

is the temporal module (W, G_w, φ') such that $\varphi'(i) = \pi_w(\varphi(i))$ for each $i \geq 1$. Note that π_w is the standard project operation. Similarly, $\sigma_F^T(M)$ is the temporal module (R_M, G_M, φ') such that $\varphi'(i) = \sigma_F(\varphi(i))$ for each $i \geq 1$. Note that σ_F is the standard select operation. Finally, if M_1 and M_2 have the same schema (R, G) , then $M_1 \cup_r M_2$ is the temporal module (R, G, φ) , where $\varphi(i)$ is the union of the windowing functions of M_1 and M_2 at each index i .

1.3 Bibliographic Notes

Extensive literature documents the general notion of time addressed by different disciplines, from philosophy to history, astronomy to physics, and psychology to computer science. Among the books addressing the issue of calendars and time granularities, [Dun98, Gou97, DvR96] are good recent references which also have extensive bibliographies. In the area of database systems, one of the earliest formalizations of time granularities can be found in [CR87]. The relevant role played by time granularity in temporal databases has been recognized by this research community, devoting a large part to this issue in the book describing the temporal extension of SQL (TSQ12) [TSQ12], an effort that has involved many of the researchers in this area for quite a long time. However, a clear and formal characterization of time granularities and their relationships was still missing in that work. Other papers involving time granularity issues in databases include [WJ191, NS92, SC93, CSS94, Wij95, Wij98, Wij99].

The formalization of time granularities as it is presented in this book has been refined during a long-term research activity [WJS95, WBB+97, BWB+95, Wan95, BWJ96b, BWJ98b, BWJ+98, BWJ98a, BWJ98c, BD99]. Relevant terms have been recently summarized in a glossary [BDE+98].

Other relevant work on time granularity has been done in the context of temporal and classical logics [MMC+92, CCM+93, Mon96, MPP99] with applications to the verification of real-time system specifications and temporal reasoning in Artificial Intelligence. Other research efforts from the area of Artificial Intelligence include [Hob85, LMF86, Lad87, DM87, Dea89, PB91, MLK95, Euz95, Ter95, CD98, GIO+98, Sha97, Ker99, SH99]. Most of these approaches will be discussed along the way when we address the related issues in this book.

In this section, we have also discussed temporal databases. Many research papers have been written on temporal databases: the paper [Soc91] and the successive updates [Kh93, TK96, WJW98] provide the best sources for a bibliography. These sources comprise almost all relevant papers published in this area. [TDB93] and the more recent [TDB98] contain collections of papers that form a good overview of the research in temporal databases. Finally, [CT99] provides an excellent presentation of temporal databases from a mathematical logic perspective.

2. Granularity Systems

The calendar is intolerable to all wisdom, the horror of all astronomy, and a laughing-stock for a mathematician's point of view.

Roger Bacon, 1267

2.1 Introduction

In this chapter we elaborate on the notion of time granularity, identifying structural properties, relationships, and symbolic representations. When time granularities are used within specific domains or applications, they are usually restricted to a subset of those satisfying the very general definition given in Chap. 1.

We propose the notion of a *granularity system* as the characterization of this subset, defined by a specific choice for the time domain, by formal properties of the member granularities, and by specific relationships between the member granularities. This formal characterization is an excellent tool to compare the different definitions of granularity that have been proposed, as well as to compare the expressive power of symbolic representations. Indeed, the granularity definitions we are aware of in the literature can all be considered particular granularity systems in this framework.

In this chapter we also address the issue of how time granularities can be effectively represented and manipulated in a computer, and how new granularities can be easily defined in terms of others. In particular, we propose an algebra for symbolic manipulation of granularities and we compare it with previous proposals from the literature.

The rest of this chapter is organized as follows. In Sect. 2.2 a number of notions related to a granularity are defined as well as several relationships that can hold among granularities. In Sect. 2.4 we study granularity systems and their properties. Section 2.3 discusses the conversion among granularities. Sections 2.5 and 2.6 address the issues of symbolic representation and algebraic operations, and Sect. 2.7 presents related work.

2.2 Formal Notions

We have introduced in Sect. 1.1 the notion of time domain, denoted by the pair $(T, \leq)$ where T is a nonempty set of time instants and $\leq$ is a total order on T . If t and t' are elements of T , the expression $t < t'$ is sometimes used instead of $t \leq t' \wedge t \neq t'$ to identify strict order between them.

The choice for time domain is typically between dense and discrete:

A time domain is *dense* if it is an infinite set and for all $t, t' \in T$ with $t < t'$, there exists $t'' \in T$ such that $t < t'' < t'$.

A time domain is *discrete* if every element except the last element (if any) has an immediate successor, and every element except the first (if any) has an immediate predecessor.

Specific applications may also need to impose left/right boundedness to the time domain. A time domain is *bounded* if it contains upper and/or lower bounds with respect to its order relationship. Formally (in the case of both lower and upper bounds), time domain T is bounded if there exist $t', t'' \in T$ such that $t' \leq t \leq t''$ for all $t \in T$. (t' is called *beginning* in [JD98].)

Similarly, a granularity can be *bounded* if there exist lower and upper bounds k_1 and k_2 in the index set such that $G(i) = \emptyset$ for all i with $i < k_1$ or $k_2 < i$.

A number of notions related to a granularity are now considered and formally defined.

Definition. Each nonempty subset $G(i)$ where i is one of the indexes and G is a granularity, is called a granule of G .

Intuitively, a granule is a set of time instants perceived as a nondecomposable temporal entity when used to describe, in terms of the granularity G , a phenomenon or, in general, when used to timestamp a set of data. A granule can be composed of a single instant, a set of contiguous instants (time-interval), or even a set of noncontiguous instants. For example, the September 1997 business month, defined as the collection of all the business days in September 1997, can be used as a granule. Two granules $G(i)$ and $G(j)$ are *contiguous* if there does not exist $t \in T$ such that $G(i) < t < G(j)$. (We say $S < t$ if and only if $\forall s \in S, s < t$, and $t < S$ if and only if $\forall s \in S, s > t$.)

It is often useful (for implementations in particular) to specially designate one granule, called the *origin* of granularity G , e.g., $G(0)$. For example, the origin of UNIX-calendar-seconds is Jan 1 0:00:00 GMT 1970. Moreover, the greatest lower bound of the set of time domain elements corresponding to the origin of G is called the *anchor* of G . Intuitively, the anchor is the earliest time domain element in the origin. It serves to “anchor” the granularity with respect to the time domain, when that granularity is specified in terms of another granularity.

Definition. The *image* of a granularity is the union of the granules in the granularity.

As an example, the image of business-days-since-1990 is the set of time instants included in each granule representing a business day, starting from the first one in 1990. The *extent* of a granularity is the smallest interval of the time domain that contains the image of the granularity.

Definition. The *extent* of a granularity G is the set $\{t \in T \mid \exists a, b \in I(G), a \leq t \leq b\}$ where T is the time domain and $I(G)$ is the image of granularity G .

In time domain $(Z, \leq)$, the extent of business-days-since-1990 is $\{t \mid t \geq k\}$ where k identifies the first instant of the time domain within the first business-day of 1990. Note that many commonly used granularities (e.g., day, month, year) have their image equal to their extent, since each granule is formed by a set of contiguous elements of the time domain and each pair of contiguous indexes is mapped to contiguous granules. This is not the case, for example, of business-day, whose image does not include any instant within a Saturday or Sunday, while its extent does.

2.2.1 Granularity Relationships

In the following we identify a number of relationships between granularities having the same time domain.

Definition. A granularity G groups into a granularity H , denoted $G \leq H$, if for each index j , there exists a (possibly infinite) subset S of the index set such that $H(j) = \bigcup_{i \in S} G(i)$.

The diagram in Fig. 2.1 shows an example of a *groups into* relationship.

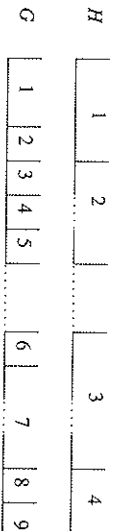


Fig. 2.1. The *groups into* relationship

Two granularities, G and H , are depicted in the diagram. Each granule is represented by an interval (with possible gaps shown as dotted segments). Granularity H consists of the granules indexed 1–4, while G has granules 1–9. The diagram also depicts the position of each granule with respect to the ordering in the underlying (implicit) time domain, so granules that overlap share elements from the underlying time domain, e.g., granule $G(1)$ is a subset of granule $H(1)$. Note that, with respect to elements in the time domain, there may be “gaps” between granules such as the one between $G(5)$ and $G(6)$, and “holes” within granules such as the one within $H(2)$. In this diagram, $G \leq H$ since every granule in H is the union of some set of granules in G . G also has an additional granule, namely $G(4)$, which has no counterpart in H .

As another example, day groups into business-day (where a business-day is Monday through Friday, excluding holidays), and day also groups into business-week.

Definition. A granularity G is finer than a granularity H , denoted $G \preceq H$, if for each index i , there exists an index j such that $G(i) \subseteq H(j)$. If $G \preceq H$, then H is coarser than G ($H \succeq G$).

The diagram in Fig. 2.2 depicts a *finer than* relationship.



Fig. 2.2. The *finer than* relationship

$G \preceq H$ since every granule in G is a subset of some granule in H . For example, business-day is finer than day, and also finer than week.

Definition. A granularity G is a subgranularity of H , denoted $G \sqsubseteq H$, if for each index i , there exists an index j such that $G(i) = H(j)$.

Definition. Granularities G and H are shift equivalent, denoted $G \leftrightarrow H$, if there exists an integer k such that $G(i) = H(i + k)$ for all i in the index set. Note that $G \leftrightarrow H$ if and only if $G \sqsubseteq H$ and $H \sqsubseteq G$.

It is possible that several granularities are different (have different mappings) but include the same sets of absolute time. For example, let G and H be two granularities denoting years. Suppose G denotes years of the Gregorian calendar, starting from $G(1)$ mapped to the first year, up to $G(2000)$ mapped to year 2000, and every other index mapped to the empty set. Consider H such that $H(-999)$ is mapped to the first year of the Gregorian calendar, $H(0)$ is year 1000, up to $H(1000)$ mapped to year 2000, and every other index mapped to the empty set. These two granularities cover the same period of absolute time dividing it into the same granules; however, they use different indexes to refer to a certain granule. While it can be sometime useful to distinguish among these granularities, it is easy to see that they can be considered equivalent by “shifting” the indexes. This is exactly the notion captured by the definition of shift-equivalent granularities:

A relationship related to groups into, but more restrictive, is *partitions*.

Definition. A granularity G partitions a granularity H if $G \sqsubseteq H$ and $G \preceq H$.

In the diagram in Fig. 2.3, G partitions H .

Note the similarity with the groups-into diagram given earlier. The only difference is that the “additional” granule in G (granule 4 in Fig. 2.1) that had no counterpart in H has been removed since its presence would violate

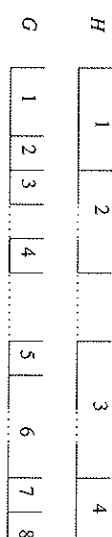


Fig. 2.3. The *partition* relationship

the finer-than relationship required for partitions. Many common (Gregorian calendar) granularities are related by the partitions relationship, e.g., day partitions week and month partitions year.

Definition. A granularity G is covered by a granularity H , denoted $G \subseteq H$, if the image of G is contained in the image of H . Formally, G is covered by H if $\bigcup_{i \in \mathcal{I}} G(i) \subseteq \bigcup_{j \in \mathcal{I}} H(j)$. If G is covered by H , then H covers G ($H \supseteq G$).

Definition. A granularity G groups periodically into a granularity H if

1. $G \sqsubseteq H$, and
2. there exist $R, P \in \mathbb{Z}^+$, where R is less than the number of granules of H , such that for all $i \in \mathbb{Z}$, if $H(i) = \bigcup_{r=0}^R G(i_r)$, and $H(i + R) \neq \emptyset$, then $H(i + R) = \bigcup_{r=0}^R G(i_r + P)$.

If $R = 1$, we also say that G groups uniformly into H .

The *groups periodically* into relationship is a special case of *groups into* characterized by a periodic repetition of the “grouping pattern” of granules of G into granules of H . Its definition may appear complicated but it is actually quite simple. Since G groups into H , any granule $H(i)$ is the union of some granules of G ; for instance, assume it is the union of the granules $G(a_1), G(a_2), \dots, G(a_k)$. The periodicity property (condition 2 in the definition) ensures that if the R^{th} granule *after* $H(i)$ exists (i.e., $H(i + R) \neq \emptyset$), then it is the union of $G(a_1 + P), G(a_2 + P), \dots, G(a_k + P)$. This results in a periodic “pattern” of the composition of R granules of H in terms of granules of G . The pattern repeats along the time domain by “shifting” each granule of H by P granules of G . P is sometimes called the *Period*.

The granularities G and H in Fig. 2.4 are such that G groups periodically into H with P and R being 7 and 2, respectively.

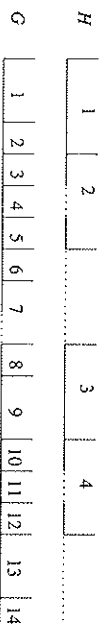


Fig. 2.4. The *periodically groups into* relationship

An instance of the “grouping pattern” is $H(1) = G(1) \cup G(2)$ and $H(2) = G(3) \cup G(5)$. This pattern repeats every seven granules of G , with the first two forming one granule of H , the third and fifth forming the next

one and the next two skipped. For example, $H(1+2) = G(1+7) \cup G(2+7)$, and $H(2+2) = G(3+7) \cup G(5+7)$.

In general, this relationship guarantees that granularity H can be finitely described (in terms of granules of G), providing the following information:

- (i) the finite sets $S_0, \dots, S_{R-1}$ of indexes of G , each one describing the composition of one of the R repeating granules of H ;
 - (ii) the value of P ;
 - (iii) the indexes of first and last granules in H , if their value is not infinite.
- In the above example, H can be described by $S_0 = \{1, 2\}$ and $S_1 = \{3, 5\}$ (two sets since $R = 2$), and $P = 7$.

The description of arbitrary granules of H can be obtained by the following formula:

$$H(j) = \bigcup_{i \in S_j \bmod R} G(P * \lfloor j/R \rfloor + i)$$

This formula applies in general, provided that $S_0, \dots, S_{R-1}$ are the sets of indexes of G describing $H(0), \dots, H(R-1)$, respectively, and H has an infinite number of granules, but it can be easily adapted to the case where H has a finite number of granules.

Many common granularities are in this kind of relationship, for example, month groups periodically into year and day groups periodically into year. It is sometimes helpful to use P and R to more precisely specify how two granularities are related, e.g., 12 months groups periodically into 1 year and 14,697 days groups periodically into 400 years. Alternatively, the relationship can also be described, say, for example, year is periodic (or 1-periodic) with respect to month, and year is periodic (or 400-periodic) with respect to day.

There exist granularities which are intuitively periodical with respect to other granularities, except for some particular spans of time where they behave in an anomalous way. The common granularity *years* when expressed in terms of *seconds* is such an example: Once in a while a *leap second* is added to a year to keep the time synchronized with the real solar year. This addition does not follow a periodical rule. In general, a granularity exception is a set of successive granules which are defined independently from other granules. It can be represented by an interval of indexes. The following definition identifies this relationship.

Definition. A granularity G groups periodically with finite exceptions into a granularity H if

1. $G \trianglelefteq H$, and
2. there exist a set of intervals $E_1, \dots, E_z$ (the granularity exceptions) and positive integers R, P , where R is less than the minimum of the number of granules of H between any two exceptions, such that for all $i \in \mathbf{Z}$, if $H(i) = \bigcup_{r=0}^k G(j_r)$ and $H(i+R) \neq \emptyset$ and $i+R < \min(E_b)$, where E_b is

the closest existing exception after $H(i)$ ¹, then $H(i+R) = \bigcup_{r=0}^k G(j_r + P)$.

Intuitively, the definition requires that all granules of H within the span of time between two exceptions have the same periodical behavior, characterized by R and P .

The granularities G and H in Fig. 2.5 are such that G groups periodically with finite exceptions into H with P and R being 2 and 1, respectively, and the exceptions being the granule of H indexed by 4 (i.e., $z = 1$ and $E_1 = [4, 4]$).

H	1	2	3	4	5	6	7	8	9
G	1	2	3	4	5	6	7	8	9

Fig. 2.5. The periodically groups with finite exceptions into relationship

Similar to periodical grouping without exception, each granularity H can be finitely described (in terms of granules of G) providing the following information:

- (i) the values of P and R ;
- (ii) for each exception $E = [\text{begin_end}]$, the definition (in terms of indexes of G) of each granule $H(\text{begin}), \dots, H(\text{end})$;
- (iii) for each “periodic” portion of H (i.e., for each set of successive granules not intersecting any exception), the definition (in terms of indexes of G) of each one of the R granules of H in one period;
- (iv) the indexes of first and last granules in H , if their values are not infinite.

A procedure to obtain the representation of arbitrary granules of H within a “periodic” portion of H can be easily devised, but it is more involved than the formula we have given for the case without exceptions and we do not report it here.

Table 2.1 summarizes the notation we have introduced.

2.2.2 Properties

We point out some properties of the set of relationships we have defined. It is easily seen that $G \sqsubseteq H$ implies $G \preceq H$, which in turn implies $G \sqsubseteq H$. Note also that $G \preceq H$ does not imply $G \trianglelefteq H$, nor vice versa. Considering the set of all possible granularities on a given time domain, $\trianglelefteq$ is an equivalence relationship inducing an equivalence class (G) for each set of shift equivalent granularities. The relationships $\sqsubseteq$, $\preceq$, $\trianglelefteq$, and $\sqsubset$ have the properties of

¹ If there is no exception after $H(i)$, then $\min(E_b)$ is taken as the maximum integer k such that $H(k) \neq \emptyset$.

$(T, \leq)$	Time Domain
G	Granularity
$G(i)$	Granule of G with index (or label) i
$G \trianglelefteq H$	G groups into H
$G \triangleleft H$	G is finer than H
$G \sqsubseteq H$	G is a subgranularity of H
$G \subsetneq H$	G is covered by H
$G \leftrightarrow H$	G is shift equivalent to H

Table 2.1. A quick reference to notation

being reflexive and transitive on that set. Finally, if we consider the set of equivalence classes induced by $\leftrightarrow$, then the relationships $\sqsubseteq$, $\trianglelefteq$, $\triangleleft$ (extended in a usual way to the equivalence classes) define partial orders.

Definition. Given a granularity order relationship Θ and a set of granularities having the same time domain, a granularity G in the set is a *bottom granularity* with respect to Θ , if $G\Theta H$ for each granularity H in the set.

For example, given the set of all granularities defined over the time domain $(\mathbf{R}, \leq)$, and the granularity relationship $\trianglelefteq$ (finer than), the granularity corresponding to the empty mapping is the bottom granularity with respect to $\trianglelefteq$. Given the set of all granularities defined over the time domain $(\mathbf{Z}, \leq)$, and the granularity relationship $\trianglelefteq$ (groups into), the granularity mapping each index into the corresponding instant (same integer number as the index) is a bottom granularity with respect to $\trianglelefteq$. An example of a set of granularities without a bottom (with respect to $\trianglelefteq$ or $\trianglelefteq$) is $\{\text{week}, \text{month}\}$. The granularity having a *chronon* for each granule, as defined in [JD98], is typically a bottom granularity.

Definition. A calendar is a set of granularities over a single time domain that includes a bottom granularity with respect to $\trianglelefteq$ (groups into).

Calendars are typically used to describe events or time-related properties over the same span of time using different granularities. For example, the Gregorian calendar comprises the granularities *day*, *month*, and *year*.

Definition. A set of granularities, having the same time domain, forms a granularity lattice with respect to a granularity order relationship Θ , if for each pair of granularities in the set, there exists a least upper bound and a greatest lower bound with respect to the relationship Θ .

Note that this is the standard definition of lattice. In the next section we will identify sets of granularities which indeed form a granularity lattice. For example, the set of granularities defined over the time domain $(\mathbf{R}, \leq)$ such that no pair of granularities is shift equivalent is a granularity lattice with respect to $\trianglelefteq$.

2.3 Granularity Conversion

When dealing with granularities, we often need to determine the granule (if any) of a granularity H that covers a given granule z of another granularity G . For example, we may wish to find the month (an interval of the absolute time) that includes a given week (another interval of the absolute time). Formally, for each index value z and granularities G and H , $\lceil z \rceil_G^H$ is undefined if $G(z) \not\subseteq H(z')$ for all z' ; otherwise, $\lceil z \rceil_G^H = z'$, where z' is the unique index value such that $G(z) \subseteq H(z')$. The uniqueness of z' is guaranteed by the monotonicity of granularities. As an example, $\lceil z \rceil_{\text{second}}^{\text{month}}$ gives the month that includes the second z . Note that while $\lceil z \rceil_{\text{month}}^{\text{second}}$ is always defined, $\lceil z \rceil_{\text{week}}^{\text{second}}$ is undefined if week z falls between two months. Similarly, $\lceil z \rceil_{\text{day}}^{\text{b-day}}$ is undefined if day z is not a business day. Note that if $G \trianglelefteq H$, then the function $\lceil z \rceil_G^H$ is defined for each index value z . For example, since $\text{day} \trianglelefteq \text{week}$, $\lceil z \rceil_{\text{day}}^{\text{week}}$ is always defined, i.e., for each day we can find the week that contains it. The notation $\lceil z \rceil^H$ is used when the source granularity can be left implicit (e.g., when we are dealing with a fixed set of granularities having a distinguished bottom granularity).

Another direction of the above transformation is as follows: Let G and H be granularities, and z an integer. Define $\lfloor z \rfloor_G^H = (z', k)$, where the resulting pair identifies the k consecutive granules of G whose union yields $H(z)$, i.e., $H(z) = G(z' + 0) \cup \dots \cup G(z' + k - 1)$. This function is useful for finding, e.g., all the days in a month. If $G \trianglelefteq H$, then $\lfloor z \rfloor_G^H$ is always defined. It is worth mentioning that in [TSQ12], it is assumed that a *calendar* subsystem provides a similar function that gives the conversion factor for a certain target granularity for each granule of a source granularity. For example, the function returns “29 days” for the month February, 1996.

The above transformation only concerns the granularities. A different kind of data conversion involves information attached to time granules. For example, we register in a database the rainfall amount for each day, record income for each year, and so on. It is sometimes desirable to obtain the rainfall amount of a month. Here, the information about rainfall is viewed as “converted” into that in terms of month and the conversion function used is summation. In Chap. 4, we propose a framework for specifying information conversion and we study related query evaluation problems.

2.4 Granularity Systems

When considering a particular application or formal context, we can specialize the general notion of granularity along the following dimensions:

- restriction on the use of the index set

- choice of the absolute time set $\mathcal{A}$
- restrictions on the structure of granules (explained below)
- restrictions on the granularities by using relationships.

We call the resulting formalization a granularity system.

Consider some possibilities for each of the above dimensions. The most relaxed choice for the index sets is to allow any integer to be the index of a granule. More restricted ones include insisting that each granule be indexed by a positive integer (i.e., $G(i) = \emptyset$ for each nonpositive integer), as it is shown in some of the applications presented in this book. The restriction to the positive integers proves to be convenient in certain application domains for representation or computational purposes, where each granularity always has a first granule. The choice for absolute time is typically between dense and discrete. In general, if the application imposes a fixed smallest granularity, such as seconds, then a discrete time domain is probably the appropriate choice. However, if one is interested in being able to represent arbitrarily finer granularities, a dense time domain (e.g., the rational numbers with the usual order) is required. Other choices, such as, for example, a continuous structure like the real numbers with the usual order, are also possible. In temporal logics, nonlinear structures are also considered as time domains, but in database applications, the linearity assumption seems to be reasonable. In both cases, specific applications could impose left/right boundedness on the absolute time set. The structure of granules could be restricted in several ways:

- (1) disallow granularities with *gaps* in a granule (b-month is an example, since each granule has gaps corresponding to weekends),
- (2) disallow granularities with *noncontiguous* granules (b-day is an example, since a granule corresponding to a Friday is not contiguous with respect to the next granule, representing a Monday),
- (3) disallow granularities whose granules do *not cover all* the absolute time (if our absolute time is between Gregorian year 0 and 2000, we disallow, for example, a granularity covering only years from 1000 to 2000), or
- (4) disallow granularities with *nonuniform* granules (only granularities with granules having the same *size* are allowed. Hence, for example, second, day, week are allowed, while month is not).

Granularities can also be restricted based on their relationships. While we formally define these relationships in the next subsection, intuitive examples are:

- (a) disallow multiple granularities that are equivalent with respect to shifting of their indices,
- (b) enforce that each pair of granularities is comparable (for example, week and month are incomparable since, for a certain week, we cannot always find a month fully including it, and vice versa).

In restricting to a granularity system, there is a tradeoff between its expressiveness and the simplicity and efficiency of the algorithms needed to manage the granularities. Here, we give the definition of a very expressive granularity system [WBB+97] that we have found to be particularly useful. We name this system General Granularities on Reals, or *GGR* for short.

Definition. A *granularity* in *GGR* is a mapping G from the set of the integers to $2^{\mathcal{R}}$ (i.e., all subsets of reals) such that $G(i) = \emptyset$ for each nonpositive integer i , and for all positive integers i and j with $i < j$, the following two conditions are satisfied:

- $G(i) \neq \emptyset$ and $G(j) \neq \emptyset$ imply that each real number in $G(i)$ is less than all real numbers in $G(j)$, and
- $G(i) = \emptyset$ implies $G(j) = \emptyset$.

The *GGR* system is defined with the following choices for the aforementioned four dimensions: Each granule in each granularity is indexed by a positive integer, while the real numbers are used as the absolute time set. There is no structural restriction on granules, but no two granularities in the system are equivalent with respect to shifting of their indices. Restriction (a) is indeed enforced by the condition that the first granule (if any) must start with index 1 (second condition in the definition of *GGR*). This system is used in all the applications illustrated in this book.

An example of a more restrictive granularity system is given in [WJS95] where a discrete absolute time is used, granules must be indexed by positive integers, and restrictions (1), (2), (3), and (a) apply. We refer to this time granularity system as Restricted Granularities on Natural Numbers. It can be easily seen that each granularity in this system corresponds to a partition of the positive integers such that each subset in the partition is a finite or infinite interval, and the first interval in the partition forms the first granule which is indexed by integer 1.

We can prove interesting formal properties for a large class of granularity systems, namely those imposing, by the above restriction (a), that no pair of different granularities can be shifting equivalent. For this class of systems, the three relationships $\preceq$, $\sqsubseteq$, and $\sqsupseteq$ are antisymmetric and, hence, each relationship is a partial order. They may not be total orders since, for example, week and month are incomparable with respect to $\preceq$, $\sqsupseteq$, and $\sqsubseteq$ (i.e., week is not finer than, does not group into, nor is a subgranularity of month, and vice versa).

We are particularly interested in the finer-than relationship. For systems enforcing restriction (a), there exists a unique least upper bound of the set of all time granularities, denoted by G_{Top} , and a unique greatest lower bound, denoted by G_{Bottom} . These top and bottom elements are defined as follows: $G_{\text{Top}}(i) = \mathcal{T}$ (the whole time domain) for some index i , and $G_{\text{Top}}(j) = \emptyset$ for each $j \neq i$, and $G_{\text{Bottom}}(i) = \emptyset$ for each index i . Moreover, for each pair of time granularities G_1, G_2 , there exist a unique least upper bound $\text{lub}(G_1, G_2)$

and a unique greatest lower bound $glb(G_1, G_2)$ of the two granularities, with respect to $\preceq$. We formalize this result referring to specific granularity systems obtained by the choices and restrictions illustrated earlier in this section (p. 20).

Theorem 2.4.1. *Any granularity system having (a) as the only restriction is a lattice with respect to the finer-than relationship.*

The proof of the above theorem shows that unique glb and lub granularities can be constructed for each pair of granularities in the system. We now show that many granularity systems, which can be obtained by applying additional restrictions among those that we have considered, are closed with respect to these glb and lub , i.e., for each pair (G, H) of their granularities, $glb(G, H)$ and $lub(G, H)$ in the general lattice of Theorem 2.4.1 are within the system. A closed granularity system obviously forms a lattice with respect to the finer-than relationship. Note that a system without restriction (a) cannot be a lattice since the presence of shifting equivalent granularities leads to the nonuniqueness of glb and lub . The fact that the index set of a granularity is infinite ensures that a sufficient number of values will be available to index granules of $glb()$ and $lub()$.

Theorem 2.4.2. *A granularity system having restriction (a) and an additional combination of restrictions (1)–(4) and (b) is closed if:*

- (2) is not in the combination unless (3) or (b) is in, and
- (4) is not in the combination unless (b) is in.

It is easily seen that if the given granularities G_1 and G_2 satisfy restriction (2) or (4), then it is not always true that $lub(G_1, G_2)$ and $glb(G_1, G_2)$ given in the general lattice of Theorem 2.4.1 satisfy the same restriction. Hence, a system with only restriction (2) or (4) is not closed. It follows from the above theorem that, among the granularity systems that are definable using the given choices (1)–(4) and (b), nonclosed systems can only be found among those enforcing restriction (2) but not restriction (3) or (b), as well as among those enforcing restriction (4) but not restriction (b). This shows that many of the systems obtained by the given choices and restrictions are closed systems, hence lattices. Note, however, that if additional restrictions to the ones considered in this paper are enforced, a system can be closed without satisfying the conditions in Theorem 2.4.2. Furthermore, a nonclosed granularity system could still form a lattice, but a particular construction for its $glbs$ and $lubs$ must be provided.

The granularity system $\mathcal{GGR}$ is an example of closed ones. The restricted granularities on natural numbers mentioned earlier form another closed system. The usefulness of the lattice structure has been shown, for example, in the logical design of temporal databases with multiple granularities (see Chap. 3), and in reasoning with temporal constraints (see Chap. 5).

2.5 Symbolic Representation

In this section, we present a symbolic representation of granularities. Note that every finite symbolic representation scheme inevitably yields some unrepresentable granularities due to the generality of the granularity definition. Indeed, for most choices of the time domain, there are uncountably many granularities. The goal of this section is to identify an intuitive symbolic representation which can capture a significant class of granularities within the formal framework.

The design of the representation scheme starts with the observation that granularities used in a calendar are not isolated, but rather are closely related. We thus design our symbolic representation based on some algebraic operations, called *calendar operations*, that capture these relationships. The symbolic representation is thus called the *calendar algebra*. Calendar operations generate new granularities by manipulating other granularities that are already generated. The relationships between the operands and resulting granularities are thus encoded in the operations. All granularities that are generated directly or indirectly from a single one (which will be the bottom granularity) form a calendar, and these granularities are related to each other through the operations that define them (from the bottom granularity). In practice, the choices for the bottom granularity include day, hour, second, microsecond and other granularities, depending on the accuracy required in each application context.

The calendar algebra consists of the following two kinds of operations: the *grouping-oriented operations* and the *granule-oriented operations*. The grouping-oriented operations group certain granules of a granularity together to form new granules in a new granularity, while the granule-oriented operations don't change the granules of a granularity, but rather make choices as to which granules should remain in the new granularity.

To simplify the algebra, we use an extended notion of granularities. More specifically, 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 for each pair of integers i and j in $\mathcal{L}$ with $i < j$, if $G(i) \neq \emptyset$ and $G(j) \neq \emptyset$, then (1) each element in $G(i)$ is less than every element of $G(j)$, and (2) for each integer k in $\mathcal{L}$ with $i < k < j$, $G(k) \neq \emptyset$. When $\mathcal{L}$ is exactly the integers, we call the granularity "full-integer labeled." Note that full-integer labeled granularities are exactly "regular" granularities, i.e., granularities as defined in Chap. 1 since labels coincide with indexes. However, in general, the set $\mathcal{L}$ of labels can be an arbitrary subset of (possibly noncontiguous) integers and these labels are used instead of indexes to identify granules. Note that each labeled granularity can use a different set of labels.

To simplify the discussion, we will still use G to denote labeled granularities when no confusion arises.

Certain calendar operations will only work on full-integer labeled granularities, while others will be more easily defined and implemented using more

flexible labelings. The calendar algebra can be easily adapted to generate only full-integer labeled granularities as well as granularities with restricted use of integers as indices for granules. The idea is to use the subset operation and a “relabel” operation as the last steps to obtain the desired granularity. For example, for granularity systems where each granule has to be indexed by a positive integer with the first granule indexed by 1, the subset operation will select a subset of the granules in which the first granule exists, and the relabel operation will simply label the first granule as 1, the second as 2, and so on. We will come back to this issue later (see Sect. 2.5.6).

To define the calendar algebra, a *label-aligned subgranularity* relationship is needed. Formally, G_1 is a label-aligned subgranularity of G_2 if the label set $\mathcal{L}_1$ of G_1 is a subset of the label set $\mathcal{L}_2$ of G_2 and for each i in $\mathcal{L}_1$ such that $G_1(i) \neq \emptyset$, we have $G_1(i) = G_2(i)$. Figure 2.6 shows an example of this relationship.

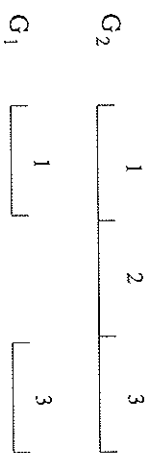


Fig. 2.6. G_1 is a label-aligned granularity of G_2

Note that the difference (other than using a different definition of granularities) from the “usual” subgranularity relationship is that here we require that each granule of G_1 uses exactly the same label as the one used by the corresponding granule in G_2 .

2.5.1 The Grouping-Oriented Operations

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 granularity such that for each integer i ,

$$G'(i) = \bigcup_{j=(i-1) \cdot m + 1}^{i \cdot m} G(j).$$

For example, given granularity **day**, **week** can be generated by **week** = $Group_7(\text{day})$ if we assume that **day(1)** corresponds to **Monday**, i.e., the first day of a week. Similarly, given **month**, **year** can be generated by **year** = $Group_{12}(\text{month})$ if we assume **month(1)** corresponds to a **January**. Note that G partitions G' , and G' is also a full-integer labeled granularity.

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 $1 \leq l \leq m$. The altering-tick operation $Alter_{l,k}^m(G_2, G_1)$ generates a new 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 goal of the altering-tick operation is to modify the granules of G_1 so that the l -th granule of every aforementioned group will have $|k|$ additional (or fewer when $k < 0$) granules of G_2 . For example, if G_1 represents 30-day groups (i.e., $G_1 = Group_{30}(\text{day})$) and we want to add a day to every 12-th month (i.e., to make December to have 31 days), we may perform $Alter_{12,1}^{12}(\text{day}, G_1)$.

More specifically, for all $i = l + m \cdot n$, 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) by taking in (or pushing out) later granules of G_2 , and the effect is propagated to later granules of G_1 . On the contrary, when $i \leq 0$, $G_1(i)$ expands (or shrinks) 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 $G^l = Alter_{l,k}^m(G_2, G_1)$ is the granularity such that for each integer i , let $G'(i) = \emptyset$ if $G_1(i) = \emptyset$, and otherwise let

$$G'(i) = \bigcup_{j=\psi_i}^{t_i} G_2(j),$$

where

$$b'_i = \begin{cases} b_i + (h-1) \cdot k, & \text{if } i = (h-1) \cdot m + l, \\ b_i + h \cdot k, & \text{otherwise,} \end{cases}$$

$$t'_i = t_i + h \cdot k,$$

and

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

Figure 2.7 shows an example of a grouping operation and an altering-tick operation. Granularity G' is defined by $G' = Group_5(G)$, while granularity G'' is defined by $G'' = Alter_{2,-1}^2(G, G')$, which means shrinking the second one of every two granules of G' by one granule of G .

Note that the grouping operation is a special case of the altering-tick operation. Indeed, $Group_m(G) = Alter_{1,m-1}^1(G, G)$, i.e., grouping every m granules together is the same as expanding every granule by $m-1$ granules.

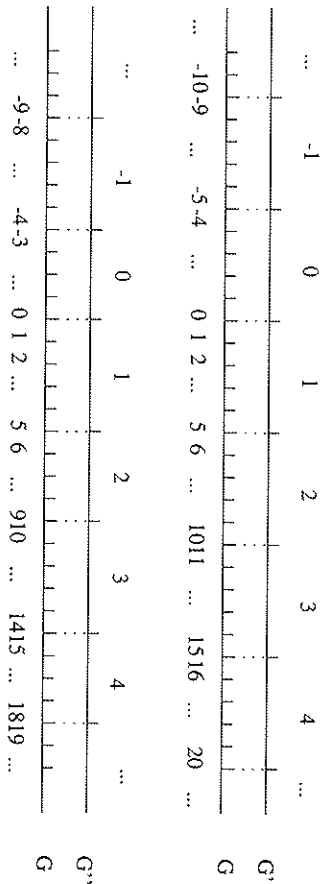


Fig. 2.7. Grouping operation and altering-tick operation

However, we keep grouping operation in the calendar algebra because of its conceptual and computational simplicity.

An extension of the above operation is also used: When the parameter m is infinity (∞), the altering-tick operation $\text{Alter}_{i,\infty}^k(G_2, G_1)$ means only altering the granule $G_1(i)$. This reflects the concept of *finite exceptions*. In this case, the parameter l can be any integer (instead of only a positive integer) so that any granule of G_1 can be altered. For example, to add a leap second to the last minute of 1998, we may use $\text{Alter}_{x,\infty}^1(\text{second}, \text{minute})$, where x is the label of the last minute of 1998.

Shifting operation. Let G be a full-integer labeled granularity, and m an integer. The shifting operation $\text{Shift}_m(G)$ generates a new granularity G' by shifting the labels of G by m positions. For each integer i , the granule $G'(i)$ will be the granule $G(i+m)$. More formally, $G' = \text{Shift}_m(G)$ is the granularity such that for each integer i , $G'(i) = G(i+m)$. Note that G is shift equivalent to G' , and G' is also full-integer labeled. The shifting operation can easily model time differences. Suppose granularity GMT-hour stands for the hours of Greenwich Mean Time. Then the hours of US Eastern Time can be generated from GMT-hour by

$$\text{USEast-Hour} = \text{Shift}_{-5}(\text{GMT-hour}).$$

Note. The grouping, altering-tick and shifting operations are collectively called *basic operations*. These basic operations are restricted to operate on full-integer labeled granularities (i.e., “regular” granularities), and the granularities generated by these operations are still full-integer labeled ones.

Combining operation. Let G_1 and G_2 be granularities with label sets $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. The combining operation $\text{Combine}(G_1, G_2)$ 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

$$s(i) = \{j \in \mathcal{L}_2 \mid \emptyset \neq G_2(j) \subseteq G_1(i)\}.$$

Then $G' = \text{Combine}(G_1, G_2)$ 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 $\mathcal{L}_{G'}$,

$$G'(i) = \bigcup_{j \in s(i)} G_2(j).$$

As an example, given granularities b-day and month, the granularity for business months can be generated by

$$\text{b-month} = \text{Combine}(\text{month}, \text{b-day}).$$

Anchored grouping operation. Let G_1 and G_2 be granularities with label sets $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, where G_2 is a label-aligned subgranularity of G_1 , and G_1 is a full-integer labeled granularity. The anchored grouping operation $\text{Anchored-group}(G_1, G_2)$ 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 formally, $G' = \text{Anchored-group}(G_1, G_2)$ 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 a resulting granule by the anchored grouping operation.

For example, each academic year at a certain university begins on the last Monday in August, and ends on the day before the beginning of the next academic year. Then, the granularity corresponding to the academic years can be generated by

$$\text{AcademicYear} = \text{Anchored-group}(\text{day}, \text{lastMondayOfAugust}).$$

2.5.2 Granule-Oriented Operations

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 $m \leq n$. The subset operation $G' = \text{Subset}_m^n(G)$ generates a new granularity G' by taking all the granules of G whose labels are between m and n . More formally, $G' = \text{Subset}_m^n(G)$ is the granularity with the label set $\mathcal{L}_{G'} = \{i \in \mathcal{L} \mid m \leq i \leq n\}$, and for each $i \in \mathcal{L}_{G'}$,

$$G'(i) = G(i).$$

For example, given granularity year, all the years in the 20th century can be generated by

20CenturyYear = Subset¹⁹⁹⁹₁₉₀₀(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 the extensions of setting $m = -\infty$ or $n = \infty$ with semantics properly extended.

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 and 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 $S = \{j_1, j_2, \dots, j_n\}$, where $j_1 < j_2 < \dots < j_n$. For each $i \leq 0$, 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 $l > 0$, $\Delta'_k(S)$ denotes the subset of S defined as follows:

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

Therefore, $\Delta'_k(S)$ 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 beginning of the list counting forward (or the $|k|$ -th one from the end of the list counting backward if $k < 0$). For example,

$$\Delta_3^2(\{1, 2, 3, 4, 5, 6, 7\}) = \{3, 4\} \text{ and } \Delta_{-7}^2(\{1, 2, 3, 4, 5, 6, 7\}) = \{1\}.$$

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 $\Delta'_k(\cdot)$ operator.

Select-down operation. For each granule $G_2(i)$, there exists a set of granules of G_1 that is contained in $G_2(i)$. The operation *Select-down* _{k} (G_1, G_2) , where $k \neq 0$ and $l > 0$ are integers, selects granules of G_1 by using $\Delta'_k(\cdot)$ on each set of granules (actually their labels) of G_1 that are contained in one granule of G_2 . More formally, $G' = \text{Select-down}_k(G_1, G_2)$ is the granularity with the label set

$$\mathcal{L}_{G'} = \cup_{i \in \mathcal{L}_2} \Delta'_k(\{j \in \mathcal{L}_1 \mid \emptyset \neq G_1(j) \subseteq G_2(i)\}),$$

and for each $i \in \mathcal{L}_{G'}$,

$$G'(i) = G_1(i).$$

For example, Thanksgiving days are the fourth Thursdays of all Novembers. If granularities Thursday and November are given, it can be generated by

$$\text{Thanksgiving} = \text{Select-down}_4(\text{Thursday}, \text{November}).$$

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, $G' = \text{Select-up}(G_1, G_2)$ 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 Thanksgiving, the weeks that contain Thanksgiving days can be defined by

$$\text{Thankweek} = \text{Select-up}(\text{week}, \text{Thanksgiving}).$$

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 intersecting $G_2(i)$.

The operation *Select-by-intersect* _{k} (G_1, G_2) , where $k \neq 0$ and $l > 0$ are integers, selects granules of G_1 by applying $\Delta'_k(\cdot)$ operator to all such sets, generating a new granularity G' . More formally, $G' = \text{Select-by-intersect}_k(G_1, G_2)$ is the granularity with the label set

$$\mathcal{L}_{G'} = \cup_{i \in \mathcal{L}_2} \Delta'_k(\{j \in \mathcal{L}_1 \mid G_1(j) \cap G_2(i) \neq \emptyset\}),$$

and for each $i \in \mathcal{L}_{G'}$,

$$G'(i) = G_1(i).$$

For example, given granularities week and month, the granularity consisting of the first week of each month (among all the weeks intersecting the month) can be generated by

$$\text{FirstWeekOfMonth} = \text{Select-by-intersect}_1(\text{week}, \text{month}).$$

Again, G' is a label-aligned subgranularity of G_1 .

Set operations. The set operations are based on the viewpoint that each granularity is a set of granules. Considering each granularity as a set with granules being the elements, we may apply the set operations (i.e., union, intersection, and difference) to existing granularities and generate new granularities. However, not all granularities can be used in set operations. For example, if we union the granularity week and month, we will not get a valid resulting granularity whatever label sets we use for them. This is because granules of two granularities are not "compatible" in the sense that granules may overlap.

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 can be 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

$$\text{WeekendDay} = \text{Sunday} \cup \text{Saturday}.$$

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 can be seen from the transitivity of the label-aligned subgranularity relationship (proof is left to the reader).

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 $i \in \mathcal{L}'$,

$$G'(i) = G_1(i) \text{ (or equivalently } G_2(i)).$$

For example, given the granularity day1, which includes exactly the first day of every month, and the granularity Monday, the granularity for the days that are both the first day of a month and a Monday can be generated by

$$\text{Monday\&day1} = \text{Monday} \cap \text{day1}.$$

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.

Given granularities Weekday and FederalHoliday,

$$\text{BusinessDay} = \text{Weekday} - \text{FederalHoliday}.$$

Note that G' is a label-aligned subgranularity of G_1 .

2.5.3 Syntactic Restrictions on Algebra Operations

The granularities participating in a calendar operation usually have to satisfy certain conditions. For example, in the altering-tick operation, the first operand granularity must partition the second one, and the set operations only apply to granularities that are label-aligned subgranularities of a common one. Checking these preconditions can be difficult.

Our solution is to use a syntactic restriction, namely to use the explicit relationship derived from the operations themselves. 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. To ensure that a granularity is a full-integer labeled one, our syntactic restriction requires that it is either the bottom granularity, or derived from the bottom granularity by only using (maybe repeatedly) the basic operations, namely the grouping, altering-tick and shifting operations. In order to ensure the conditions (2) and (3) above, we use Table 2.2. The entries in the table were already mentioned when the operations were defined earlier.

Operation	Relationships
Grouping	operand partitions result
Altering-tick	operand 2 partitions result
Shifting	operand partitions result, and vice versa
Combining	none
Anchored grouping	none
Subset	result is a label-aligned subgranularity of operand
Selecting operations	result is a label-aligned subgranularity of operand 1
Set operations	If both operands are label-aligned subgranularities of H , then result is a label-aligned subgranularity of H .

Table 2.2. Partition and label-aligned subgranularity relationships

The above syntactic restriction actually gives a classification of the granularities that can be generated from the calendar algebra. The granularities can be seen as organized into three layers. The membership of a granularity in a layer is determined by the operations and the operands used to define it. Fig. 2.8 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 (may be repeatedly) the basic operations (grouping, altering-tick and shifting). *Layer 2* consists of the granularities that are the result of applying (may be repeatedly) the subset operation and the selecting operations on the full-integer labeled granularities in the first layer. Note that the second operand used in the selecting

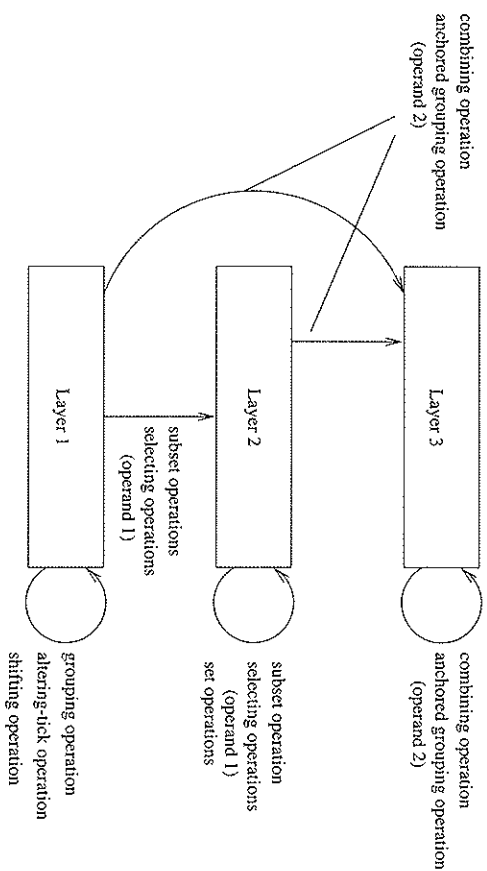


Fig. 2.8. Transition between the three layers

operations can be a granularity in any layer. *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.

Granularities in the three layers have distinct properties. All the granularities in layer 1 are full-integer labeled granularities. All granularities in layer 2 may not be full-integer labeled ones, but there is no gap within each granule of every granularity, i.e., each granule is an interval of granules of the bottom granularity. The granularities in layer 3, however, may contain gaps within a granule.

2.5.4 Examples

In this subsection, we present some more examples of granularities represented by the calendar algebra. We assume *second* is the bottom granularity. We further assume that *second(1)* is the first second of January of year 1 (hence the first second of a month, day, etc.), as well as the first second of a week. In the current Gregorian calendar, this assumption is correct.² Now we have the following.

- *minute* = *Group*₆₀(*second*),

² The “cal” program in the Unix operating system contains an error which does not calculate the leap years appropriately (e.g., year 1000 should not be a leap year, but the “cal” program says it is). Hence, if you check the week day of January first of year 1, “cal” tells you a wrong answer.

- *hour* = *Group*₆₀(*minute*),
 - *day* = *Group*₂₄(*hour*),
 - *week* = *Group*₇(*day*),
 - *pseudomonth* = *Alter*_{1,-1}¹²(*day*, *Alter*_{6,-1}¹²(*day*, *Alter*_{6,-1}¹²(*day*, *Alter*_{6,-1}¹²(*day*, *Alter*_{4,-1}¹²(*day*, *Alter*_{2,-3}¹²(*day*, *Group*₃₁(*day*))))), where *pseudomonth* is generated by grouping 31 days, and then shrink April (4), June (6), September (9) and November (11) by one day, and shrink February (2) by 3 days,
 - *month* = *Alter*_{2+12*389,1}^{12*100}(*day*, *Alter*_{2+12*99,-1}^{12*100}(*day*, *Alter*_{2+12*3,1}^{12*4}(*day*, *pseudomonth*))), where the February of each leap year is adjusted appropriately;
 - *Monday* = *Select-down*₁¹(*day*, *week*),
 ...
 - *Sunday* = *Select-down*₇¹(*day*, *week*).

2.5.5 Granularity Conversion

Given two granularities G_1 and G_2 , we are interested in the conversion functions $\lceil z \rceil_{G_1}^{G_2}$ and $\lfloor z \rfloor_{G_1}^{G_2}$. We assume that all granularities used in an application are generated from a single bottom granularity by using the calendar algebra. Thus, there always exists a single granularity such that G_1 and G_2 are derived from it. This common single “source” granularity H for G_1 and G_2 , however, need not be the bottom one. Indeed, in the example given in the previous subsection, month and week are derived from day instead of the bottom granularity *second*.

If G_1 and G_2 are both derived from the single granularity H , then to perform functions $\lceil z \rceil_{G_1}^{G_2}$ and $\lfloor z \rfloor_{G_1}^{G_2}$, we only need to consider the special case $\lceil z \rceil_H^{G_1}$ and $\lfloor z \rfloor_H^{G_1}$. Indeed, consider $\lceil z \rceil_{G_1}^{G_2}$. We first perform $\lceil z \rceil_H^{G_1}$, and then do $\lceil z' \rceil_H^{G_2}$ for each z' in $\lfloor z \rfloor_H^{G_1}$. If each $\lceil z' \rceil_H^{G_2}$ yields the same result j , then $\lceil z \rceil_{G_1}^{G_2} = j$, and undefined otherwise. Consider now $\lfloor z \rfloor_{G_1}^{G_2}$. We first perform $\lfloor z \rfloor_H^{G_1}$, and denote the result as S . Then we perform $\bigcup_{z' \in S} \lceil z' \rceil_H^{G_2}$, and denote the result as S' . Then S' is the result of $\lfloor z \rfloor_{G_1}^{G_2}$ if $\bigcup_{z' \in S'} \lfloor z' \rfloor_H^{G_2} = S$, and undefined otherwise.

The correctness of the above procedures is easily seen by observing the fact that granularity H groups into both G_1 and G_2 . We omit the proof here, but instead consider the following example. Suppose year and month are both defined on day, *year(1)* corresponds to year 1970, and *month(1)* corresponds to January of 1970. To find $\lceil 144 \rceil_{\text{month}}^{\text{year}}$, i.e., find the year in which the 144-th month falls in, we only need to find out all the days this 144-th month is composed of (by using $\lceil 144 \rceil_{\text{day}}^{\text{month}}$), and then from these days, we can find the year 1981 (by applying $\lceil \text{day} \rceil_{\text{year}}$ on all the days in the 144-th month) since each of these days falls into the same year. Conversely, to find $\lfloor 1981 \rfloor_{\text{month}}^{\text{year}}$, we will (1) find all the days of 1981 (by using $\lfloor 1981 \rfloor_{\text{day}}^{\text{year}}$), (2) find all the months

(by applying $\prod_{\text{day}}^{\text{month}}$ on each day found in the previous step), and (3) test to see if all these months together will give us exactly the same days when we apply $\prod_{\text{day}}^{\text{month}}$.

It is easily seen from the definitions of the calendar algebra operations that if G is derived from a single granularity H by using the calendar algebra, then either G has only a finite number of granules, or H groups periodically into G , or H groups periodically with finite exceptions into G . Therefore, we may represent the granularities using the enumeration method outlined in Sect. 2.2.1, and by knowing the enumeration representation of the operands, it is not difficult to derive the enumeration representation of the result. We leave the details to the readers.

In many cases, a more efficient method for granularity conversion is possible. For example, if we know the definition of the week in terms of day (e.g., $\text{week} = \text{Group}_7(\text{day})$), then it is much more efficient to use this direct relationship to convert between week and day than using the enumeration. Another example is the relationship between year and day. To use the enumeration method, we need to represent one whole period of year in terms of day, which is 400 years. By using the definition of the year in terms of day, it is easy to come up with a conversion algorithm which is close to what a "common" computer program does, namely to figure out the conversion as if no leap year exists, and then adjust the numbers by taking leap years into account. These efficient methods of conversion are interesting topics for further studies.

2.5.6 Accommodating Restrictions on Index/Label Sets

A granularity as defined in the introduction uses the set of the integers as the index set while a granularity generated by the calendar algebra uses a subset of the integers as its label set, and each granularity can use a different one. Furthermore, a granularity system may only allow granules to be indexed by positive integers. In order to generate granularities in the corresponding systems, the algebra needs to be extended correspondingly.

This extension is not difficult via the following method. We first define the granularities using the algebra as it is, and then relabel the granules. We introduce the following relabeling operation for this purpose.

Relabeling. Let G be a labeled granularity, i and j integers with i in the label set of G such that $G(i) \neq \emptyset$. The relabeling operation $G' = \text{Relabel}_i^j(G)$ generates a full-integer labeled granularity G' by relabeling $G(i)$ as $G'(j)$ and relabel the next (and previous) granule of G by the next (previous, respectively) integer. More formally, for each integer k , if $k = j$, then let $G'(k) = G(i)$, and otherwise let $G'(k) = G(i')$, where $G(i')$ is the $|j - k|$ -th granule of G after (before, respectively) $G(i)$. If the required $|j - k|$ -th granule of G does not exist, then let $G'(k) = \emptyset$. Note that G' is always a full-integer labeled granularity.

As an example, assume $\text{Monday} = \text{Select-down}_1^1(\text{day}, \text{week})$. This granularity Monday does not label all the Mondays contiguously. Then the operation $\text{Relabel}_1^1(\text{Monday})$ gives the granularity which labels the $\text{Monday}(1)$ as 1, the first Monday after $\text{Monday}(1)$ as 2, and so on. The Mondays before the $\text{Monday}(1)$ are labeled 0, -1, and so on.

For a granularity system where granules must be indexed by positive integers, we may use the subset operation (with $n = \infty$) to make sure that there is always a first granule in the generated granularity. After that, a relabel operation will be able to label granules by only positive integers (by using a positive integer j). In the above example regarding Mondays, we may use $\text{Relabel}_1^1(\text{Subset}_1^\infty(\text{Monday}))$ to obtain the granularity which labels the first Monday as 1, the second Monday as 2 and so on.

Conversion of granularities that are relabeled can be solved via a *next* function. Let G be a granularity, i and $n \neq 0$ integers. Then let $\text{Next}_G(i, n) = j$ such that $G(j)$ is the $|n|$ -th granule among the granules of G which have labels greater (less if $n < 0$) than i . If no such granule exists, then $\text{Next}_G(i, n)$ is undefined. If $n = 0$, then let $\text{Next}_G(i, n) = i$ if $G(i)$ is a granule of G , and undefined otherwise. Basically, this next function provides the label value for the next (or previous) $|n|$ -th granule of G .

When the above *Next* function is given, the relabeling of granularities can be done easily. For example, assume that $G(1)$ is a granule of G . Then $\text{Relabel}_1^1(G)$ gives the granularity G' such that for each integer i , $G'(i) = G(\text{Next}_G(1, i))$ if $\text{Next}_G(1, i)$ is defined, and $G'(i) = \emptyset$ otherwise.

The granularity conversions $\prod_H^G$ and $\prod_G^H$ can be easily done for the relabeled granularities when the *Next* function is known. In order to compute the *Next* function, we can again use the enumeration representation of the granularities. As for the $\prod_H^G$ and $\prod_G^H$ conversions, many optimization can be applied for special cases.

2.6 Expressiveness and Alternative Representations

As mentioned earlier, the calendar algebra is not able to generate all possible granularities since the number of granularities can be uncountably many. In this section, we characterize the expressiveness of the algebra and compare its expressiveness with that of other symbolic representation formalisms.

Most practical problems seem to require only a granularity system containing a set of time granularities which are all periodical with respect to a *bottom* granularity. In this section we consider a generic calendar with the bottom granularity covering the whole time domain.

Definition. We say that a granularity G is periodical in a calendar if it is periodical with respect to the bottom granularity of that calendar.

A similar definition holds for granularities periodical with finite exceptions.

In Fig. 2.9 we represent the whole set of granularities definable in a generic calendar according to our definitions. They are partitioned in two main subsets: those having all granules with contiguous values (NO-GAP) and those admitting granules with noncontiguous values (GAP). The inner circle identifies finite and periodical granularities: finite granularities are divided (dash line) into finite irregular and finite periodical³ while infinite periodical granularities are divided into those having a first granule and no last granule (INFINITE-R), those having a last granule and no first granule (INFINITE-L), and those infinite on both sides (INFINITE). This classification will be useful when considering the expressive power of symbolic formalisms.

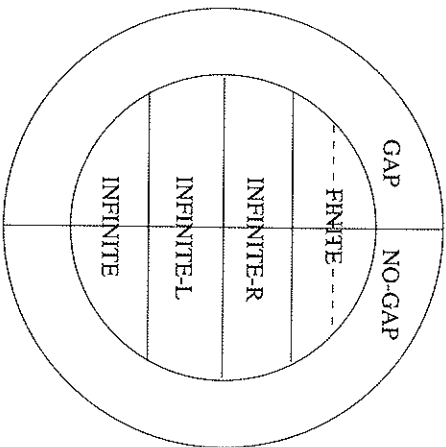


Fig. 2.9. A classification of time granularities in a calendar

Theorem 2.6.1. *The calendar algebra given in Sect. 2.5 can represent all the granularities which are either periodical or finite nonperiodical.*

In addition to the periodical and finite nonperiodical granularities, the calendar algebra can also represent other granularities, including any granularity G which is periodical with finite exceptions, i.e., any granularity G such that the bottom granularity groups periodically with finite exceptions into G . In order to avoid the finite exceptions, we may disallow the extensions to the altering-tick operations and only allow the Subset operation to be used as the last step of deriving a granularity. Details are omitted and left to the reader. Note that a union of a granularity in INFINITE-R with another in INFINITE-L (see Fig. 2.9), when allowed, may produce a granularity with different periodic characteristics on the two sides (one side goes to $+\infty$ and the

³ Despite this formal distinction, finite granularities will be treated uniformly in the results.

other to $-\infty$). If a calendar system does not desire these special granularities, then appropriate restriction on the use of the subset and union operations can be applied. Again, details are omitted. The granularities captured by the calendar algebra are shown in Fig. 2.10.

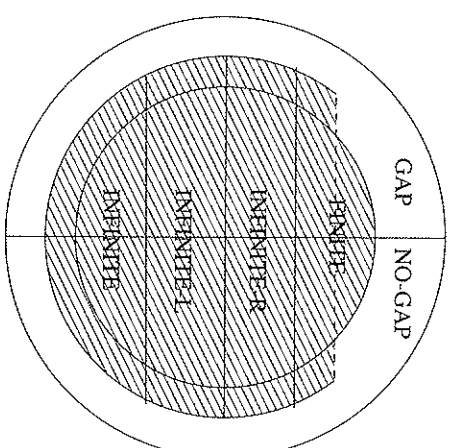


Fig. 2.10. The subset of granularities captured by the calendar algebra

2.6.1 Alternative Representations

A symbolic formalism, based on collections of temporal intervals, was proposed in [LMF86] to represent temporal expressions occurring in natural language and used in several application domains like appointment scheduling and time management. This formalism has been adopted with some extensions by many researchers in the AI [Koo91, Ter95, CD98] and Database area [CSS94, CPP+99]. From the deductive database community, a second influential proposal is the *slice* formalism introduced in [NS92], and adopted, among others, in [BBF+98].

2.6.2 Collections and Slices

The temporal intervals collection formalism was proposed in [LMF86]. A *collection* is a structured set of intervals where the order of the collection gives a measure of the structure depth: an order 1 collection is an ordered list of intervals, and an order n ($n > 1$) collection is an ordered list of collections having order $n - 1$. Each interval denotes a set of contiguous moments of time. For example, the collection of Month, where each month is represented as the collection of days in that month, is a collection of order 2. In order to provide a user-friendly representation of collections, the authors introduce

two classes of operators on collections and the notion of *primitive collection*.⁴ A primitive collection is defined as an order 1 collection formed by an infinite number of meeting⁵ intervals which may start from a specific one. The two classes of operators are called *dicing* and *slicing*. A dicing operator allows us to further divide each interval within a collection into another collection. For example, *Week: during: January1998* divides the interval corresponding to January1998 into the intervals corresponding to the weeks that are fully contained in that month. Other dicing operators are allowed, adopting a subset of Allen's interval relations [All83]. Slicing operators provide means of selecting intervals from collections. For example, *[1, -1]/Week: during: January1998* selects the first and last week from those identified by the dicing operator above. In general, slicing can be done using a list of integers, as well as with the keyword *the*, which identifies the single interval of the collection (if it is single), and the keyword *any*, which gives nondeterministically one of the intervals. Collection expressions can be arbitrarily composed using these two classes of operators starting from primitive collections, which are explicitly specified either by a periodic set of intervals, or as a grouping of intervals from previously defined primitive collections.

The slice formalism was introduced in [NS92] as an alternative to the collection formalism in order to have an underlying evaluation procedure for the symbolic expressions.

The base building blocks for expressions are periodic infinite sets of consecutive intervals, which we call here *simple granularities*.⁶ The only difference with respect to primitive collections is that they don't have first nor last interval, and intervals in a simple granularity are indexed by consecutive integers. Once a simple granularity is given in terms of the time domain, others can be defined dynamically from existing ones by the construct *Generate(sp, C, l₁, ..., l_m)* which generates a new simple granularity with m intervals in each period, the first one obtained grouping l_1 granules of simple granularity C , starting from $C(sp)$, the second grouping the successive l_2 granules, and so on, with $l_1, \dots, l_m$ treated as a circular list. A (finite or infinite) set of not necessarily consecutive intervals and denoting a slice is a symbolic expression built from simple granularities and denoting a $\sum_{i=1}^n O_i.C_i \triangleright D$ where the sum identifies the starting points of the intervals and D their duration. Each C_i is a symbol denoting a simple granularity and O_i is either a set of natural numbers or the keyword *all*. If the sum is simply $O_1.C_1$, it denotes the starting points of the intervals of C_1 whose index belongs to O_1 , or the starting points of all intervals if $O_1 = \text{all}$. If the sum is $\sum_{i=1}^{n-1} O_i.C_i + O_n.C_n$ with $O_n = \{o_n\}$, it denotes the starting points of the o_n -th interval of C_n following each point in $\sum_{i=1}^{n-1} O_i.C_i$. For example, the sum *all.Year + {2, 4}.Month + {1}.Day* denotes the set of points corre-

sponding to the beginning of the first day of February and April of each year. The duration D has the form $h.C_d$ where C_d is a symbol denoting a simple granularity such that $C_d \leq C_n$,⁷ and h is the number of successive intervals of C_d specifying the duration. Hence, the slice *all.Year + {2, 4}.Month + {1}.Day > 2.Day* denotes a set of intervals corresponding to the first two days of February and April of each year.

We refer the reader interested in a more comprehensive presentation of the formalisms to the respective papers [LMF86, NS92].

2.6.3 Expressiveness and Relationships

Both collections and slices essentially characterize periodic sets. Similar to granularities, even in these formalisms, there is the notion of a bottom granularity among the primitive collections/simple granularities, which defines the finest time units in the domain. Without loss of generality, in the following, we assume that C is the bottom granularity for our general calendar and for their expressions. A *period*, in terms of C can be associated with each slice expression S as well as with any collection expression E . Intuitively, the period indicates the number of instances of C after which the same pattern of intervals denoted by the expression is repeating; each interval in a period can be obtained by a constant shift of the corresponding interval in another period. If $C_1, \dots, C_n$ are the simple granularities appearing in the expression, then the period is the least common multiple of $Period(C_i/C)$. Technically, $Period(C_i/C)$ is defined as $\sum_{j=1}^{\xi} elem(j, Duration(C_i/C))$, where $Duration(C_i/C)$ is a list of integers, each one denoting the duration of an interval of C_i in terms of C , $\xi = length(Duration(C_i/C))$, $elem(j, list)$ returns the j -th element of the *list*, and $length(list)$ returns the number of elements in the *list*. For example, $Duration(Year/Day) = \langle 365, 365, 365, 366 \rangle$,⁸ and hence, $Period(Year/Day) = 1461$.

We now consider the expressiveness of slice expressions with respect to the formal notion of granularity we have given. If all the intervals denoted by a slice S are disjoint, we call S a *disjoint slice*.

We have shown that, once we have fixed the bottom granularity, disjoint slices can represent all and only no-gap finite, and no-gap infinite periodical granularity, as illustrated in Fig. 2.11. Nondisjoint slices, by definition, do not represent granularities at all. Unfortunately, it seems that there is no way to enforce disjointness by simple syntax restrictions.

Technically, if $S = \sum_{i=1}^n O_i.C_i \triangleright h.C_d$ is an infinite slice ($O_1 = \text{all}$), we have an algorithm to derive the intervals $\{[r_1, r_1 + H_1 - 1], \dots, [r_n, r_n + H_n - 1]\}$, where H_i is the length in terms of the bottom granularity C corresponding to h granules of C_d , starting at r_i . These intervals are the ones denoted by S within a slice period. Then, a periodical granularity G can be defined by

⁴ The authors in [LMF86] call this primitive collection a *calendar*.

⁵ Interval I_1 meets interval I_2 if I_2 starts when I_1 finishes.

⁶ The authors in [NS92] call them *calendars*.

⁷ According to our definition of the relationship $\leq$.

⁸ Ignoring exceptions to leap years

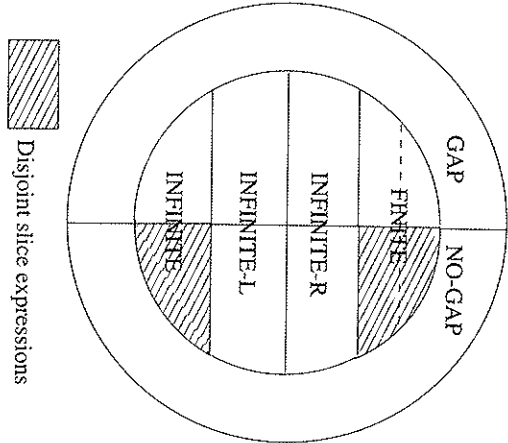


Fig. 2.11. The subset of granularities captured by the slice formalism

taking $R = n$, $P = \text{Period}$, where Period is the slice period in terms of C , and $G(i) = \bigcup_{x=x_i}^{x_i+H_i-1} C(x)$ for each $i = 1, \dots, n$. It is shown that G is equivalent to S . When S is finite, the same algorithm can be easily adapted to derive all the intervals S denotes. Then, the equivalent granularity is simply defined explicitly mapping each granule to one of these intervals. Disjointness ensures that the result of this mapping is indeed a granularity.

Example 2.6.1. Let $S = \text{all}.\text{Week} + \{2, 3\}.\text{Day} \triangleright 12.\text{Hour}$ be an infinite slice and Hour be the bottom granularity. The slice Period is 168 hours (the number of hours in a week) and in the period containing $\text{Hour}(1)$ the slice denotes the set of intervals $\{[25, 36], [49, 60]\}$. The periodical granularity G , equivalent to S , is defined by taking $R = 2$ (the number of intervals in a period), $P = \text{Period}$, $G(1) = \bigcup_{x=25}^{x=36} \text{Hour}(x)$ and $G(2) = \bigcup_{x=49}^{x=60} \text{Hour}(x)$.

INFINITE-R and INFINITE-L granularities cannot be represented by a slice, since the only way to denote an infinite set of intervals with a slice is to have $O_1 = \text{all}$, and there is no way within the slice formalism to impose a minimum or a maximum on that set.⁹

We now consider the collection formalism. Any collection E resulting from the application of a dicing or slicing operator is such that any two intervals t and u contained in E are either equal or disjoint. This follows from the semantics of the operators, and from the fact that each primitive collection contains only disjoint intervals.

⁹ Note however, that the addition of a *reference interval* (bound) to each slice, as used in [BBF+98], provides an easy extension to capture all no-gap periodical granularities.

We have shown in [BD99] that, once the bottom granularity is fixed, collection expressions can represent all no-gap periodical or finite nonperiodical granularities, as illustrated in Fig. 2.12.

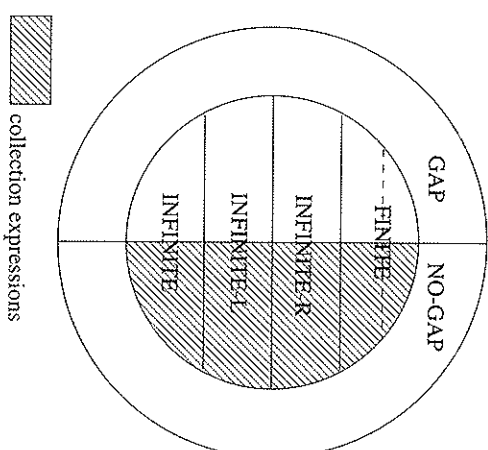


Fig. 2.12. The subset of granularities captured by the collection formalism

Similar to the case of slices, we developed an algorithm to parse the expression, to derive its period, the intervals it denotes within the period,¹⁰ and lower/upper bounds if present. Once the intervals are derived, we have all the data that is needed to define the granularity G , since it will have the same period, the intervals within the period define the corresponding granules, and the lower/upper bounds are used to impose a starting/ending granule.

Example 2.6.2. Consider $E = \{1/\text{Monday}:\text{during}:\text{Year}.\triangleright.2000\}$. This collection expression identifies an order 1 collection that contains all first Mondays of each year starting from Monday, January 1st 2001. We assume Day is the bottom granularity with $\text{Day}(1) = \text{Saturday, Jan 1st 2000}$. We first have to compute the expression period. Since Monday is defined as $1/\text{Day}:\text{during}:\text{Week}$ with the periods of Day and Week equal to 1 and 7 respectively, $\text{lcm}(1, 7) = 7$ is the period computed for Monday. Similarly, since the period for Year with respect to the bottom granularity is 1461 (4 years in Day), the whole expression period is computed as $\text{lcm}(7, 1461) = 10227$ (28 years in Day). Then, G is defined as having period $P = 10227$, $R = 28$ (the number of granules in each period), $G(1) = \text{Day}(367)$ (2001/1/1), $G(2) = \text{Day}(738)$ (2002/7/1), ..., $G(28) = \text{Day}(10228)$ (2028/3/1), and $G(i) = \emptyset$ for $i < 1$. To obtain these intervals the algorithm first restricts

¹⁰ The intervals may be structured in a collection of order higher than 1, but this is irrelevant with respect to the time granules that the expression denotes.

years to those after 2000, then it represents all Mondays within those years, and in the end it extracts the intervals corresponding to the first Monday.

Note that in this case, all granularities in the right side of the inner circle of Fig. 2.12 are captured. We can conclude that slices and collections have incomparable expressiveness, since slices can represent sets of overlapping intervals, and collections can represent INFINITE-R and INFINITE-L periodical granularities. From the above results, it is clearly possible to translate from one formalism to the other, when considering expressions denoting FINITE or INFINITE granularities, but it seems to be difficult to devise general rules to translate at the symbolic level, preserving the intuitiveness of the expression. Indeed, despite the fact that the + operator in slices may be intuitively interpreted as equivalent to : during : in collections, they actually have a different semantics.

The collection formalism has been extended with some additional operators in [CSS94]. In particular, control statements if-then-else and while are introduced to facilitate the representation of certain sets of intervals, as for example, *the fourth Saturday of April if not a holiday, and the previous business day otherwise*. Unfortunately, the syntax allows the user to define collections which contain overlapping intervals.¹¹ This implies that there are collection expressions in the extended formalism for which there does not exist an equivalent granularity.

The slice and collection formalisms we discussed above are all interesting approaches to provide an intuitive yet exact representation of commonly used granularities. Manipulation of known granularities into new ones is the central idea to both approaches. These formalisms use a small number of operators to derive a great number of practical granularities. Instead of concerning ourselves with the number of operators, the calendar algebra uses the operations that directly mimic the way people are defining the granularities. We believe this gives us the most flexibility and hence can be used to derive more granularities. It also gives rise to more opportunities for efficient manipulation of granularities in applications.

2.7 Bibliographic Notes

A first formalization of a time granularity system is probably that of [CR87]. However, the system proposed in [CR87], as well as most of the time granularity systems proposed in the literature (e.g., [Dea89, MMC+92]), is quite

¹¹ For example, consider an expression representing *a semester following the last day of the month, if it is a Sunday, otherwise the week including that day*. Considering 1998/5/31 and 1998/6/30 as two days each one being the last day of a month, the expression will denote the semester starting 1998/6/1 for the first, and the week including 1998/7/1 for the second. Hence, both intervals will be in the resulting collection, with the first properly containing the second.

restrictive. For example, these systems often impose a total order on granularities. The restrictions are usually motivated by useful formal and computational properties that can be achieved with them. In [Dea89] a time granularity system is introduced and the author shows how it can be exploited to expedite operations on large temporal databases. In particular, the emphasis is on temporal reasoning tasks on large repositories of event descriptions temporally related to each other. The granularity system, called *hierarchical partitioning scheme*, is quite restrictive and can be easily characterized in terms of our general model: absolute time is the real numbers, restrictions (1), (2), and (3) on the structure of granules apply to this system in addition to both restrictions (a) and (b) on the relationships among granularities (see p. 20). The result is a set of granularities totally ordered by the *finer-than* relationship, and each one covering the whole time line. The specification of timestamps and distances among events allows the use of different granularities; however, before applying temporal reasoning algorithms, each value is translated in terms of a bottom granularity.

The proposal for the temporal extension of SQL (TSQL2) included a substantial part dealing with time granularity [TSQL2]. Even if there is no formal description of the intended granularity system, it is clear that it is a very general system allowing, for example, noncontiguous granules and incomparable granularities. In terms of our formal model, it could be characterized by using a finite subset of integers for absolute time and for indexing granules, with only restriction (1) on the structure of granules.

It is worth mentioning that the notion of granularity and calendar adopted here are somewhat different from the notion with the same name found in some literature. Indeed, some authors used "granularity" to denote a type of granule (e.g., one day) and "calendars" as sets of contiguous granules (e.g., the set of all days). In our terminology, the first is simply "a granule" of some granularity and the second is simply a granularity with contiguous granules. A calendar is a set of granularities (possibly sharing some properties) with a common bottom granularity. This "broader" notion of calendar seems now to be commonly accepted by most researchers.

Relevant work on time granularities has been done also in other areas like logic programming [MMC+92], and real time system specification [CCM+93]. In these papers the emphasis is on embedding these notions into a logical formalism. The granularity system proposed in [MMC+92] can be described in our framework exactly as the one discussed above for [Dea89] with the extra restriction (4) of equal size granules in each granularity. In [CCM+93] a slightly generalized version of that system is used, eliminating restriction (4) on granules and restriction (b) on granularity relationships, while additional restrictions on granularity relationships are specified through logic axioms.

Other formalism for symbolic representations can be found in the literature; most of these are extensions to the *slice* and *collection* formalisms that we have analyzed. However, none of these formalisms and extensions seems

to have the expressive power we are seeking; despite the fact that some of the proposals include features that go beyond what is needed in our framework. For example, [CD98] provides a powerful formalism to represent calendars and time repetition, including existential and universal quantification, which supports the representation of uncertainty, a feature not considered in our framework. Moreover, some *calendar expressions* in [CD98] go beyond the specification of granularities, as defined in [BWJ98a, BDE+98] and in this paper, allowing the representation of overlapping granules of time. The formalism can represent recurring events in the form of nonconvex intervals, but it does not seem to be able to represent *gap-granularities*, where gaps may not only occur between one granule and the next, but also within granules, e.g., a business-month.

Part II

Applications to Databases