$, $G'(i) = G_1(i)$. For example, Sunday is considered the first day in a week in the United States. Then given granularities `week` and `day`, the granularity for the Sundays can be generated by `Sunday` = *Select-down*$_1^1$(`day`, `week`). Note that G' is a label-aligned subgranularity of G_1.

Select-up operation. The select-up operation *Select-up*(G_1, G_2) generates a new granularity G' by selecting the granules of G_1 that contain one or more granules of G_2. More formally, $G' =$ *Select-up*(G_1, G_2) is the granularity with the label

set $\mathcal{L}_{G'} = \{i \in \mathcal{L}_1 \mid \exists j \in \mathcal{L}_2(\emptyset \neq G_2(j) \subseteq G_1(i))\}$, and for each $i \in \mathcal{L}_{G'}$, $G'(i) = G_1(i)$. For example, given granularities `week` and `FirstDayOfMonth`, the granularity of the first week of each month can be defined by `FirstWeekOfMonth` = *Select-up*(`week`, `FirstDayOfMonth`). Note that G' is a label-aligned subgranularity of G_1.

Select-by-intersect operation. For each granule $G_2(i)$, there may exist a set of granules of G_1, each granule intersecting $G_2(i)$. The operation *Select-by-intersect*$^l_k(G_1, G_2)$, where $k \neq 0$ and $l > 0$ are integers, selects granules of G_1 by selecting l granules starting from the kth one in all such sets, generating a new granularity G'. More formally, G' = *Select-by-intersect*$^l_k(G_1, G_2)$ is the granularity with the label set $\mathcal{L}_{G'} = \bigcup_{i \in \mathcal{L}_2} \Delta^l_k(\{j \in \mathcal{L}_1 \mid G_1(j) \cap G_2(i) \neq \emptyset\})$, and for each $i \in \mathcal{L}_{G'}$, $G'(i) = G_1(i)$. For example, given the granularities `week` and `Semester`, the granularity consisting of the last week of each semester (among all the weeks intersecting the semester) can be generated by `LastWeekOfSemester` = *Select-by-intersect*$^1_{-1}$(`week`, `Semester`). Again, G' is a label-aligned subgranularity of G_1.

3.2.3. Set operations

The set operations are based on the viewpoint that each granularity corresponds to a set of granules mapped from the labels. In order to have the set operations as a part of the calendar algebra and to make certain computations easier, we restrict the operand granularities participating in the set operations so that the result of the operation is always a valid granularity: *the set operations are defined on G_1 and G_2 only if there exists a granularity H such that G_1 and G_2 are both label-aligned subgranularities of H*. In the following, we describe the union, intersection and difference operations of G_1 and G_2, assuming that they satisfy the requirement.

Union. The union operation $G_1 \cup G_2$ generates a new granularity G' by collecting all the granules from both G_1 and G_2. More formally, $G' = G_1 \cup G_2$ is the granularity with the label set $\mathcal{L}' = \mathcal{L}_1 \cup \mathcal{L}_2$, and for each $i \in \mathcal{L}_{G'}$,

$$G'(i) = \begin{cases} G_1(i), & i \in \mathcal{L}_1, \\ G_2(i), & i \in \mathcal{L}_2 - \mathcal{L}_1. \end{cases}$$

For example, given granularities `Sunday` and `Saturday`, the granularity of the weekend days can be generated by `WeekendDay` = `Sunday` ∪ `Saturday`. Note that G_1 and G_2 are label-aligned subgranularities of G'. In addition, if G_1 and G_2 are label-aligned subgranularity of H, then G' is also a label-aligned subgranularity of H. This is derived from the transitivity of the label-aligned subgranularity relationship.

Intersection. The intersection operation $G_1 \cap G_2$ generates a new granularity G' by taking the common granules from both G_1 and G_2. More formally, $G' = G_1 \cap G_2$ is the granularity with the label set $\mathcal{L}' = \mathcal{L}_1 \cap \mathcal{L}_2$, and for each $i \in \mathcal{L}'$, $G'(i) = G_1(i)$ (or equivalently $G_2(i)$). For example, given the granularity `FullMoonDay`, which includes the days when there is a full moon, and the granularity `Weekday`, the

granularity for the days that are both full-moon days and weekdays can be generated by $\texttt{FullMoonWeekDay} = \texttt{FullMoonDay} \cap \texttt{Weekday}$. Note that G' is a label-aligned subgranularity of both G_1 and G_2.

Difference. The difference operation $G_1 - G_2$ generates a new granularity G' by excluding the granules of G_2 from those of G_1. More formally, $G' = G_1 - G_2$ is the granularity with the label set $\mathcal{L}' = \mathcal{L}_1 - \mathcal{L}_2$, and for each $i \in \mathcal{L}'$, $G'(i) = G_1(i)$. For example, business days are all the week days that are not federal holidays. Then given granularities `Weekday` and `FederalHoliday`, `BusinessDay` can be generated by $\texttt{BusinessDay} = \texttt{Weekday} - \texttt{FederalHoliday}$. Note that G' is a label-aligned subgranularity of G_1.

From the definition of the calendar algebraic operations, it is clear that the operations are defined by manipulating the granules of the operand granularities. In addition, we have the following lemma on the basis of the definition of the calendar algebraic operations.

Lemma 1. Each granule of a granularity generated by a calendar algebraic operation consists of one or many granules of at least one operand granularity.

3.3. Examples

Here we present some more example granularities represented in the calendar algebra. Assuming that `second` is given, we may have the following.

- $\texttt{minute} = \mathit{Group}_{60}(\texttt{second})$,
- $\texttt{hour} = \mathit{Group}_{60}(\texttt{minute})$,
- $\texttt{day} = \mathit{Group}_{24}(\texttt{hour})$,
- $\texttt{week} = \mathit{Group}_{7}(\texttt{day})$,
- $\texttt{pseudomonth} = \mathit{Alter}^{12}_{11,-1}(\texttt{day}, \mathit{Alter}^{12}_{9,-1}(\texttt{day}, \mathit{Alter}^{12}_{6,-1}(\texttt{day}, \mathit{Alter}^{12}_{4,-1}(\texttt{day}, \mathit{Alter}^{12}_{2,-3}(\texttt{day}, \mathit{Group}_{31}(\texttt{day}))))))$, where the granularity `pseudomonth` is generated by grouping 31 days, and then shrink April (4), June (6), September (9) and November (11) by 1 day, and shrink February (2) by 3 days,
- $\texttt{month} = \mathit{Alter}^{12*400}_{2+12*399,1}(\texttt{day}, \mathit{Alter}^{12*100}_{2+12*99,-1}(\texttt{day}, \mathit{Alter}^{12*4}_{2+12*3,1}(\texttt{day}, \texttt{pseudomonth})))$, where the February of each leap year is adjusted appropriately,
- $\texttt{Monday} = \mathit{Select\text{-}down}^{1}_{1}(\texttt{day}, \texttt{week})$,
- ...,
- $\texttt{Sunday} = \mathit{Select\text{-}down}^{1}_{7}(\texttt{day}, \texttt{week})$.

In the above examples, we assumed that `second(1)` starts a minute, `minute(1)` starts an hour, etc. Note that the granularity `pseudomonth` was first defined to represent the months in nonleap years, and then `month` was generated by taking into consideration the leap years based on `pseudomonth`.

Informally, we call a calendar algebraic operation or a composition of several operations with the operand granularity(ies) a *calendar algebraic expression*, or simply a *calendar expression*. In the above examples, the granularity `hour` is defined by a calendar algebraic expression $Group_{60}$(`minute`), which consists of only one grouping operation, and the granularity `month` is defined by a more complex calendar algebraic expression with three altering-tick operations.

3.4. Further assumption and expressiveness

By the definition of the operations, the granularities participating in a calendar operation may have to satisfy certain conditions. For example, the set operations only apply to granularities that are label-aligned subgranularities of a common one. In general, checking these preconditions is infeasible. For example, given two granularities G_1 and G_2 over the same time domain, to determine whether G_1 groups into G_2 is to check whether there exists a set S of granules of G_1 for each granule $G_2(i)$ such that $G_2(i) = \sum_{j \in S} G_1(j)$, and it is impossible if G_1 and G_2 are arbitrary granularities that have infinite number of granules.

To make the check of preconditions feasible, we assume that there is only one predefined, full-integer labeled granularity called the *bottom granularity* and all the other granularities are generated from the bottom granularity by calendar algebraic operations. (Indeed, this is how human calendars are formed. There is usually one finest granularity (e.g., `day`, `second`, or `nanosecond`), on the basis of which all the other granularities are formed.)

Note that the bottom granularity does not have to cover the entire time domain; it may have gaps within and/or between its granules. Since our algebraic operations are defined by manipulating the labels of the operand granularities, all the operations are still well defined. For example, we may use the granularity of `WeekDays` as the bottom granularity, assuming `WeekDays` is full-integer labeled. Then we may define two new granularities by $G = Group_5$(`WeekDays`) and $G' = Alter^2_{1,-1}$(`WeekDays`, G). Figure 4 shows the granularities `WeekDays`, G and G', where the dotted segments represent the part of the time domain not covered by the bottom granularity. Note that $G'(1)$ consists of `WeekDays`(1) through `WeekDays`(4), $G'(2)$ consists of `WeekDays`(5) through `WeekDays`(9), and so on. Both G and G' are full-integer labeled.

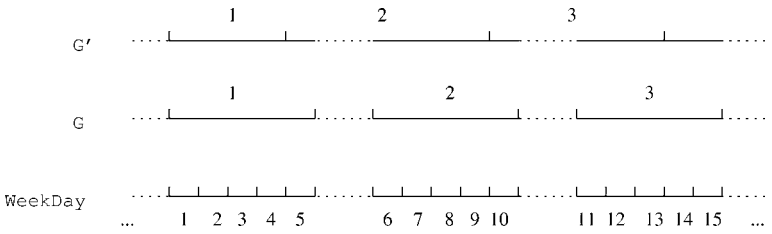


Figure 4. Using a bottom granularity that does not cover the entire time domain.

Also note that we do not specify the bottom granularity, but only require it to be full-integer labeled. For example, the bottom granularity can be one that consists of granules with equal size (e.g., `second`, `minute`), and it can also be one that have granules with different sizes (e.g., `month`). Although the choice of the bottom granularity affects the granularities generated by calendar algebra, it does not affect the relationships between the generated granularities and itself. Indeed, many temporal applications (e.g., temporal databases) only care about the relationships between different granularities.

To study the expressiveness of the calendar algebra, we define the concepts of periodical granularity and finite granularity.

Definition 4. A granularity G is said to be *periodical* (with respect to the bottom granularity B) if B groups periodically into G.

Assume $B =$ `day`. It is easily seen that `week` is periodical since `day` groups periodically into `week` (with $R = 1$ and $P = 7$). Furthermore, `year` is also periodical since `day` groups periodically into `year` (with $R = 400$ and P being the number of days in a four-hundred year period).

Definition 5. A granularity G is said to be *finite* (with respect to the bottom granularity B) if B groups into G and the number of G's granules is finite.

Theorem 1. The calendar algebra can represent all the granularities that are either periodical or finite (with respect to the bottom granularity).

Proof. Assume that the bottom granularity B groups into all the granularities being considered. Consider an infinite periodical granularity G that has no first nor last granule. It is clear that there exist positive integers R and P, where R is less than the number of granules of G, such that (1) for each label i of G, $i + R$ is a label of G, and (2) for each label i of G, if $G(i) = \bigcup_{r=0}^{k} B(j_r)$, then $G(i + R) = \bigcup_{r=0}^{k} B(j_r + P)$. We divide the granules of B into P-granule periods. Then it is easily seen that *Select-down*$_i^1$(`bottom`, *Group*$_P$(`bottom`)) gives a granularity that consists of the ith granule of the bottom granularity for each period. Let S be the (finite) set of granules of the bottom granularity that are in the first period and also contained in granules of G. By using the above expression repeated by a finite number of times (once for each integer in S) and using the union operation (note that each above granularity is a label-aligned subgranularity of the bottom), we can get a granularity that is a subgranularity of the bottom whose granules are exactly those that group into the granules of G. Call this granularity H. Now assume that G' is the subgranularity of the bottom such that each granule of G starts exactly with a granule of G'. It is not difficult to see that G' is also periodical and can be built by using the *Group, Select-down*, and *union* operations from B. Now it is easily seen that $G =$ *Anchored-group*(H, G').

Consider an infinite periodical granularity G that has a first granule. Let m be the label of the first granule. We can construct an infinite periodical granularity G'

by repeating the granules of G backward so that it is an infinite periodical granularity without first and last granule. According to the above discussion, G' can be represented by calendar algebraic operations. Thus, G can by defined by $G = \mathit{Subset}_m^{\infty}(G')$. An infinite periodical granularity having a last granule can be defined similarly.

Consider a finite granularity G. Let m and n be the smallest and the largest label of G, respectively. We can construct an infinite periodical granularity G' by repeating the granules of G both forward and backward. According to the above discussion, G' can be represented by calendar algebraic operations. Thus, G can be defined by $G = \mathit{Subset}_m^n(G')$. □

From the proof of theorem 1 we can see not all operations are required to represent all finite or periodical granularities. However, the goal of the calendar algebra is to provide a natural and flexible representation that can encode the relationships among different granularities. Each of the operations indeed captures one particular way in which a new granularity is generated from existing ones.

Note that the calendar algebra can express granularities that are not periodic nor finite due to the variation of the altering-tick operation, which may use ∞ as the period. For example, if we add a leap second into `minute` (defined in section 3.3) by `minute_with_leap_second` $= \mathit{Alter}_{x,1}^{\infty}(\texttt{second}, \texttt{minute})$, where x is the label of the minute into which the leap second is added, then `minute_with_leap_second` is neither periodic nor finite. An exact characterization is beyond the scope of this paper.

Remark. The calendar algebra has some properties that may facilitate reasoning about the generated granularities. As a simple example, consider $G_1 = \mathit{Group}_5(\mathit{Shift}_5(G))$ and $G_2 = \mathit{Shift}_1(\mathit{Group}_5(G))$. We can easily conclude that $G_1 = G_2$, i.e., they represent the same granularity. Thus, all the granularities that are finer than (or group into) G_1 are finer than (or group into) G_2. However, due to space reasons, full exploration of the algebraic equivalence of calendar expressions and its applications is out of the scope of this paper.

4. Calendar

4.1. Syntactic restrictions

Checking the preconditions could still be rather annoying with the assumption of the bottom granularity. For example, suppose we need to check whether G is full-integer labeled, where G is defined by $G = (\bigcup_{i=1}^{l_1} \mathit{Select\text{-}down}_{k_i}^1(\texttt{day}, \texttt{month})) \cup (\bigcup_{j=1}^{l_2} \mathit{Select\text{-}down}_{m_j}^1(\texttt{day}, \texttt{week}))$ on the basis of the granularities in section 3.3. In general, we only need to check the integers within a period of G with respect to `day`, since `day` groups periodically into G. However, we cannot decide whether an integer in the period is a valid label or not without using the exact values of k_i's and m_j's. Indeed,

we have to use these parameters to check all the integers in the whole period, since each integer in the period has the possibility of being either a valid or an invalid label.

To avoid the above situations, we further adopt some syntactic restrictions, namely to use the explicit relationships derived from the operations themselves, so that only the operators (not the parameters in the operators) need to be checked. As an additional benefit, the granule conversion problem (see section 5) is also simplified due to the syntactic restrictions.

Note that the preconditions of the operations only use the following kinds of requirements: (1) a granularity must be a full-integer labeled one, (2) a granularity must partition another one, and (3) a granularity is a label-aligned subgranularity of another.

We partition the granularities generated by the calendar algebra into three layers. Which layer a generated granularity is in is determined by the operations and the operands used to define the granularity. Figure 5 shows the three-layered partition of the granularities defined by the calendar algebra and the transitions between the layers resulting from calendar algebraic operations. *Layer* 1 consists of the bottom granularity and the granularities generated by only applying (maybe repeatedly) the basic operations (grouping, altering-tick and shifting). *Layer* 2 consists of the granularities that are the result of applying (maybe repeatedly) the subset operation and the selecting operations on the full-integer labeled granularities in the first layer. In addition to *subset* and selecting operations, set operations can be applied to the granularities in layer 2 that are label-aligned subgranularities of the same granularity in layer 1. The result will be in layer 2. *Layer* 3 consists of granularities that are the result of the combining and anchored grouping operations. Note that operand 1 for the anchored grouping operation must be from layer 1 (a full-integer labeled granularity), while the combine operation may take granularities of any layers.

The three-layer partition not only facilitates the calendar algebraic operations, but also leaves some interesting properties. All the granularities in layer 1 are full-integer

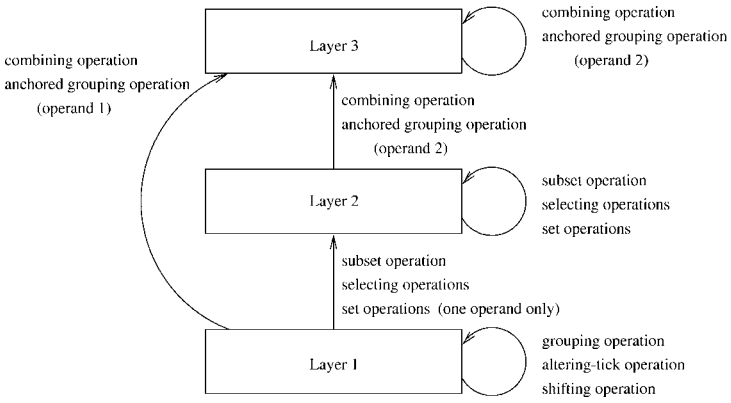


Figure 5. Three-layered partition of the granularities generated from one bottom granularity.

labeled. Such a granularity may have granules that have gaps in terms of the time domain when the bottom granularity has such gaps in its granules; however, there will be no gaps in terms of the granules of the bottom granularity. All granularities in layer 2 may not be full-integer labeled, but there is no gap within each granule of every granularity in terms of the bottom granularity, i.e., each granule is an interval of granules of the bottom granularity. The granularities in layer 3, however, may contain gaps (in terms of the bottom granularity) within a granule. Thus, from the layer in which a granularity is, we can infer its granule pattern in terms of the bottom granularity.

Note that the proof of theorem 1 follows the syntactic restrictions. This means that even with the syntactic restrictions, the calendar algebra can still represent all the granularities that are either periodical or finite.

4.2. Formalization of calendar

Intuitively, a calendar is a way to arrange correlated granularities. Here we abuse the word *calendar* a bit to denote both the set of granularities and how they are defined. A calendar can be considered being built up in a constructive manner. Initially, the calendar consists of only the bottom granularity. New granularities can be generated by the calendar algebra operations on the basis of granularities already in the calendar. Then the new granularities and the way they are defined (i.e., the calendar expression) are added into the calendar. The formal definition of calendar is as follows.

Definition 6. Let B be the bottom granularity. A *calendar* is recursively defined as follows:

- $(\{B\}, \{\})$ is a calendar;
- If (T, E) is a calendar, e is a calendar algebraic expression on the granularities in T, and $G \notin T$ is the granularity defined by e, then $(T \cup \{G\}, E \cup \{e\})$ is a calendar. We say e is the expression associated with G.

According to the above definition, all the calendars originate from the bottom granularity, and a new calendar is formed by adding into an existing calendar new granularities defined by calendar expressions. Each granularity in a calendar has an expression associated except for the bottom granularity.

Example. Let `second` be the bottom granularity. Then we may have a calendar $C_0 = (\{\texttt{second}\}, \{\})$. Given the calendar algebraic expression $\texttt{minute} = \mathit{Group}_{60}(\texttt{second})$, $C_1 = (\{\texttt{second}, \texttt{minute}\}, \{\texttt{minute} = \mathit{Group}_{60}(\texttt{second})\})$ is also a calendar. Similarly, the calendar $C = (T, E)$ can be constructed from C_1, where

$T = \{$`second, minute, hour, day, week, pseudomonth, month, year, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday`$\}$

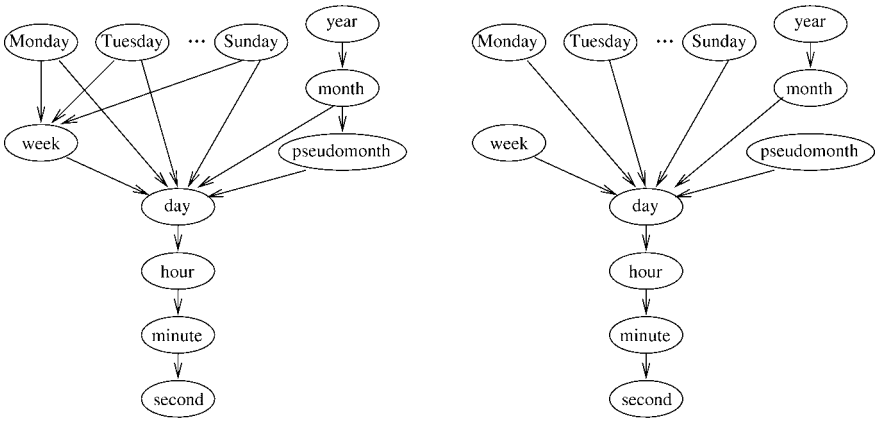
and E is given by the calendar expressions in section 3.3.

The granularities in a calendar are all related to each other by the calendar expressions that define them. The relationship between them can be pictorially demonstrated by a dependency graph.

Definition 7. The *dependency graph* for a calendar (T, E) is a directed graph, whose nodes are the granularities in T and for each pair of granularities G and G' in T, there is an edge from G to G' if and only if G' appears in the expression associated with G.

The dependency graph for a calendar shows what granularities are used to define other granularities. The nodes to which there is an edge from a granularity G represent all the granularities that appear in the expression associated with G. It follows directly from the basic graph theory that in a dependency graph for a calendar with the bottom granularity B, there exists a path from each granularity G (other than B) to B. This intuitively says that all granularities directly or indirectly depends on the bottom granularity. In addition, it also follows (from definitions 6 and 7) that the dependency graph is acyclic. Figure 6(a) shows the dependency graph for calendar C in the above example.

In the dependency graph for a calendar, all the edges from a granularity G point to all the granularities in the expression associated with G, and G can be defined by the expression associated with these granularities. For example, in figure 6(a), the edges from `Monday` point to `week` and `day`, and `Monday` is defined by the expression $\texttt{Monday} = \mathit{Select\text{-}down}_1^1(\texttt{day}, \texttt{week})$. Note that a granularity (e.g., `week`) in the expression associated with G has edges pointing to other granularities (e.g., `day`) if it is not the bottom granularity. By induction, it is easy to see that a granularity G can be defined by the granularities at the ends of all the paths and the composition of the expressions associated with the granularities along the paths. In particular, if all the paths from G finally come to a single granularity H, then G can be defined by H using the composition of the expressions along the paths. For example, in figure 6(a), all the



(a) The dependency graph (b) The defined-only-on relation

Figure 6. The dependency graph and the defined-only-on relation for calendar C.

paths from `Monday` come to `day`. Thus, `Monday` is defined by `day` and the expressions associated with `week` and `Monday`. This observation is captured by the following definition of *defined-only-on* relation.

Definition 8. Let (T, E) be a calendar with the bottom granularity B, and G_1 and G_2 two granularities in T. Then G_1 is said to be *defined-only-on* G_2 if G_2 is on all the paths from G_1 to B in the dependency graph.

Figure 6(b) shows the relation defined-only-on among the granularities defined in calendar C in the above example. (The transitive edges are omitted from the figure.) It is easy to see that in a calendar with the bottom granularity B, all the granularities G other than B are defined-only-on B.

Note that the syntactic restriction for the altering-tick operation actually requires that the second operand granularity is defined-only-on the first one.

The granularities in a calendar form a partial order with respect to the defined-only-on relation. Let (T, E) be a calendar with the bottom granularity B, and G_1, G_2 and G_3 any three granularities in the calendar. If G_1 is defined-only-on G_2, then, by definition, G_2 is in all the paths from G_1 to B in the dependency graph of the calendar. Then G_2 cannot be defined-only-on G_1. Indeed, otherwise, G_1 would be in all the paths from G_2 to B, and there would be a cycle involving G_1 and G_2. Thus, defined-only-on is anti-symmetric. If G_1 is defined-only-on G_2, and G_2 is defined-only-on G_3, then G_2 is in all the paths from G_1 to B, and G_3 is in all the paths from G_2 to B. This means G_3 is also in all the paths from G_1 to B, i.e., G_1 is defined-only-on G_3. Thus, defined-only-on is transitive.

Indeed, the relationship among the granularities in a calendar is more than just a partial order; it is a partial order with greatest lower bound. To see this, we first show that if G is defined-only-on both H_1 and H_2, then either H_1 is defined-only-on H_2, or H_2 is defined-only-on H_1. Since G is defined-only-on both H_1 and H_2, both H_1 and H_2 are in all the paths from G to B in the dependency graph. Then all the paths that contain H_1 and H_2 are either from H_1 to H_2, or from H_2 to H_1. Otherwise, there will be a cycle in the dependency graph. It then follows that either H_1 is defined-only-on H_2 or H_2 is defined-only-on H_1.

Now consider two different granularities G_1 and G_2 in the calendar. Since all the granularities in the calendar are defined-only-on B, G_1 and G_2 must have a lower bound with respect to defined-only-on relation. If G_1 and G_2 have more than one lower bounds, then let H_1 and H_2 be any two of them. It follows that G_1 is defined-only-on both H_1 and H_2, and either H_1 is defined-only-on H_2 or H_2 is defined-only-on H_1. That is, the lower bounds of G_1 and G_2 with respect to defined-only-on relation are comparable. Thus, the greatest lower bound of G_1 and G_2 always exists. This can be easily extended to the result stated in lemma 2.

Lemma 2. The greatest lower bound of any set of granularities with respect to the relation defined-only-on always exists in a calendar.

The following theorem further reveals the relationship between two granularities if one is defined-only-on the other in a calendar. This theorem and lemma 2 form the foundation of granule conversion to be discussed in the next section.

Theorem 2. Let G_1 and G_2 be any two granularities in a calendar. If G_1 is defined-only-on G_2, then G_2 groups into G_1.

Proof. By lemma 1, each granule of a resulting granularity consists of one or many granules of at least one operand granularity. Depicted in the dependency graph, each granule of a non-bottom granularity consists of one or many granules of at least one granularity to which the former one is connected by a directed edge. By induction, there is at least one path from a non-bottom granularity to the bottom one such that each granule of the non-bottom granularity consists of one or many granules of any granularity in the path. Since G_1 is defined-only-on G_2, G_2 is in every path from G_1 to G_2. This means that each granule of G_1 consists of one or many granules of G_2. Therefore, G_2 must group into G_1. □

5. Granule conversion

In order to process data measured in different granularities, systems should have the ability to convert the granules in one granularity to those in another. For example, suppose a database stores daily sales information. To get the sales data per business month, the database application has to have the information about which day is in which business month. We refer to the process of finding granules in one granularity in terms of the granules in another as *granule conversion*.

There can be many different semantics for granule conversions. In this section, we propose a generic conversion method on the basis of calendar algebra regardless of the semantics of the conversion. Our method is based on three basic constructs: *up conversion, down conversion*, and *next conversion* (which is a conversion within one granularity). General purpose granule conversions can be performed using the three basic conversions with further consideration of conversion semantics.

5.1. Three basic conversions

By theorem 2, one granularity groups into another in a calendar if the latter is defined-only-on the former. Hence, for each granule of the latter granularity, there exists a set of granules of the former such that both sets of granules cover the same part of the time domain. Since the greatest lower bound with respect to the defined-only-on relation (GLB) always exists (by lemma 2), granule conversion between two granularities can be performed with their GLB as an intermediary. (In the worst case, the bottom granularity will be this intermediary.) In other words, granule conversion between two granularities can be performed by granule conversions between each of them and their GLB. Thus, general purpose granule conversion can be reduced to granule conversions between two

granularities where one is defined-only-on the other and granule conversions within one granularity.

In the following, we present three basic granule conversions to support general purpose granule conversions.

Definition 9. Let G and H be granularities, where G is defined-only-on H. *Down conversion* from G to H, denoted $\lfloor\cdot\rfloor_H^G$, is a mapping from the label set of G to the subsets of the label set of H such that for each label i of G, the down conversion $\lfloor i \rfloor_H^G$ consists of all and only the labels of the granules of H that group into $G(i)$, i.e., $G(i) = \bigcup_{j \in \lfloor i \rfloor_H^G} H(j)$.

Definition 10. Let G, H be granularities, where G is defined-only-on H. *Up conversion* from H to G, denoted $\lceil\cdot\rceil_H^G$, is a mapping from the label set of H to the label set of G such that for each label i of H, if there exists a granule $G(j)$ that contains $H(i)$, then $\lceil i \rceil_H^G = j$; otherwise $\lceil i \rceil_H^G$ is undefined.

When granularity G is defined-only-on granularity H in a calendar, up and down conversion represent two directions of the conversions between G and H. Down conversion from G to H gets the labels of the granules of H that group into a certain granule of G, representing the *down* direction, while up conversion from H to G gets the label to the granule that contains a certain granule of H, representing the *up* direction.

Using the granularity examples given earlier, we can see that the down conversion from `month` to `day` is a mapping such that

$$\lfloor 1 \rfloor_{\texttt{day}}^{\texttt{month}} = \{1, 2, \ldots, 31\}, \quad \lfloor 2 \rfloor_{\texttt{day}}^{\texttt{month}} = \{32, 33, \ldots, 59\}, \quad \text{etc.}$$

The up conversion from `day` to `month` is a mapping such that

$$\lceil 1 \rceil_{\texttt{day}}^{\texttt{month}} = \lceil 2 \rceil_{\texttt{day}}^{\texttt{month}} = \ldots = \lceil 31 \rceil_{\texttt{day}}^{\texttt{month}} = 1,$$
$$\lceil 32 \rceil_{\texttt{day}}^{\texttt{month}} = \lceil 33 \rceil_{\texttt{day}}^{\texttt{month}} = \ldots = \lceil 59 \rceil_{\texttt{day}}^{\texttt{month}} = 2, \quad \text{etc.}$$

Sometimes it is necessary to convert granules within one granularity. For example, one may need to know the label of the day that is 3 days after a certain date. Next conversion is proposed to accommodate this need.

Definition 11. Let G be a granularity. *Next conversion* within granularity G, denoted $Next_G(\cdot)$, is a mapping from $\mathbb{Z} \times \mathbb{Z}$ to the label set of G such that for each pair (i, n)

- if $n > 0$ and there exists a granule $G(j)$ that is the nth granule of G whose label is greater than i, let $Next_G(i, n) = j$;
- if $n < 0$ and there exists a granule $G(j)$ that is the $|n|$th granule of G whose label is less than i, let $Next_G(i, n) = j$;
- if $n = 0$ and $G(i)$ is a granule of G, let $Next_G(i, 0) = i$;
- otherwise let $Next_G(i)$ be undefined.

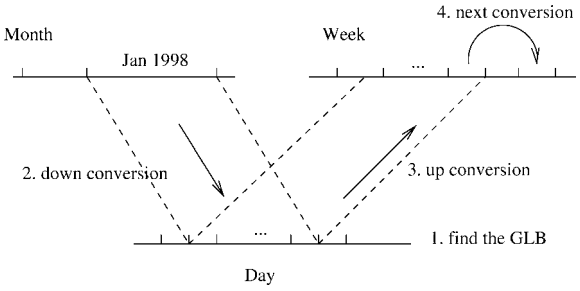


Figure 7. A conversion between `month` and `week`.

With down, up and next conversions, general conversions can be performed by further considering the conversion semantics. Let G_1 and G_2 be granularities involved in a granule conversion problem. The first step of the conversion would be to find the GLB of G_1 and G_2 in the calendar. Let granularity H be the GLB of them. With H as the intermediary, an appropriate set of granules of G_1 will be converted to H by down conversion. Then a corresponding set of granules of G_2 can be found by up conversion. Finally, the conversion problem is solved by the combination of up, down and next conversion using the conversion semantics.

For example, suppose we want to know the second week after `January 1998` (in `month`). As the first step, we find that their GLB in the calendar is granularity `day`. So we use `day` as the intermediary for this conversion. As the second step, the `day` that group into `January 1998` are collected by down conversion from `month` to `day`. As the third step, the `week` containing the last `day` of `January 1998` is found by an up conversion from `day` to `week`. Finally, the `second week` after `January 1998` is computed with a next conversion. Figure 7 shows the process of this conversion.

5.2. *Conversion semantics*

In TSQL2, two operations, *scale* and *cast*, are proposed to covert time values between granularities [18]. The conversion function $scale(g, H)$, where g is a sequence of granules of a granularity G and H is the target granularity, converts g into the smallest sequence h of granules of H such that h covers g. The conversion function $cast(g, H)$, where g is a sequence of granules (from l_G to u_G) of granularity G and H is the target granularity, returns the sequence of granules of H from l_H to u_H such that l_H is $\min(scale(l_G, H))$ and u_H is $\min(scale(u_G, H))$.

However, there are some granule conversions that cannot be expressed by using these two operations. For example, if we want to list the weekly sales that occurred within January 2000, we cannot use neither *scale* nor *cast* operation to locate the relevant weeks according to the month. Indeed, the *scale* operation always converts input granules to output granules that covers a bigger part of the time domain. In the following, we generalize *scale* into three types of conversions to accommodate other types of

granule conversions. The three types of conversions are distinguished by the relationships between the input and the output granules. Since *cast* is defined on the basis of *scale*, *cast* can be generalized similarly.

- *Covering*. The granule conversion should return all the granules of the destination granularity such that the time represented by the source granules contains the time represented by each destination granule.
- *Covered-by*. The granule conversion should return the smallest set of granules of the destination granularity such that the time represented by the source granules are covered by the time represented by the destination granules. This type of granule conversion corresponds to the direct generalization of *scale*. Note that *scale* deals with contiguous input granules, while granule conversion with respect to *covered-by* can accommodate input granules that are not contiguous.
- *Overlap*. The granule conversion should return all and only the granules of the destination granularity such that the time represented by the source granules overlaps the time of each destination granule.

5.3. Computation of down and up conversions

As discussed earlier, the computation of up and down conversions is very critical. Because of the label manipulation nature of the calendar algebraic operations, up and down conversions can be recursively computed. For the sake of presentation, we assume granularities G and H, where G is defined-only-on H, are the granularities involved in the up or down conversion.

Recall that all the granularities in layer 1 are full-integer labeled. Taking advantage of this fact, we use two recursive procedures, *first* and *last*, to help compute the down conversion from G to H. The procedure *first* takes full-integer labeled granularities G, H and an integer i as input and outputs the label of the first granule of H contained in $G(i)$. If G and H are the same, the procedure *first* trivially returns i; if G is defined by $G = \mathit{Shift}_m(H_1)$, the procedure recursively calls and returns $\mathit{first}(H_1, H, i - m)$; if G is defined by $G = \mathit{Group}_m(H_1)$, the procedure recursively calls and returns $\mathit{first}(H_1, H, (i-1)m + 1)$; if G is defined by $G = \mathit{Alter}^m_{l,k}(H_1, H_2)$, the procedure computes $j := \mathit{first}(H_2, H_1, i)$ and $h := \mathit{INT}((i-l)/m) + 1$, and depending on whether $i = (h-1)m + l$, it recursively calls either $\mathit{first}(H_1, H, j + (h-1)k)$ or $\mathit{first}(H_1, H, j + hk)$. The procedure *last* works in a similar way; it takes two full-integer labeled granularities G, H and an integer I as input, and outputs the label of the last granule of H contained in $G(i)$. (Details are omitted.) Thus, if both G and H are in layer 1, we can simply compute $j := \mathit{first}(G, H, i)$ and $j' := \mathit{last}(G, H, i)$, and then the down conversion $\lfloor i \rfloor^G_H = [j, j']$, i.e., the interval from j to j'.

The up conversions for the granularities in layer 1 can simply be computed recursively if no altering-tick operation is involved. For G and H in layer 1, if G is defined by $G = \mathit{Shift}_m(H_1)$, then $\lceil i \rceil^G_H = \lceil i \rceil^{H_1}_H + m$; if G is defined by $G = \mathit{Group}_m(H_1)$, then $\lceil i \rceil^G_H = \mathit{INT}((j-1)/m) + 1$, where $j = \lceil i \rceil^{H_1}_H$.

However, if the altering-tick operation is involved, for example, $G = Alter^m_{l,k}(H, H_2)$, there is no simple formula for the up conversion, since each granule of H_2 may contain different number of granules of H. Indeed, the formula (if we insist on having one) will be conditioned on the intervals of the granules of H, and the computation of an up conversion will involve multiple comparisons to determine which interval the input label falls in. In section 5.4, we will study an estimation based method that can give a rather accurate estimation when the granules of the coarser granularity (e.g., H_2 in the above example) contain similar number of finer granules (e.g., H in the example). Note that if we have a more complex altering-tick operation, for example, $G = Alter^m_{l,k}(H_1, H_2)$, we can first get $j = \lceil i \rceil^{H_1}_H$ and then $\lceil i \rceil^G_H = \lceil j \rceil^G_{H_1}$.

In layer 1, all the granularities are full-integer labeled. There exist simple formulas for up and down conversions for the shifting operation and the grouping operation. Though the altering-tick operation is a bit more complex, there also exists simple formula for the down conversion, and the up conversion can be performed on the basis of the down conversion. Furthermore, the up conversion for the altering-tick operation can be estimated, and the difference between the estimated value and the real up conversion is usually bounded by a small number. Suppose the number of basic operations that are involved in the conversion is n. The complexity of the up and the down conversion in layer 1 is linear to n if there is no up conversion for the altering-tick operation. If there exists up conversion for the altering-tick operation, the complexity is $O(n \log_2 P)$ in the worst case, where P is the bound between the estimated and the real up conversions. Therefore, there exist efficient algorithms for the up and down conversions in the first layer.

Down conversions involving granularities in layers 2 or 3 can be computed recursively according to the definitions of the operations that generate these granularities. For example, given $G = Subset^n_m(H_1)$, if $m \leqslant i \leqslant n$, then $\lfloor i \rfloor^G_H = \lfloor i \rfloor^{H_1}_H$; otherwise, $\lfloor i \rfloor^G_H$ is *undefined*. As another example, if G is defined by $G = Select\text{-}down^l_k(H_1, H_2)$, we first check whether i is a valid label of G. This is done by finding the granule $H_2(j)$ that contains $H_1(i)$, computing the set of granules of H_1 contained in $H_2(j)$, and then checking if $H_1(i)$ is selected by the operation. If i is a valid label of G, then $\lfloor i \rfloor^G_H = \lfloor i \rfloor^{H_1}_H$; otherwise, $\lfloor i \rfloor^G_H$ is *undefined*.

Up conversions involving granularities in layers 2 or 3 can be computed recursively as well, and sometimes with the help of the corresponding down conversions. For example, given $G = Subset^n_m(H_1)$, we first compute $j = \lceil i \rceil^{H_1}_H$. If j is not *undefined* and $m \leqslant j \leqslant n$, then $\lceil i \rceil^G_H = j$; otherwise, $\lceil i \rceil^G_H$ is *undefined*. As an example of using the corresponding down conversion, consider $G = Select\text{-}down^l_k(H_1, H_2)$ again. We first compute $j = \lceil i \rceil^{H_1}_H$ and then $s = \lfloor j \rfloor^{H_1}_H$. If s is *undefined*, $\lceil i \rceil^G_H$ is *undefined*; otherwise, $\lceil i \rceil^G_H = j$.

In general, the algorithms for up and down conversions involving granularities in layers 2 or 3 are not only affected by the number of operations involved, but also by the correspondence of the granules of both operand granularities, e.g., how many granules of the first operand are contained in the granules of the second operand in select-down operation. Because the labels of a second-layer or third-layer granularity may not be

contiguous, the conversion has to individually manipulate the labels. However, with further knowledge, e.g., one or both operands are in layer 1 or layer 2, the conversion algorithm can be more efficient. If both operands of the selecting operations or the combining operation are not in the third layer, then each operand granularity does not have inside gaps, and the processing of the coarser granularity is simplified. If it is further known that the finer operand is in layer 1, the fact that the labels of the finer operand are contiguous can be used, and the complexity of the conversion is only related to the number of operations involved.

5.4. Estimating up conversion for the altering-tick operation

In this subsection, we present an estimation-based approach to computing up conversions involving the altering-tick operation. Suppose H, G and G' are granularities in a calendar, where G is defined-only-on H and G' is defined by $G' = Alter^m_{l,k}(H, G)$. Our approach is based on the following observations.

- Granularity H groups periodically into G if G is defined-only-on H.
- For real life granularities, if G is defined-only-on H in a calendar, granules of G usually contain similar numbers of granules of H. (Recall that H groups into G if G is defined-only-on H.) For example, each month has about 30 days, though precisely speaking a month may have 28, 29, 30 or 31 days.

The first observation enables us to identify the period that the up conversion falls in, while the second provides an opportunity to estimate the label within the period. The exact up conversion can then be searched within the error bounds around the estimated value.

Identifying the period. Here we introduce some notations for convenience. Since H groups periodically into G, there must exist positive integers N and P such that for all labels i of G, if $G(i) = \bigcup_{j=j_1}^{j_2} H(j)$ and $G(i + P) \neq \emptyset$ then $G(i + P) = \bigcup_{j=j_1}^{j_2} H(j + N)$. Here we say P is the *period of G with respect to H* and N is the *period of H with respect to G*. Since G is defined-only-on H and only basic operations can be involved between them, the above periods can be recursively computed as follows.

- If $G = H$, then $P = 1$ and $N = 1$;
- If $G = Shift_m(G_0)$, and the period of G_0 with respect to H and the period of H with respect to G_0 are P_0 and N_0, respectively, then $P = P_0$ and $N = N_0$;
- If $G = Group_m(G_0)$, and the period of G_0 with respect to H and the period of H with respect to G_0 are P_0 and N_0, respectively, then $P = P_0/\gcd(m, P_0)$ and $N = N_0 m/\gcd(m, P_0)$;
- If $G = Alter^m_{l,k}(H, G_0)$, and the period of G_0 with respect to H and the period of H with respect to G_0 are P_0 and N_0, respectively, then $P = P_0 m/\gcd(m, P_0)$ and $N = (P_0 k + m N_0)/\gcd(P_0, m)$;

- If $G = Alter^{m}_{l,k}(G_0, G_1)$, and the period of G_1 with respect to H and the period of H with respect to G_1 are P_1 and N_1, respectively, then $P = P_1 m/\gcd(m, P_1)$ and $N = last(G, H, P) - last(G, H, 0)$.

To determine the period in which the up conversion $\lceil i \rceil^{G'}_{H}$ is, we use the granule $G'(0)$ as reference. Let t_0 be the label of the last granule of H contained in $G'(0)$, and $d = \lfloor (i - t_0)/N \rfloor$. Then we can easily determine that $dP + 1 \leqslant \lceil i \rceil^{G'}_{H} \leqslant (d + 1)P$.

We could just perform a binary search within the interval $[dP + 1, (d + 1)P]$ and determine whether a granule of G' is the result by performing a down conversion from G' to H. The complexity of such a method is $O(n \log_2 P)$, where n is the number of operations involved from G' to H, and P is as defined above.

Estimating up conversion. Our strategy to estimate the up conversion is to assume that G' is defined by a grouping operation on H. Given the periods P and N, the average number of granules of H that group into a granule of G' is $\overline{y} = N/P$. Then G' is assumed to be defined by $G' = Group_{\overline{y}}(H)$. It follows that the up conversion $\lceil i \rceil^{G'}_{H}$ can be estimated by $\overline{i} = (i - t_0 + 1)/\overline{y}$. Since H groups periodically into G', the estimated up conversion is in the same period as the real one.

Estimating error bounds. Now we try to estimate the upper and the lower bound of the difference between the real and the estimated up conversion. Suppose $G' = Alter^{m}_{l,k}(H, G)$. In addition, suppose for all granularities H' in the paths from G' to H in the dependency graph, if H' is defined by an altering-tick operation in the form of $Alter^{m'}_{l',k'}(H_1, H_2)$, then H_1 is defined by a series of grouping and shifting operations. Then each altering-tick operation $Alter^{m'}_{l',k'}(H_1, H_2)$ can be transformed into $Alter^{m'}_{l',k''}(H, H_2)$, where $k' = km_1m_2 \cdots m_n$, and $m_1, m_2, \ldots, m_n$ are the parameters involved in the grouping operations.

Let granularity G be any one in the paths from H to G' in the dependency graph. It follows that H partitions G. For each label i of G, let t_i be the last label of H contained in $G(i)$, and let $x_i = t_i - t_0$. By viewing granularity G as defined by the grouping operation $G = Group_{\overline{y}}(H)$, we can estimate $\lceil j \rceil^{G}_{H}$ as $\overline{i} = (j - t_0)/\overline{y}$. Suppose the real up conversion from H to G is i. Then the error of the estimation is $\overline{i} - i = (j - t_0)/\overline{y} - i = (j - t_0 - i\overline{y})/\overline{y}$.

Let $\delta_j = j - t_0 - i\overline{y}$. Now consider the bounds of δ_j. Since $H(j)$ is in the ith granule of G, $x_{i-1} < j - t_0 \leqslant x_i$. Let $\delta_{jl} = x_{i-1} - i\overline{y}$ and $\delta_{ju} = x_i - i\overline{y}$, then δ_j is always bounded by δ_{jl} and δ_{ju}, i.e., $\delta_{jl} < \delta_j \leqslant \delta_{ju}$. Let $\delta_l = \min(\delta_{jl})$, $\delta_u = \max(\delta_{ju})$. Then $\delta_l < \delta_j \leqslant \delta_u$ for each granule $H(j)$. Thus, the error of the estimated up conversion is always bounded by $\delta_l/\overline{y}$ and $\delta_u/\overline{y}$.

The integral components of the error bound, δ_l and δ_u, can be estimated recursively as shown in the following three cases.

(1) $G' = Shift_m(G)$.
Since $G'(i) = G(i - m)$ for each label i of G', $x'_i = x_i - x_m$. Since the ith granule of G' should be the $(i - m)$th granule of G, for each $H(j)$, $\delta'_{jl} = x_{i-1} - x_m - (i - m)\overline{y} =$

$\delta_{jl} - (x_m - m\overline{y}) \geqslant \delta_l - \delta_u$. It follows that $\delta'_l = \delta_l - \delta_u$. Similarly, we can get $\delta'_u = \delta_u - \delta_l$.

(2) $G' = Group_m(G)$.
For each granule $G'(i)$, $x'_i = x_{im}$. For each $H(j)$, $\delta'_{jl} = x'_{i-1} - i\overline{y'} = x_{(i-1)m} - [(i-1)m+1]\overline{y} - (m-1)\overline{y} > \delta_l - (m-1)\overline{y}$, and we can have the lower bound $\delta'_l = \delta_l - (m-1)\overline{y}$. Similarly, since $\delta'_{ju} = x'_i - i\overline{y'} = x_{im} - im\overline{y} \leqslant \delta_u$, the upper bound is $\delta'_u = \delta_u$.

(3) $G' = Alter^m_{l,k}(H, G)$.
For each granule $G'(i)$, we have $x'_i = x_i + (\lfloor (i-l)/m \rfloor + 1)k$. Consider the estimation of δ_l. If $k > 0$, $x'_i \geqslant x_i + ((i-l)/m)k$. Then for each granule $H(j)$,

$$\delta'_{jl} = x'_{i-1} - i\overline{y'} \geqslant x_{i-1} + \left(\frac{i-1-l}{m}\right)k - i\left(\overline{y} + \frac{k}{m}\right) = \delta_{jl} - k\frac{l+1}{m}.$$

If $k < 0$, $x'_i \geqslant x_i + ((i-1-l)/m + 1)k$. Then for each granule $H(j)$,

$$\delta'_{jl} = x'_{i-1} - i\overline{y'} \geqslant x_{i-1} + \left(\frac{i-1-l}{m} + 1\right)k - i\left(\overline{y} + \frac{k}{m}\right) = \delta_{jl} + k\left(1 - \frac{l+1}{m}\right).$$

Thus, δ'_l can be estimated as follows.

$$\delta'_l = \begin{cases} \delta_l - k\dfrac{l+1}{m}, & \text{if } k > 0, \\ \delta_l + k\left(1 - \dfrac{l+1}{m}\right), & \text{if } k < 0. \end{cases}$$

Now consider the estimation of δ_u. If $k > 0$, $x'_i \leqslant x_i + ((i-l)/m + 1)k$. Then for each granule $H(j)$,

$$\delta'_{ju} = x'_i - i\overline{y'} \leqslant x_i + \left(\frac{i-l}{m} + 1\right)k - i\left(\overline{y} + \frac{k}{m}\right) = \delta_{ju} + k\left(1 - \frac{l}{m}\right).$$

If $k < 0$, $x'_i \leqslant x_i + ((i-1)/m)k$. Then for each granule $H(j)$,

$$\delta'_{ju} = x'_i - i\overline{y'} \leqslant x_i + \frac{i-l}{m}k - i\left(\overline{y} + \frac{k}{m}\right) = \delta_{ju} - k\frac{l}{m}.$$

Thus, δ'_u can be estimated as follows.

$$\delta'_u = \begin{cases} \delta_u + k\left(1 - \dfrac{l}{m}\right), & \text{if } k > 0, \\ \delta_u - k\dfrac{l}{m}, & \text{if } k < 0. \end{cases}$$

With the above recursive estimation, we can always get the δ_l and δ_u for G' in time linear to the number of involved operations. As a result, we can always be certain that the real value of any up conversion from H to G is within the interval $[\overline{i} + \delta_l/\overline{y}, \overline{i} + \delta_u/\overline{y}]$.

For example, for up conversions from `day` to `month` (see section 3.3 for corresponding algebraic expressions), $\delta_l/\overline{y} = -1.158$ and $\delta_u/\overline{y} = 0.138$. Then for all estimated up conversion $\overline{i}$, the real up conversion is bounded by $[\overline{i} - 1.158, \overline{i} + 0.138]$. With the estimated error bounds, the up conversion can be done by an estimation of the up conversion followed by a binary search within the bound. The the complexity becomes $O(nb)$, where n is the number of operations and $b = \lceil(\delta_u - \delta_l)/\overline{y}\rceil$.

The error bounds are estimated conservatively. The estimated bounds are usually looser than the real error. This suggests that the search for the up conversion within the bound should be biased towards the estimated value.

5.5. *Computation of next conversion*

Next conversion is trivial for layer 1 granularities because of the contiguity of their labels ($Next_G(i, n) = i + n$). However, it can be a difficult problem for the granularities in layers 2 and 3, where labels may not be contiguous any more.

A desirable solution would be getting the result with the information of the operations. For example, given a granularity that stands for the first day of every month, to get the nth granule after a base granule, say i, we only need to get the nth month after the month containing granule i, and finding the result would be easy. In this case, the next conversion for one granularity is translated into a trivial one for another granularity. However, this is not a general solution for all granularities. In this paper, we outline several alternative ways.

There are two straightforward ways to solve this problem in addition to making use of the information of the operations.

1. Search for the nth granule by testing. The basic construct is to determine whether an integer is a valid label or not. To get the result, the algorithm tests the integers one by one until the nth valid granule is found.
2. Enumerate the valid labels. This involves precomputation and storage of valid labels. For periodical granularities, enumeration of one period is enough.

Obviously, the first method is only suitable for small n and granularities that do not have big gaps between adjacent labels. In other cases, it will result in unacceptable performance. Although the second method has good computation performance, it does not scale well when the period gets big. In the following, we propose several enhancements that can improve the scalability.

Our first enhancement is to use a hash table to maintain the valid label information for a granularity. We distinguish two kinds of hash tables, the first one is a *positive* hash table, in which valid labels within a period are stored, and the second one is a *negative* hash table, in which missing labels (i.e., the integers that are not valid labels) are stored. It is easy to see that the positive hash table and the negative hash table are complementary. The positive hash table is used when most of the integers in a period are not valid labels, while the negative hash table is used when most of the integers in a period are labels. The distinction of positive and negative hash tables reduces the size

of the enumeration by at least a half. However, this may not solve the scalability problem. Nevertheless, we can improve the scalability by sacrificing some computational efficiency. If the hash table gets too big, we can reduce the size by only storing a part of valid labels.

An alternative enhancement is to use bitmap for the valid labels of a granularity. Similarly, bitmap is only necessary for a period. Each bit in the bitmap corresponds to an integer. A bit is 1 if the corresponding integer is a valid label, 0 if not. Finding the nth granule after a base granule is just counting n 1s in the bitmap and finding the corresponding integer. The advantage of such representation is that only generalized granularities defined by selecting operations need precomputed bitmap. If a granularity is defined by a set operation, then the corresponding bitmap can be easily composed with the bitmaps of the operands. Suppose A and B are granularities with bitmaps a and b, respectively. Then the bitmaps for $A \cup B$, $A \cap B$ and $A - B$ are a OR b , a AND b and a AND (NOT b), respectively. To save the space, compression method, e.g., run length encoding, can be utilized, which can make the counting of 1s even faster. However, the bitmap method may not scale well for granularities with long periods.

Note that the above two specific enhancements may be applied to calendar representation schemes that are based on enumeration. The difference, though, is that these enhancements are among other possibilities. Other methods that directly take advantage of the compact algebraic representations of this paper may not be available to enumeration-based representation schemes. Since the "rules" with which the granularities are generated are encoded in the calendar expressions and can be *recovered* when needed, the algebraic representation gives rise to more opportunities for optimizing our computations involving granularities. For example, suppose we define a granularity `FirstDayOfMonth` = *Select-down*$_1^1$(`day`, `month`), where `day` and `month` are defined in section 3.3. From the calendar expressions associated with `PseudoMonth` and `month`, it is easy to figure out that there is exactly one granule of `FirstDayOfMonth` contained in each granule of `month`. Then the next conversion for `FirstDayOfMonth` can be transformed to a conversion to `month` with respect to *covered-by* (or *overlap*), followed by a next conversion for `month`, which is trivial since `month` is full-integer labeled, and then followed by a conversion with respect to *covering* (or *overlap*). Nevertheless, a general framework that can take full advantage of the algebraic representations is beyond the scope of this paper.

5.6. Vector labels: relative representation of granules

The mapping viewpoint of granularity reveals the nature of time granularities, and time can be processed based on the labels (or indices) of the granularities, which are countable and easy to process with computers. However, human users are used to *relative and textual representation* of granules. It is rather difficult for human beings to use, for example, the 731,854th granule of `day`.

Computer applications that interact with human users often provide textual representation for time. For example, database applications usually print a SQL DATE as a

label consisting of the year, month and day of the particular date. However, the textual representations are usually predefined for specific time granularities (e.g., granularity `day` in the above example). Textual representation of user-defined granularities would require special care.

The idea of label mappings for formally defined granularities has been suggested in [1] to provide a relative representation of granules. Several works, though proposed before [1], can be used to partially address this issue. The 1-dimensional space proposed in [13] can be used to provide a relative representation of granules with a sequence of granularities. However, the granularities used with a 1-dimensional space are restricted to those that have "regular" relationships between each other, i.e., each granule of a coarser granularity must have the same number of granules of a finer granularity. Thus, we cannot use granularities, for example, `year`, `month` and `day` to provide a relative representation for days. The *Set of Composite Numbers* [14], which was proposed for nonmetric measurement systems, and the representation mechanism for time spans using multiple granularities [9] can be used to provide relative representations for unanchored temporal data (i.e., time durations). However, they cannot be directly applied to relative representations of granules. (Indeed, they both have restrictions when representing time durations. Details will be discussed in section 6.)

In this subsection, by applying the granule conversions discussed in the previous section, we introduce a notion of *vector label* to provide more general label mappings for granules. Note that vector labels do not address the relative representations of time spans (i.e., time durations).

We assume that all the "words" used in the textual representation of granules are enumerable, so they can be encoded as integers. For example, the English words for months, i.e., January, February, March, ..., and December, can be encoded as integers 1 to 12. As a result, we need only deal with integers for the textual representation.

Definition 12. Given a granularity G, let $G_k, G_{k-1}, \ldots, G_1$ be a sequence of granularities, where G is a label-aligned subgranularity of G_1. A vector $(j_k, j_{k-1}, \ldots, j_1)$ in N^k is said to be a *vector label* of granule $G(i)$ with respect to *overlap/covered-by* if by letting $j'_k = j_k$ and $G_l(j'_l)$, $1 \leqslant l < k$, be the j_lth granule of all the granules of G_l that overlap/are covered-by $G_{l+1}(j'_{l+1})$ we can have $G_1(j'_1) = G(i)$. If any of the above j_lth granule does not exist, then the above vector is not a vector label of any granule. The sequence of granularities $G_k, G_{k-1}, \ldots, G_1$ are called the *referent granularities* for the vector label.

For example, consider the granularity `Sunday` (which is label-aligned subgranularity of itself) and the referent granularities `year`, `month`, `Sunday`. Then (1998, 7, 2) is a vector label with respect to overlap (or covered-by) that represents *the second Sunday in July 1998*. In addition, since `Sunday` is label-aligned subgranularity of `day`, we can also use `year`, `month`, `day` as referent granularities. In this case, the vector (1998, 5, 3) (i.e., May 3rd, 1998), which is the first Sunday in May 1998, is a valid vector label with respect to overlap (or covered-by) for `Sunday`, while the vector (1998, 5, 4) (i.e.,

May 4, 1998, which is a Monday) is not a vector label, since it cannot be mapped to any granule of `Sunday`.

In the above examples, the vector labels with respect to overlap are exactly the same as the vector labels with respect to covered-by. This is because the referent granularities form a total order with respect to the finer-than relationship. (`Sunday` (`day`) is finer than `month`, which is finer than `year`.) When this total order does not exist, the vector labels with respect to overlap and covered-by will have different meanings. For example, with the referent granularities `year` and `week`, the vector label (1998, 1) with respect to overlap refers to the first week that overlaps with year 1998, which includes December 28, 1997 through January 3, 1998, and the vector label (1998, 1) with respect to covered-by refers to the first week that is contained in year 1998, which includes January 4 through 10 in 1998.

In the formalism of vector labels, we used the conversion semantics overlap and covered-by, but did not use the conversion semantics covering. Intuitively, vector labels with respect to overlap (covered-by) correspond to locating smaller granules of "finer" granularities through bigger granules of "coarser" granularities using the overlap (covered-by) conversion semantics; however, it cannot locate nothing to use the covering semantics with vector labels. On the other hand, it is against human intuition to locate bigger granules of "coarser" granularities through smaller granules of "finer" ones. For example, human users never refer to a year as *the first year that contains the* 731,854th *day*.

Vector labels can be easily converted into labels by granule conversions. For example, given referent granularities `year`, `month`, `Sunday` and a vector label (1998, 7, 2) with respect to overlap, we can get the label, say l_m, of the 7th month that overlaps with the year 1998 by a granule conversion between `year` and `month` and then get the label, say l, of the 2nd Sunday that overlaps with the above month by another granule conversion between `month` and `Sunday`. On the other hand, vector labels can be generated from labels by granule conversions as well. For example, given the label l of the above Sunday, we can first compute the label l_m of the month that overlaps with this Sunday, then by another granule conversion get the set of Sundays that overlaps with month l_m, and find out this is the 2nd Sunday in the month, that is, the last position of the vector label is 2. Similarly, we can compute that year 1998 overlaps with the month l_m and the month is the 7th among all the months overlapping with year 1998. Thus, the vector label with respect to overlap of the Sunday is (1998, 7, 2).

6. Related work

Much work has been done on the problem of granularity representation in temporal database area as well as other areas like artificial intelligence and real time systems. Some of them address the formalization of time granularity systems [4,6,7,15]. Our work is an instantiation of the general framework proposed in [4].

A symbolic representation of granularities that allows natural language expression was proposed in [10] on the basis of structured collections of intervals. This represen-

tation was later implemented in *POSTGRES* [5]. As the foundation of the system, the primitive collections, e.g., *day*, *month*, *year*, have to be enumerated, though there exist some patterns in them. Another formalism was later introduced in [16] as an alternative to the above one. On the basis of existing granularities, it defines a new, periodical granularity by enumerating a set of starting points and durations within a period. The expressiveness of these representations were studied in [3] and an extension to the former one was proposed. Both of the two representations (and the extension) require enumeration of either the primitive granularities (collections) or the newly defined ones. Our work does not require explicit enumeration but tries to capture the patterns of the granularities; thus, our approach usually requires less space and can encode the relationships between granularities in a more compact and direct way.

There are also other proposals for granularity representation. Wijsen proposed a string-based model called *granspec* to represent periodical granularities as repeated string patterns over a set of three symbols [19]. The canonical representation of granspecs and the symbolic computation over granspecs were also studied. In [11], a granularity (called calendar in [11]) was modeled as a totally ordered set of intervals with additional semantics, and several calendar operations are introduced to generate user defined time granularities. Both of these proposals define a new granularity by the relative pattern of its granules with respect to the granules of another granularity. Similar to the primitive collections in [10], this is basically enumeration. Our approach can achieve the same results with a subset of operations without enumeration.

To accommodate mixed granularities in TSQL2 (the temporal extension to SQL), Dyreson et al. proposed to organize multiple granularities into a lattice and support the query language using two operations, *scale* and *cast* [18]. This approach was further refined in [8,12]. Specifically, the user specifies granularities by providing conversion functions (*scale* and *cast*) between some pairs of granularities. Two types of mappings were distinguished: regular mapping, which has well-defined formula, and irregular mapping, which is specified by user supplied functions. On the one hand, their conversion approach could be more efficient than ours, since the user supplied conversion functions could be optimized for the granularities involved. On the other hand, our approach involves less development cost and is easier to use. In addition, we generalized the conversion function *scale* to granule conversion with respect to *covered-by*, and included granule conversions with respect to *covering* and *overlap*, which cannot be expressed by the original *scale* function. We do not have the *cast* function; however, it can be easily added into our model, since *cast* is defined on the basis of *scale*.

Lorentzos introduced two generic data types, named (*compound*) *1-Dimensional* (*1-D*) *Space* and *Set of Composite Numbers* (*SCN*), to provide compound representation and operations for non-metric data types [13,14]. Though targeted more generic data types, 1-D space and SCN can be used to provide relative representation for time points and durations, respectively, using mixed granularities. However, both compound 1-D space and SCN are quite restrictive, since they can only deal with "regular situations" where the granules of the finer granularities are evenly distributed in the granules of the coarser ones. For example, 1-D space and SCN can deal with granularities with "regular"

relationship (e.g., `Hour`, `Minute` and `Second`), but they cannot accommodate those with "irregular" relationship like `Month` and `Day`, since one month may have 28 to 31 days. Our formalism of vector labels and *Up* and *Down* conversion functions are more general than 1-D space and the associated functions in the sense that they can accommodate both the "regular" granularities (generated by the *Group* operation) and the "irregular" ones generated by other calendaric operations.

Goralwalla et al. studied representation of unanchored temporal data, conversion between time spans represented in mixed granularities as well as canonical forms for unanchored time spans [9]. We consider this work as complementary to ours, since our work focuses on the representation of anchored temporal granularities and conversion between anchored temporal data. Indeed, our *Up* and *Down* conversions can be used to derive the "conversion functions" defined in [9], which are the foundation of the conversions between unanchored time spans represented in mixed granularities. In other words, our work can be combined with the approach proposed in [9] to address the conversions of unanchored temporal data. The approach proposed in [9] has two limitations. First, it does not consider the granularities with gaps within granules (e.g., *business month*). Second, the conversions discussed in [9] are based on a (hidden) assumption (reflected by the definition of *conversion function*) that in a totally ordered list of granularities, each granule of a coarser granularity consists of an integral number of granules of a finer granularity.

Finally, there is an interesting program called Remind that provides a script language to define sophisticated "calendars" and alarms [17]. However, the word "calendar" here really means *schedule* or *arrangement*. Indeed, the script language in Remind is designed to specify complex event patterns (i.e., triggers and alarms) on the basis of existing calendars (i.e., Gregorian and Hebrew calendars), while our calendar algebra is to define the calendars themselves. Thus, we consider the script language in Remind as complementary to ours, and it can be used on top of the granularities defined using the calendar algebra.

7. Conclusion

In this paper, we presented an algebraic representation of time granularities. This representation is natural and flexible in the sense that it captures the ways in which human calendars are organized. As a consequence, the calendar algebra usually leads to compact representations. Given the assumption that there exists a bottom granularity known to the system, the calendar algebra can represent all the granularities that are either periodical or finite with respect to the bottom granularity.

The granularities generated by calendar algebraic operations from the bottom granularity can be organized into a calendar. We partition the granularities in one calendar into three layers to help check the preconditions of the algebraic operations. With the three-layer restriction, validation of operations can be performed efficiently. Although all the granularities in a calendar are originated from the bottom granularity, they do not necessarily depend on each other. We say one granularity is defined-only-on another

granularity if the generation of the former one depends solely on the latter one. If one granularity is defined-only-on another granularity, then the latter one groups into the former one.

To make the calendar algebra useful for applications such as temporal databases, we studied the granule conversion problem. General conversion between granularities can be performed on the basis of three basic conversions (up conversion, down conversion, and next conversion). The computation of the three basic conversions are the foundation of granule conversion. In layer 1, all the three basic conversions can be performed quite efficiently. In layers 2 and 3, the computation is more complex and may be affected by the correspondence of the granules of the operand granularities. As an interesting application, granule conversion was applied to provide a relative representation of granules.

Acknowledgements

The authors are grateful to the anonymous reviewers for their valuable comments. The work was partially supported by a grant from the U.S. Army Research Office under the contract number DAAG-55-98-1-0302 and a grant from the National Science Foundation with the grant number 9633541. The work of Wang was also partially supported by a Career Award from the National Science Foundation under the grant number 9875114.

References

[1] C. Bettini, C.E. Dyreson, W.S. Evans, R.R. Snodgrass and X. S. Wang, Temporal databases: research and practice, in: Lecture Notes in Computer Science, Vol. 1399 (Springer, 1998) chapter "A glossary of time granularity concepts".

[2] C. Bettini, S. Jajodia and X.S. Wang, *Time Granularities in Databases, Data Mining, and Temporal Reasoning* (Springer, 2000).

[3] C. Bettini and R.D. Sibi, Symbolic representation of user-defined time granularities, in: *Proc. of 6th Int'l Workshop on Temporal Representation and Reasoning* (1999) pp. 17–28.

[4] C. Bettini, X.S. Wang and S. Jajodia, A general framework for time granularity and its application to temporal reasoning, Annals of Mathematics and Artificial Intelligence 22(1–2) (1998) 29–58.

[5] R. Chandra, A. Segev and M. Stonebraker, Implementing calendars and temporal rules in next generation databases, in: *Proceedings of ICDE* (1994) pp. 264–273.

[6] J. Clifford and A. Rao, A simple, general structure for temporal domains, in: *Proc. of the Conference on Temporal Aspects in Information Systems* (1987) pp. 23–30.

[7] T. Dean, Using temporal hierarchies to efficiently maintain large temporal databases, Journal of ACM 36 (1989) 687–718.

[8] C.E. Dyreson, W.S. Evans, H. Lin and R.T. Snodgrass, Efficiently supporting temporal granularities, IEEE Transactions on Knowledge and Data Engineering 12(4) (2000) 568–587.

[9] I.A. Goralwalla, Y. Leontiev, Özsu, D. Szafron and C. Combi, Temporal granularity for unanchored temporal data, in: *Proc. of the 1998 ACM CIKM Internat. Conference on Information and Knowledge Management* (1998) pp. 414–423.

[10] B. Leban, D. McDonald and D. Foster, A representation for collections of temporal intervals, in: *Proceedings of AAAI* (1986) pp. 367–371.

[11] J.Y. Lee, E. Ramez and J. Won, Specification of calendars and time series for temporal databases, in: *Int'l Conf. on the Entity Relationship Approach* (1996) pp. 341–356.
[12] H. Lin, Efficient conversion between temporal granularities, Technical report 19, Time Center (1997).
[13] N.A. Lorentzos, DBMS support for time and totally ordered compound data types, Information Systems 17(5) (1992) 347–358.
[14] N.A. Lorentzos, DBMS support for nonmetric measurement systems, IEEE Transactions on Knowledge and Data Engineering 6(6) (1994) 945–953.
[15] A. Montanari, E. Maim, E. Ciapessoni and E. Ratto, Dealing with time granularity in the event calculus, in: *Proc. of the Internat. Conference on Fifth Generation Computer Systems*, Tokyo, Japan (1992) pp. 702–712.
[16] M. Niezette and J. Stevenne, An efficient symbolic representation of periodic time, in: *Proceedings of CIKM* (1992) pp. 161–168.
[17] D.F. Skoll, Remind calendar program, available at http://www.roaringpenguin.com/remind.html.
[18] R.T. Snodgrass (ed.), *The TSQL2 Temporal Query Language* (Kluwer Academic, Dordrecht, 1995).
[19] J. Wijsen, A string-based model for infinite granularities, in: *AAAI-2000 Workshop on Spatial and Temporal Granularities* (2000) pp. 9–16.