A General Framework and Reasoning Models for Time Granularity*

Claudio Bettini

Dept. of Information Science (DSI)
University of Milan
Milan, Italy 20135

X. Sean Wang Sushil Jajodia

Dept. of Info.& Software Systems Eng.
George Mason University
Fairfax,VA 22030

Abstract

This paper presents a general framework to define time granularity systems. We identify the main choices differentiating the systems and investigate the formal relationships among granularities in these systems. The paper also introduces the notion of a network of temporal constraints with granularities emphasizing the semantical and computational differences from constraint networks with a single granularity. Consistency is shown to be NP-hard and an approximate algorithm proposed.

1 Introduction

Human activities heavily relate to calendar units and clock units (e.g., weeks, months, hours and seconds). System support and reasoning involving these units, also called granularities, has been recognized to be an important issue. However, many different definitions of granularities exist in the literature, and moreover, these definitions are often quite restrictive. The purpose of this paper is to introduce and study a general, unifying model for time granularities, and to propose and investigate temporal constraints with granularities.

Our approach is to define time granularities (called temporal types) in a very general way. On top of the general definition, we identify four dimensions of choices. A set of particular choices along the dimensions defines a *temporal type system*. The temporal granularity definitions we are aware of in the literature are all particular type systems in this framework. We also define and study relationships among temporal types. These relationships are often used as part of the definition of the type systems in the literature. This make the investigation of these relationships important. Data conversions among different granularities are also studied.

Regarding reasoning models, we introduce the concept of constraint network with granularities. It is interesting to note that the research in the literature mainly studied constraint networks with a single granularity, assuming that constraints with multiple granularities can be equivalently translated into ones with a single granularity (see related work section). We argue that the constraint network should allow multiple granularities and a constraint in one granularity may not be equivalent to one in another granularity. As an example, if a constraint says that an event must happen in the next *day* after another event happens, then this constraint cannot be translated into one in terms of *hours*. Indeed, it is incorrect to say that the second event must happen in 24 hours after the first event happens. A solution could be considering not single constraints but the whole network, by adding new nodes in such a way that the original network can be reduced into an equivalent one where all constraints are expressed in a basic granularity. In the above example, we could add two nodes representing the events of beginning and end of day (the boundaries of the granularity) and then add the constraints specifying the distance in hours of the nodes representing the original events from the boundaries (the new nodes). In general, this solution has two problems: (i) It is not clear what type of constraint should be used to identify granularity boundaries and how to deal with these new constraints during the propagation process, (ii) this approach cannot give a precise translation when granularities with time ticks of different length and/or with gaps among ticks (e.g., business days) are considered.

We study the consistency problem of multiple granularity constraint networks. We show that the consistency checking of such networks is NP-hard, even though we do not allow explicit disjunctive conditions. Any sound and complete propagation algorithm is thus unlikely efficient. We therefore propose an ap-

*Work of Bettini and Jajodia was supported in part by an ARPA grant, administered by the Office of Naval Research under grant number N0014-92-J-4038. Work of Wang was supported in part by the NSF grants IRI-9409769 and BIR-9404831.

0-8186-7528/96 $5.00

proximate algorithm based on constraint propagation.

The rest of the paper is organized as follows. In Section 2, definitions of temporal types and temporal type systems are given, and some properties of types and type systems are discussed. In Section 3, constraint networks with granularities are introduced and constraint propagation algorithms are studied. Section 4 discusses related work, and Section 5 concludes the paper.

2 The granularity model

We start with the formal concept of a temporal type that captures intuitive time granularities.

Definition (Temporal type)
Let $\mathcal{I}$ (index) be a discrete set wrt to a linear order $<_\mathcal{I}$ and $\mathcal{A}$ (absolute time) a set with a linear order. Then *a temporal type on* $(\mathcal{I}, \mathcal{A})$ is a mapping μ from $\mathcal{I}$ to $2^\mathcal{A}$ such that

- $\mu(i) \neq \emptyset$ and $\mu(j) \neq \emptyset$, where $i <_\mathcal{I} j$, imply that each element in $\mu(i)$ is less than (wrt the linear order in $\mathcal{A}$) all the elements in $\mu(j)$,
- for all $i <_\mathcal{I} j$, if $\mu(i) \neq \emptyset$ and $\mu(j) \neq \emptyset$, then $\forall k$ $i <_\mathcal{I} k <_\mathcal{I} j$ implies $\mu(k) \neq \emptyset$.

Property (1) states that the mapping must be *monotonic*. Property (2) disallows an empty set to be the value of a mapping for a certain index value if a lower index *and* a higher index are mapped to non-empty values. The set $\mu(i)$ is said to be the *i-th tick of* μ, or *tick* i *of* μ, or simply *a tick of* μ.

Intuitive temporal types, e.g., `day`, `month`, `week` and `year`, satisfy the above definition. For example, we can take the set of positive integers as index set and define a special temporal type `year` starting from year 1800 as follows: `year`(1) is the set of absolute time corresponding to the year 1800, `year`(2) is the set of absolute time corresponding to the year 1801, and so on. Note that the sets in the type `year` are consecutive intervals; however, this does not have to be the case for all types. Leap years, which are not consecutive intervals, also constitute a temporal type. If we take 1892 as the first leap year, then `leap-year`(2) corresponds to 1896, `leap-year`(3) corresponds to 1904,[1] `leap-year`(4) corresponds to 1908, and so on. We can also represent a finite collection of "ticks" as a temporal type as well. For example, to specify the year 1993, we can use the temporal type T such that $T(1)$ is the set of absolute time corresponding to the year 1993, and $T(i) = \emptyset$ for each $i > 1$.

[1] Note 1900 is not a leap year.

Note that this definition allows temporal types in which ticks are mapped to more than one interval. For example, it is possible to have a temporal type `b-month`, where a business month is a union of all business days (`b-day`) in a month (i.e., excluding all Saturdays, Sundays, and general holidays). Figure 1 illustrates some of the aforementioned temporal types considering the span of time from February 26th till April 2nd 1996.

It is also possible that several temporal types are different (have different mappings) but include the same sets of absolute time. For example, let the index set be $\mathbf{Z}$ (the integers), and let μ and ν be two types denoting years. Suppose μ denotes years of the Gregorian calendar, starting from $\mu(1)$ mapped to the first year, up to $\mu(2000)$ mapped to year 2000, and every other index mapped to the empty set. Consider ν such that $\nu(-999)$ is mapped to the first year of the Gregorian calendar, $\nu(0)$ is year 1000, up to $\nu(1000)$ mapped to year 2000, and every other index mapped to the empty set. These two types cover the same period of absolute time dividing it into the same ticks; however they use different indexes to refer to a certain tick. While it can be sometime useful to distinguish among these types, it is easy to see that they can be considered equivalent under the intuitive operation of *shifting* their indexes.

The proposed definition is a generalization of most previous definitions of time granularities. When considering a particular application or formal context, we can specialize this very general model along the following dimensions:

- choice of the index set $\mathcal{I}$
- choice of the absolute time set $\mathcal{A}$
- restrictions on the structure of ticks (explained below)
- restrictions on the temporal types by using relationships

We call the resulting formalization a *temporal type system.*

Consider some possibilities for each of the above four dimensions. Convenient choices for the index set are natural numbers, integers, and any finite subset of them. The choice for absolute time is typically between dense and discrete. In general, if the application imposes a fixed basic granularity (such as `second`), then a discrete absolute time in terms of the basic granularity is probably the appropriate choice. However, if one is interested in being able to represent arbitrary finer temporal types, a dense absolute

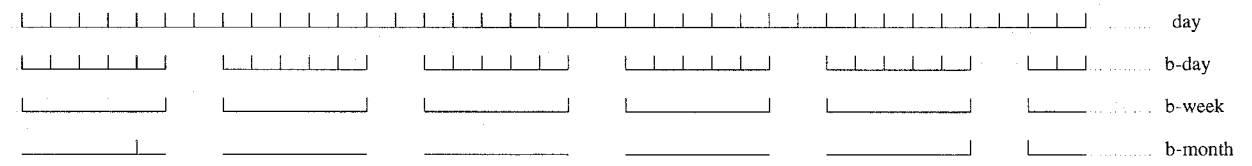

Figure 1: Three temporal types with day as the absolute time.

time is required. In both cases, specific applications could impose left/right boundedness on the absolute time set. The structure of ticks could be restricted in several ways: (1) disallow types with *gaps* in a tick (`b-month` is an example, since each tick has gaps corresponding to weekends), (2) disallow types with *non-contiguous* ticks (`b-day` is an example, since the tick corresponding to Friday is not contiguous wrt the next tick, representing Monday), (3) disallow types whose ticks do *not cover all* the absolute time (if our absolute time is between Gregorian year 0 and 2000, we disallow, for example, a type covering only years from 1000 to 2000), or (4) disallow types with *non-uniform* ticks (only types with ticks having the same *size* are allowed. Hence, for example, `second`, `day`, `week` are allowed, while `month` is not). Temporal types can also be restricted based on their relationships. While we formally define these relationships in the next subsection, intuitive examples are: (1) disallow multiple types that are equivalent wrt shifting of their indices, (2) enforce that each pair of types 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).

There is a tradeoff in restricting a temporal type system between its expressiveness and the simplicity and efficiency of the algorithms needed to manage the temporal types. Here, we give the definition of a very expressive temporal type system [WBBJ] that we have found to be particularly interesting.

Definition ($\mathcal{TTS}_1$)
A *temporal type* in $\mathcal{TTS}_1$ is a mapping μ from the set of the positive integers to $2^{\mathcal{R}}$ such that for all positive integers i and j with $i < j$, the following two conditions are satisfied:

1. $\mu(i) \neq \emptyset$ and $\mu(j) \neq \emptyset$ imply that each real number in $\mu(i)$ is less than all real numbers in $\mu(j)$, and

2. $\mu(i) = \emptyset$ implies $\mu(j) = \emptyset$.

The $\mathcal{TTS}_1$ system is defined with the following choices of the above four dimensions: The positive integers are used as the index set, while the real numbers are used as the absolute time set. There is no structural restriction on ticks (e.g., gaps are allowed), but no two types in the system are equivalent wrt shifting of their indices. This last restriction is enforced by the condition that the first non-empty tick (if any) must start with index 1 (condition 2 in the definition of $\mathcal{TTS}_1$).

An example of a more restrictive granularity system is the one we defined in [WJS95, BWBJ95] where a discrete absolute time and index set (positive integers) are used. Furthermore, no gaps are allowed in a tick (i.e., each tick is an interval on positive integers), and two ticks with consecutive index values must be contiguous (i.e., $\mu(i) \cup \mu(i+1)$ is an interval or the empty set for all i). Furthermore, each type must cover the whole absolute time line. Shifting equivalent types are not allowed in the system by enforcing that the first non-empty tick (always exists) has index 1. In this paper we refer to this temporal type system as $\mathcal{TTS}_2$. It can be easily seen that each type in $\mathcal{TTS}_2$ corresponds to a partition of the positive integers such that each subset in the partition is a finite or infinite interval. The index can be viewed as a label to the intervals with 1 assigned to the first interval.

2.1 Relationships and formal properties

We define a number of interesting relationships among temporal types.

Definition Let μ and ν be temporal types on $(\mathcal{I}, \mathcal{A})$.

- *Finer-than.* μ is said to be *finer than* ν, denoted $\mu \preceq \nu$, if for each $i \in \mathcal{I}$, there exists $j \in \mathcal{I}$ such that $\mu(i) \subseteq \nu(j)$.

- *Groups-into.* μ groups into ν, denoted $\mu \trianglelefteq \nu$, if for each $j \in \mathcal{I}$, there exist $i \in \mathcal{I}$ and a positive integer k such that $\nu(j) = \bigcup_{r=0}^{k} \mu(i+r)$.

- *Subtype.* μ is a subtype of ν, denoted $\mu \sqsubseteq \nu$, if for each $i \in \mathcal{I}$ exists $j \in \mathcal{I}$ such that $\mu(i) = \nu(j)$.

- *Shifting.* μ and ν are shifting equivalent, denoted $\mu \overset{\leftrightarrow}{=} \nu$, if $\mu \sqsubseteq \nu$ *and* $\nu \sqsubseteq \mu$.

Consider now the intuitive meaning and properties of these relationships. The finer relationship formalizes the notion of finer "partitions" of the absolute time. For example, `hour` is finer than `day` which in turn is finer than `month`. By definition, this relation is reflexive, i.e., $\mu \preceq \mu$ for each temporal type μ. Furthermore, the finer relation is obviously transitive. However, if no restrictions are given, it is not antisymmetric, and hence it is not a partial order. Indeed, $\mu \preceq \nu$ and $\nu \preceq \mu$ does not imply $\mu = \nu$, but only $\mu \overset{\leftrightarrow}{=} \nu$. The *groups-into* relation ensures that for each tick of ν there exists a set of ticks of μ covering exactly the same span of time. For example, `hour` groups into `b-day`. The relation is useful, for example, in applications where attribute values are associated with time ticks; sometimes it is possible to obtain the value associated with a tick of granularity ν from the values associated with the ticks of μ whose union covers the same time. Note that $\mu \preceq \nu$ does not imply $\mu \trianglelefteq \nu$ or vice versa. The groups-into relation has the same two properties of the finer-than relation. The *subtype* relation intuitively identifies a type corresponding to subsets of ticks of another type. As an example, `b-day` is a subtype of `day`. Note that $\mu \sqsubseteq \nu$ implies $\mu \preceq \nu$. As the two previous relations, subtype is reflexive and transitive. Finally, the shifting relation has been intuitively described above with the example about two different types representing years which are *shifting equivalent.* Shifting is an equivalence relation.

We can prove some more interesting formal properties for the two specific temporal type systems $\mathcal{TTS}_1$ and $\mathcal{TTS}_2$ illustrated above. In particular, since no pair of different types in the system are shifting equivalent, the three relationships $\preceq$, $\sqsubseteq$, and $\trianglelefteq$ are antisymmetric, and hence, each relationship is a partial order. They are not total orders since, for example, `week` and `month` are incomparable with respect to $\preceq$, $\trianglelefteq$, and $\sqsubseteq$, i.e., `week` is not finer than, does not group into, nor it is a subtype of `month`, and vice versa.

We are particularly interested in the finer-than relationship. Consider $\mathcal{TTS}_1$. There exists a unique least upper bound of the set of all temporal types denoted by $\mu_\top$, and a unique greatest lower bound, denoted by $\mu_\bot$. These top and bottom elements are defined as follows: $\mu_\top(1) = \mathcal{R}$ and $\mu_\top(i) = \emptyset$ for each $i > 1$, and $\mu_\bot(i) = \emptyset$ for each positive integer i. Moreover, for each pair of temporal types μ_1, μ_2, there exist a unique least upper bound $lub(\mu_1, \mu_2)$ and a unique greatest lower bound $glb(\mu_1, \mu_2)$ of the two types, with respect to $\preceq$. We formally state the following result:

Theorem 1 The set of all temporal types definable in $\mathcal{TTS}_1$ is a lattice with respect to the finer-than relationship. The same holds for the types definable in $\mathcal{TTS}_2$.

The usefulness of the lattice structure has been shown, for example, in the logical design of temporal databases with multiple granularities [WBBJ].

2.2 Data conversion

We consider two kinds of data conversions. One is time tick conversions and the other timestamped information conversion.

When dealing with temporal types, we often need to determine the tick (if any) of a temporal type ν that covers a given tick z of another temporal type μ. 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 temporal types μ and ν, $\lceil z \rceil^\nu_\mu$ is undefined if $\mu(z) \not\subseteq \nu(z')$ for all z'; otherwise, $\lceil z \rceil^\nu_\mu = z'$, where z' is the unique index value such that $\mu(z) \subseteq \nu(z')$ The uniqueness of z' is guaranteed by the monotonicity of temporal types. As an example, $\lceil z \rceil^{\texttt{month}}_{\texttt{second}}$ gives the month that includes the second z. Note that while $\lceil z \rceil^{\texttt{month}}_{\texttt{second}}$ is always defined, $\lceil z \rceil^{\texttt{month}}_{\texttt{week}}$ is undefined if week z falls between two months. Similarly, $\lceil z \rceil^{\texttt{b-day}}_{\texttt{day}}$ is undefined if day z is not a business day. Note that if $\mu \preceq \nu$, then the function $\lceil z \rceil^\nu_\mu$ is defined for each index value z. For example, since `day` $\preceq$ `week`, $\lceil z \rceil^{\texttt{week}}_{\texttt{day}}$ is always defined, i.e., for each day we can find the week that contains it.

Another direction of the above transformation is as follows: Let μ and ν be temporal types, and z an integer. Define $\lfloor z \rfloor^\nu_\mu = (z', k)$, where the resulting pair identifies the k consecutive ticks of μ whose union makes $\nu(z)$, i.e., $\nu(z) = \mu(z'+0) \cup \cdots \cup \mu(z'+k-1)$. This function is useful for finding, e.g., all the days in a month. If $\mu \trianglelefteq \nu$, then $\lfloor \rfloor^\nu_\mu$ is always defined. It is worth mentioning that in [TSQL2], it is assumed a similar function is provided by a *calendar.* For each tick of a source type the function gives the conversion factor for a certain target type. For example, given the month Feb, 1996, the function returns "29 days".

The above transformation only concerns the temporal types. The second kind of data conversion involves information attached to time ticks. For example, we register in a database the rainfall amounts 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`. The conversion function used is the summation. In [BWBJ95],

we proposed a framework for specifying such information conversion and studied related query evaluation problems.

3 Temporal constraints with granularities

In the temporal reasoning area a lot of work has been done on formalisms to express networks of constraints and on algorithms to solve the related problems of consistency and minimal network. In this section, we propose an extension of quantitative temporal constraints to be expressed in terms of temporal types. We start with the definition of a temporal constraint with granularity. For simplicity, we assume to work with temporal types in $\mathcal{TTS}_1$.

Definition Let $m \leq n$ be non-negative integers and μ a temporal type. Then $[m,n]\,\mu$, called a *temporal constraint with granularity* or *TCG*, is the binary relation on positive integers defined as follows: For positive integers t_1 and t_2, $(t_1,t_2) \in [m,n]\,\mu$ is true (or t_1 *and* t_2 *satisfy* $[m,n]\,\mu$) iff (1) $t_1 \leq t_2$, (2) $\lceil t_1 \rceil^\mu$ and $\lceil t_2 \rceil^\mu$ are both defined, and (3) $m \leq (\lceil t_2 \rceil^\mu - \lceil t_1 \rceil^\mu) \leq n$.

Intuitively, for instants $t_1 \leq t_2$ (in terms of seconds), t_1 and t_2 satisfy $[m,n]\,\mu$ if the difference of the integers t_1' and t_2' is between m and n (inclusive), where $\mu(t_1')$ and $\mu(t_2')$ are the ticks of μ (if exist) that cover, respectively, the t_1-th and t_2-th seconds. For example, the pair (t_1,t_2) satisfies TCG $[0,0]\,\texttt{day}$ if t_1 and t_2 are within the same day but $t_1 \leq t_2$. Similarly, (t_1,t_2) satisfies TCG $[0,2]\,\texttt{hours}$ if t_2 is either in the same second as t_1 or within two hours after t_1. Finally, (t_1,t_2) satisfies $[1,1]\,\texttt{month}$ if t_2 is in the next month with respect to t_1.

It is important to note that it is not always possible to convert a TCG $[m,n]\,\mu$, with $\mu \neq \texttt{second}$, into a TCG in seconds. Indeed, consider $[0,0]\,\texttt{day}$. Two instants satisfy the constraint if they fall within the same day. In terms of second, they could differ from 0 seconds to $24 * 60 * 60 - 1 = 86399$ seconds. However, $[0,86399]\,\texttt{second}$ does not reflect the original constraint. For example, if instant t_1 corresponds to 11pm of one day and instant t_2 to 4am in the next day, then t_1 and t_2 do not satisfy $[0,0]\,\texttt{day}$; however, they do satisfy $[0,86399]\,\texttt{second}$.

Definition A *constraint network (with granularities)* is a directed acyclic graph (W,A,Γ), where W is a finite set of variables, $A \subseteq W \times W$ and Γ is a mapping from A to the finite sets of TCGs.

Intuitively, a constraint network specifies a complex temporal relationship where each variable in W represents a specific instant (for example the occurrence time of an event). The set of TCGs assigned to an edge is taken as conjunction. That is, for each TCG in the set assigned to the edge (X_i,X_j), the instants assigned to X_i and X_j must satisfy the TCG. Figure 2 shows two constraint networks.

3.1 Complexity of consistency checking

For a given constraint network, it is of practical interest to check if the network is consistent. However, we have:

Theorem 2 It is NP-hard to decide if an arbitrary constraint network with granularities is consistent.

The proof consists of a reduction from the "subset sum" problem [GJ79]. The result holds for our general definition of temporal type as well as for $\mathcal{TTS}_1$ and $\mathcal{TTS}_2$ (the proof only requires that incomparable granularities are allowed). The basic difficulty of the consistency checking is due to the fact that the presence of different granularities in the constraints allows us to express a form of disjunction. Consider the graph in Figure 2(b). The relationship between X_1 and X_0 dictates that the event assigned to X_0 must happen during the first month of a year (each year has 12 months). Likewise, the event assigned to X_2 must happen during the first month of a year (maybe in a different year from the event assigned to X_0). Since the original relationship between X_0 and X_2 is that their distance is between 0 to 12 months, it follows that the distance between X_0 and X_2 must be either 0 *or* 12 months.

3.2 An approximate algorithm

We consider here an approximate algorithm for consistency checking. The approach is constraint propagation.

Let $\mathcal{CN} = (W,A,\Gamma)$ be a constraint network and M the set of temporal types appearing in Γ. The algorithm proceeds as follows. It first partitions TCGs in a constraint network into groups, each group having TCGs in the same temporal type. That is, for each μ in M, let $\mathcal{C}_\mu$ be the set of all the constraints $X - Y \in [m,n]$, where X, Y are in W and $[m,n]\,\mu \in \Gamma(X,Y)$. Now, the propagation within $\mathcal{C}_\mu$ is a problem known as the Simple Temporal Problem [DMP91]. We apply a path consistency algorithm within each group. Since constraints expressed in a granularity could imply constraints in other granularities, we try to convert them and add the derived constraints to the corresponding groups. Hence, for each pair of temporal types μ and ν in M such that a conversion is allowed[2], we convert

[2]The conversion must guarantee that a logically implied constraint exists in the target granularity. We use the sufficient condition that the set of time instants corresponding to the source

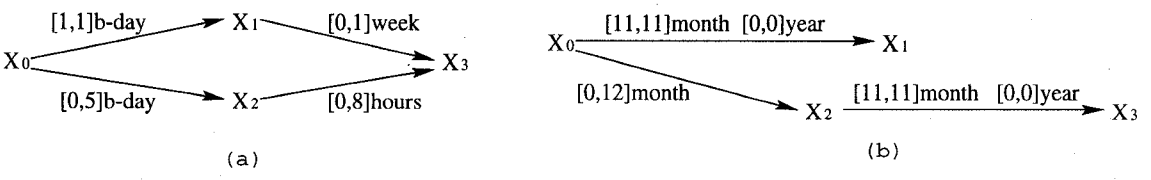

Figure 2: Two constraint networks.

each constraint in C_μ into one in terms of ν and add it into C_ν. This conversion is done using the procedure reported in Appendix. The process is repeated with the path consistency algorithm and the conversion, until no new constraints appear in any group. If any of the resulting constraints are empty (not satisfiable), the network is inconsistent.

We say that the algorithm is *sound* if any assignment satisfying the given constraint network $\mathcal{CN} = (W, A, \Gamma)$, also satisfies $\mathcal{CN}' = (W, A', \Gamma')$, where A' and Γ' are given by the algorithm (e.g., if $X - Y \in [m,n]$ is in C_μ, then $(Y,X) \in A'$ and $[m,n]\mu \in \Gamma'(Y,X)$).

Theorem 3 The approximate propagation algorithm is sound, terminates, and requires time $O(c*P)$ where c is the time taken by the procedure to convert a constraint and P is a polynomial in the number of nodes in the network, the number of temporal types, and the maximum conversion factor among types.

Note that, in practice, the time c can be considered a constant factor (see Appendix).

The aforementioned algorithm is an *approximate* propagation for two main reasons. First, some types cannot be converted in other types (e.g. `b-day` into `week-end`) and, in general, conversions involve some loss of precision. Second, the set of temporal types we use are only those that appear in the constraint network. The algorithm may derive tighter constraints (in the sense of logical implication) if additional temporal types are used.

4 Related work

Most of the time granularity systems proposed in the literature are quite restrictive, often imposing a total order on granularities [CR87, Dea89, MMCR92]. The restrictions are usually motivated by useful formal and computational properties that can be achieved with them.

granularity must be a subset of that corresponding to the target granularity.

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 and index set are respectively reals and integers, restrictions 1, 2, and 3 on the structure of ticks apply to this system in addition to both restrictions 1 and 2 on the relationships among types. The result is a set of temporal types totally ordered by the *finer* relation, 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 any value is translated in terms of a basic granularity. In this paper we propose a different semantics for constraints with granularities, and, consequently, different algorithms.

The recent proposal for the temporal extension of SQL (TSQL2) includes 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, non-contiguous ticks and incomparable types. In terms of our formal model it could be characterized by using a finite subset of integers for both absolute time and index set with only restriction (1) on the structure of ticks.

Relevant work on this subject has been done also in other areas like logic programming [MMCR92], and real time system specification [CCMP93]. In these papers the emphasis is on embedding these notions into a logical formalism. The granularity system proposed in [MMCR92] 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 [CCMP93] a slightly generalized version of that system is used, eliminating restriction (4) on ticks and restriction (2) on type relationships, while more prop-

erties on these relationships are specified through logic axioms.

Considering constraint propagation, a related work is that of [Euz95]. While we address the problem of reasoning with quantitative temporal constraints on multiple granularities, that paper considers both temporal and spatial aspects of granularity, but limited to qualitative constraints.

Finally, several papers address the problem of the representation and implementation of calendars and granularities [LMF86, NS92, TSQL2, CSS94]. The set of granularities expressible in these proposals are often characterized only by a representation language. This set can be formally described as one of our temporal type systems. We see this work as complementary to ours. Indeed, a real system can only treat a subset of the temporal types that we have defined, namely those that have finite representations. These languages could be used in the application domain to specify the temporal types and implement operations on them.

5 Conclusion

We illustrated a general framework for defining time granularities that extends most of the proposed granularity systems. Furthermore, we introduced the notion of a constraint network with time granularities and showed that such a constraint network has semantical and computational differences wrt similar constraint networks without time granularities.

We have applied the proposed framework for time granularities in several areas: federated temporal databases [WJS95], logical design of temporal databases with multiple granularities [WBBJ], querying temporal databases with semantic assumptions [BWBJ95], and mining large event sequences for complex temporal relationships [BWJ96]. There is certainly another broad area that deserves an investigation for the application of the presented granularity model and constraint propagation techniques. This area includes the many A.I. applications involving temporal reasoning, from scheduling and planning to diagnosis and natural language processing.

With respect to temporal logics, an interesting direction is the study of metric temporal logics along the work of [CCMP93], but extending these formalisms to deal with such general models of granularity.

References

[BWBJ95] C. Bettini, X. Wang, E. Bertino, and S. Jajodia. Semantic assumptions and query evaluation in temporal databases. In *Proc. of ACM SIGMOD-95*, pages 257–268, San Jose,CA, 1995.

[BWJ96] C. Bettini, X. Wang, and S. Jajodia. Testing complex temporal relationships involving multiple granularities and its application to data mining. In *Proc. of ACM PODS-96*, Montreal, Canada, 1996.

[CCMP93] E. Ciapessoni, E. Corsetti, A. Montanari, and P. San Pietro. Embedding time granularity in a logical specification language for synchronous real_time systems. *Science of Computer Programming*, 20(1), 1993.

[CR87] J. Clifford and A. Rao. A simple, general structure for temporal domains. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 23–30, France, May 1987.

[CSS94] R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars and temporal rules in next generation databases. In *Proc. ICDE*, 1994.

[Dea89] T. Dean. Artificial intelligence: Using temporal hierarchies to efficiently maintain large temporal databases. *JACM*, 36(4):687, 1989.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49, 1991.

[TSQL2] The TSQL2 Temporal Query Language, R. T. Snodgrass Editor, Kluwer Academic Publishers, 1995.

[Euz95] J. Euzenat. An algebraic approach for granularity in qualitative space and time representation. In *Proc. IJCAI-95*, pages 894–900, San Mateo, CA, 1995.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability – A guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[LMF86] B. Leban, D. Mcdonald, and D. Foster. A representation for collections of temporal intervals. In *Proc. AAAI-86*, pages 367–371, 1986.

[MMCR92] A. Montanari, E. Maim, E. Ciapessoni, and E. Ratto. Dealing with time granularity in the event calculus. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems 1992*, volume 2, pages 702–712, Tokyo, Japan, June 1992. ICOT.

[NS92] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proc. CIKM-92*, Baltimore, 1992.

[WBBJ] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical design for temporal databases with multiple granularities. *ACM TODS*, to appear.

[WJS95] X. Wang, S. Jajodia, and V.S. Subrahmanian. Temporal modules: An approach toward federated temporal databases. *Information Sciences*, 82:103–128, 1995.

Appendix: Conversion of constraints in different granularities

Given a constraint $Y - X \in [m, n]$ in terms of a granularity μ_1 (i.e., contained in C_{μ_1}), we want to derive an implied[3] constraint in terms of a granularity μ_2 (to be added in C_{μ_2}). If we only have a total order of granularities, like e.g. `minute`,`hour`, and `day`, then the conversion algorithm is trivial since fixed conversion factors can be used. However, if incomparable types like `week` and `month`, or types with "gaps" like `b-day` are considered, the conversion becomes more complex. In Figure 3 we propose an algorithm that is sufficiently general to apply to any pair of temporal types, provided that the target type covers a span of time equal or larger than the span of time covered by the source type. For example, we can convert a constraint in terms of `b-week` into a constraint in terms of `week`, `month`, or `b-day`, but not into a constraint in terms of `week-ends`. The above condition on the target type is quite restrictive and could be relaxed, but it ensures that the proposed algorithm performs only conversions deriving logically implied constraints.

INPUT: a constraint $Y - X \in [m, n]$, a source type μ_1, a target type μ_2 s.t. $\forall i, t\ (t \in \mu_1(i) \rightarrow \exists j\ t \in \mu_2(j))$.

OUTPUT: a constraint $Y - X \in [\overline{m}, \overline{n}]$ in terms of μ_2 implied by $Y - X \in [m, n]$ in terms of μ_1.

METHOD:

1. $\overline{n} = min(S) - 1$, where
   $S = \{s \mid minsize(\mu_2, s) > maxsize(\mu_1, n+1) - 1\}$
2. $\overline{m} = min(R) - 1$, where
   $R = \{r \mid maxsize(\mu_2, r) > mingap(\mu_1, m)\}$

Figure 3: Algorithm for the conversion of constraints

[3] A constraint is implied by another constraint if each pair of values satisfying the second satisfies also the first one.

The algorithm assumes the existence of a primitive type μ (e.g. `second`) in the considered granularity system such that any tick of the other types can be obtained as union of ticks of the primitive type. Formally, μ *groups-into* ν ($\mu \trianglelefteq \nu$) for each type ν. The algorithm uses an approximation of the relation between each type and the primitive type, and hence, in some cases, it does not give the tightest possible bounds as an output.

To specify the algorithm we first have to introduce some notation. Let $minsize(\nu, k)$ and $maxsize(\nu, k)$ be respectively the minimum and maximum length of k consequent ticks of ν, expressed in ticks of the primitive type. For example, $minsize(\texttt{month}, 1) = 28$, $maxsize(\texttt{month}, 1) = 31$, and $maxsize(\texttt{b-day}, 2) = 4$ if `day` is used as the primitive type. We also use $mingap(\nu, k)$ to denote the minimal distance in terms of the primitive type between each tick of ν and the kth one after it. Formally, $mingap(\nu, k) = min_i\{s_i \mid \exists j\ \mu(j) \cap \nu(i) \neq \emptyset$ and $\mu(j + s_i) \cap \nu(i + k) \neq \emptyset\}$. For example, $mingap(\texttt{b-week}, 1) = 3$, i.e., the minimum distance in terms of days between two events that occur in two different business weeks is 3 (one on Friday and the other on Monday).

Steps (1) and (2) compute respectively the new maximum and minimum in terms of the target type. The maximum and minimum distances in terms of the primitive type are respectively $maxsize(\mu_1, n + 1) - 1$ and $mingap(\mu_1, m)$. We have to find these distances in terms of the target type. For the maximum we use the minimal length ($minsize()$) of a group of ticks in the target type, since we want to maximize the number of ticks needed to cover the given distance. Analogously, we use the maximal length ($maxsize()$) in the computation of the minimum value.

We assume that the algorithm can access a table containing the values of $minsize(\nu, k)$, $maxsize(\nu, k)$, $mingap(\nu, k)$ for each considered type ν and positive integer k limited by some constant. For some granularities, these values can be easily automatically derived. In general, when the values are not available for a certain k they can be approximated with a linear combination of the known values (this is not shown in the algorithm).