A General Framework for Time Granularity and Its Application to Temporal Reasoning*

Technical Report: ISSE-TR-96-10

Claudio Bettini† X. Sean Wang‡ Sushil Jajodia‡

Abstract

This paper presents a general framework to define time granularity systems. We identify the main dimensions along which different systems can be characterized, and investigate the formal relationships among granularities in these systems. The paper also introduces the notion of a network of temporal constraints with (multiple) granularities emphasizing the semantic and computational differences from constraint networks with a single granularity. Consistency of networks with multiple granularities is shown to be NP-hard in general and approximate solutions for this problem and for the *minimal network* problem are 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 [Hob85, CR87, TSQL2]. However, many different definitions of granularities exist in the literature and, moreover, these definitions are often quite restrictive. The

*A preliminary version of this paper appeared in the Proceedings of 3rd International Workshop on Temporal Representation and Reasoning, Key West, FL, 1996. The work of Bettini was partially carried out while visiting George Mason University. The work of Wang was partially supported by the National Science Foundation under the grant IRI–9633541. The work of Jajodia was partially supported by the National Science Foundation under the grants IRI–9633541 and INT–9412507.

†Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39, 20135 Milano, Italy. bettini@dsi.unimi.it

‡Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030-4444, USA. {xywang, jajodia}@isse.gmu.edu

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 time 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. An investigation of these relationships is important because they are often used as part of the definition of the type systems in the literature. Data conversions among different granularities are also considered.

A second contribution of this paper is the extension of the well-known problem of temporal constraint satisfaction to the case in which constraints are specified in terms of multiple granularities within a temporal type system. We introduce constraint networks with granularities specified as a set of variables representing instants with conditions of the form $[m, n]\mu$ associated with pairs of variables, where m and n are integers and μ is a temporal type. The meaning of $[m, n]\mu$ associated with (X, Y) is that the two time instants assigned to X and Y must be temporally apart by at least m ticks and at most n ticks in terms of μ. For example, $[0, 0]$`day` means that the two time instants must lie within the same day (i.e., at most and at least 0 days apart). We assume that a number of distinct temporal types appear in the network. We then consider the two main questions that are usually asked about constraint networks: (i) Does the given network have a solution? (ii) Which network has the tightest constraints among the networks with the same solutions as the given one? These problems are known as the *consistency* problem and the *minimal network* problem.

It is interesting to note that the research in the literature mainly studied constraint networks with a single granularity, with the underlying assumption that the constraints with multiple granularities can be equivalently translated into a set of constraints with a single granularity (see related work section). We argue that the constraint network should allow multiple granularities because a constraint in one granularity may not always be equivalent to one in another granularity. As an example, consider a constraint which says that an event must happen during the next `day` after a certain event happens. This constraint cannot be translated into one in terms of `hours`. Indeed, it is incorrect to say that the second event must happen within 24 hours after the first event happens.

At the first glance, a solution could be provided by considering not just single constraints, but the whole constraint network. More specifically, one might consider adding new nodes into a given network to facilitate a translation from the given constraint network with multiple granularities into

an equivalent one with all constraints expressed in a single basic granularity. In the above example, we could add two nodes representing the events of beginning and end of days (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 reasoning process, and (ii) when granularities with time ticks of different length (e.g., months) and/or with gaps among ticks (e.g., business days) are considered, this approach cannot give a translation into an equivalent network.

In view of these considerations, we directly study the constraint networks with multiple granularities. 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 to be efficient. We therefore propose an approximate algorithm based on constraint propagation. The basic idea of the algorithm is to repeat the following two steps: (1) apply the path consistency algorithm [DMP91] to each subnetwork having constraints in terms of the same granularity, and (2) translate the derived constraints into the other granularities appearing in the network. We give two methods to perform step (2). The first method assumes that the system has the information on the relationships between all pairs of temporal types, while the second assumes that the system knows only the relationship between each temporal type and a basic one. We show, in each case, that the propagation algorithm is correct, although the first method generally yields better solutions.

In various parts of the paper, we also briefly mention applications of the temporal type systems and constraint networks. These applications are essentially from our previous papers [WJS95, WBBJ97, BWBJ95, BWJ96].

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 multiple granularities are introduced, and the problem of their consistency is shown to be NP-hard. In Section 4 approximate constraint propagation algorithms for these networks are studied. Section 5 discusses related work, and Section 6 concludes the paper. The appendix contains the proofs of the theorems in the paper.

2 The granularity model

We start with the concept of a temporal type to formalize the notion of time granularities.

Definition Let $(\mathcal{I}, \leq_{\mathcal{I}})$ (index) be a discrete linear order isomorphic to a subset of the integers with the usual order relation, and let $(\mathcal{A}, \leq_{\mathcal{A}})$ (absolute time) be 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 (with respect to $<_{\mathcal{A}}$) 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 sets. 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 the 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 $i > 1$.

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` (business months), 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.

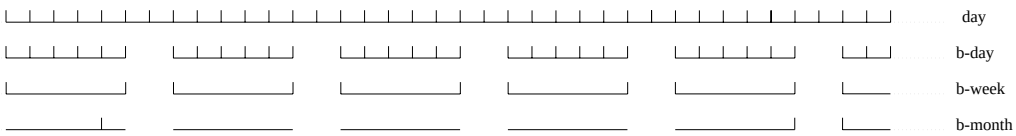

Figure 1: Three temporal types with `day` as the absolute time.

[1]Note 1900 is not a leap year.

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 by "shifting" the indices.

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 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 a tick corresponding to a Friday is not contiguous with respect to the next tick, representing a 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:

(a) disallow multiple types that are equivalent with respect to shifting of their indices,

(b) 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).

In restricting to a temporal type system, there is a tradeoff 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 [WBBJ97] that we have found to be particularly useful.

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

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

The $\mathcal{TTS}_1$ system is defined with the following choices for the aforementioned 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, but no two types in the system are equivalent with respect to shifting of their indices. Restriction (a) is indeed enforced by the condition that the first non-empty tick (if any) must start with index 1 (second condition in the definition of $\mathcal{TTS}_1$).

An example of a more restrictive granularity system is given in [WJS95, BWBJ95] where a discrete absolute time and index sets (both positive integers) are used, and restrictions (1), (2), (3), and (a) apply. 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*: If μ and ν are temporal types, then μ is said to *group into* ν, denoted $\mu \trianglelefteq \nu$, if for each non-empty tick $\nu(j)$, there exists a (possibly infinite) subset S of $\mathcal{I}$ such that $\nu(j) = \bigcup_{i \in S} \mu(i)$.
- *Subtype*: μ is said to be a *subtype* of ν, denoted $\mu \sqsubseteq \nu$, if for each $i \in \mathcal{I}$, there exists $j \in \mathcal{I}$ such that $\mu(i) = \nu(j)$.
- *Shifting*: μ and ν are said to be *shifting equivalent*, denoted $\mu \overset{\leftrightarrow}{=} \nu$, if $\mu \sqsubseteq \nu$ *and* $\nu \sqsubseteq \mu$.

When a temporal type μ_1 is finer than a temporal type μ_2, we also say that μ_2 is *coarser* than μ_1.

Consider now the intuitive meaning and properties of these relationships. The finer-than 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-than 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 $\mu \overset{\leftrightarrow}{=} \nu$. Considering the *groups-into* relation, $\mu \trianglelefteq \nu$ 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 $\mu \trianglelefteq \nu$; it's converse does not hold either. 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 $\mu \preceq \nu$. Similar to the two previous relations, subtype is reflexive and transitive. Finally, shifting is clearly an equivalence relation.

We can prove other interesting formal properties for a large class of temporal type systems, namely those imposing, by the above restriction (a), that no pair of different types can be shifting equivalent. For this class of systems, the three relationships $\preceq$, $\sqsubseteq$, and $\trianglelefteq$ 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 $\preceq$, $\trianglelefteq$, and $\sqsubseteq$ (i.e., `week` is not finer than, does not group into, nor is a subtype 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 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: $\mu_\top(i) = \mathcal{A}$ for some $i \in \mathcal{I}$ and $\mu_\top(j) = \emptyset$ for each $j \neq i$, and $\mu_\bot(i) = \emptyset$ for each i in $\mathcal{I}$. 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 formalize this result referring to specific temporal type systems obtained by the choices and restrictions illustrated earlier in this section (page 5).

Theorem 1 *Any temporal type system having an infinite index, and (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* types can be constructed for each pair of types 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 (μ, ν) of their types, $glb(\mu, \nu)$ and $lub(\mu, \nu)$ in the general lattice of Theorem 1 are within the system. A closed granularity system obviously form 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 types leads to the non-uniqueness of *glb* and *lub*. The infiniteness requirement on the index simply ensures that a sufficient number of values will be available to index ticks of $glb()$ and $lub()$.

Theorem 2 *A granularity system satisfying the conditions in Theorem 1 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 temporal types μ_1 and μ_2 satisfy restriction (2) or (4), then it is not always true that $lub(\mu_1, \mu_2)$ and $glb(\mu_1, \mu_2)$ given in the general lattice of Theorem 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), non-closed 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. Furthermore, a non-closed granularity system could still form a lattice, but a particular construction for its *glb*s and *lub*s must be provided.

$\mathcal{TTS}_1$ and $\mathcal{TTS}_2$ are examples of closed granularity systems. The usefulness of the lattice structure has been shown, for example, in the logical design of temporal databases with multiple granularities [WBBJ97], and will be shown in Section 3 for temporal constraint satisfaction.

2.2 Data 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, $\lceil z \rceil_\mu^\nu$ is undefined if $\mu(z) \not\subseteq \nu(z')$ for all z'; otherwise, $\lceil z \rceil_\mu^\nu = z'$, 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, $\lceil z \rceil_{\texttt{second}}^{\texttt{month}}$ gives the month that includes the second z. Note that while $\lceil z \rceil_{\texttt{second}}^{\texttt{month}}$ is always defined, $\lceil z \rceil_{\texttt{week}}^{\texttt{month}}$ is undefined if week z falls between two months. Similarly, $\lceil z \rceil_{\texttt{day}}^{\texttt{b-day}}$ is undefined if day z is not a business day. Note that if $\mu \preceq \nu$, then the function $\lceil z \rceil_\mu^\nu$ is defined for each index value z. For example, since $\texttt{day} \preceq \texttt{week}$, $\lceil z \rceil_{\texttt{day}}^{\texttt{week}}$ is always defined, i.e., for each day we can find the week that contains it. The notation $\lceil z \rceil^\nu$ is used when the source type can be left implicit (e.g., when we are dealing with a fixed set of granularities having a distinguished basic type).

Another direction of the above transformation is as follows: Let μ and ν be temporal types, and z an integer. Define $\lfloor z \rfloor_\mu^\nu = (z', k)$, where the resulting pair identifies the k consecutive ticks of μ whose union yields $\nu(z)$, 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 $\lfloor z \rfloor_\mu^\nu$ is always defined. It is worth mentioning that in [TSQL2], it is assumed that a *calendar* subsystem provides a similar function that gives the conversion factor for a certain target type for each tick of a source type. For example, the function returns "29 days" for the month February, 1996.

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` and the conversion function used is 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 the quantitative temporal constraints to incorporate temporal types. We start with the definition of a temporal constraint with granularity. For simplicity, we assume that `second` is the implicit primitive type for the $\lceil\rceil$ functions, i.e., $\lceil z \rceil^{\mu}$ means $\lceil z \rceil^{\mu}_{\texttt{second}}$, and we work with only temporal types μ in $\mathcal{TTS}_1$ such that $\texttt{second} \trianglelefteq \mu$.

Definition Let $m \leq n$ be 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, t_2) *satisfies* $[m, n]\,\mu$) iff (1) $\lceil t_1 \rceil^{\mu}$ and $\lceil t_2 \rceil^{\mu}$ are both defined, and (2) $m \leq (\lceil t_2 \rceil^{\mu} - \lceil t_1 \rceil^{\mu}) \leq n$.

In the paper, given a TCG $[m, n]\,\mu$ we refer to the integer set $\{x \mid m \leq x \leq n\}$ as the *range* of the TCG.

Intuitively, for instants t_1 and t_2 (in terms of seconds), (t_1, t_2) satisfies $[m, n]\,\mu$ 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 the t_1-th and t_2-th seconds, respectively. The instants t_1 and t_2 are first translated in terms of μ, and then the difference is taken. If the difference is at least m and at most n, then the pair of instants is said to satisfy the constraint. 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. Similarly, (t_1, t_2) satisfies TCG $[-1, 1]\,\texttt{hour}$ if t_1 and t_2 are at most one hour apart (and the order of them is immaterial). 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` (i.e., as $[m', n']\,\texttt{second}$ for any m' and n'). 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, $[0, 86399]$ `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 $[0, 0]$ `day`; however, they do satisfy $[0, 86399]$ `second`.

Networks of constraints with granularities are defined as follows:

Definition A *constraint network (with granularities)* is a directed graph $(W, A, ,)$, where W is a finite set of variables, $A \subseteq W \times W$ and , 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, Y), the instants assigned to X and Y must satisfy the TCG.

Example 1 Figure 2 shows an example of a constraint network with granularities.

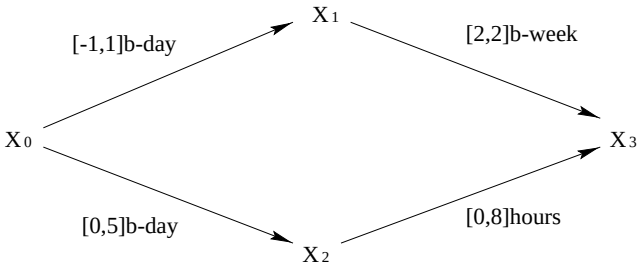

Figure 2: A constraint network with granularities.

The network involves three different granularities: business days (`b-day`), business weeks (`b-week`), and hours (`hour`). This network could be used, for example, to specify a pattern of events E_0, E_1, E_2, E_3 such that when their occurrence times are assigned to the variables X_0, X_1, X_2, X_3 respectively, we have i) E_0 and E_1 occur in the same, or in successive business days (the order of their occurrences is irrelevant), ii) E_2 occurs within 5 business days from the occurrence of E_0, iii) E_3 occurs within 8 hours from E_2 and in the second business week after E_1. □

For a given constraint network $\mathcal{CN}$, it is of practical interest to check if the network is consistent.

Definition A constraint network $\mathcal{CN} = (W, A, ,)$ is *consistent* if there exists an assignment from the set of positive integers to the variables in W such that for each arc $(X, Y) \in A$ the pair (t_X, t_Y) of values assigned to X and Y, respectively, satisfies each TCG in , (X, Y).

Determining the consistency of constraint networks with granularities turns out to be a difficult problem:

Theorem 3 *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 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 3.

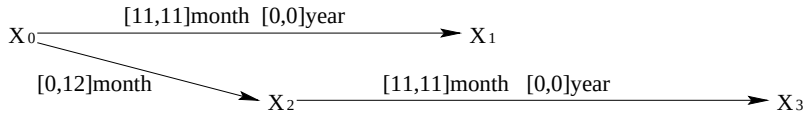

Figure 3: A constraint network with an implied disjunctive constraint.

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.

4 Approximate solutions for the consistency and minimal network problems

We consider here an approximate algorithm for consistency checking. The approach consists of using constraint propagation techniques coupled with conversion of constraints in different granularities. Constraints in each granularity are incrementally refined using the information from the original network. If the refinement of a constraint leads to an empty set for admissible values, the whole network is inconsistent. The algorithm also gives an approximation of what is usually called a minimal network. Intuitively, a minimal network is a network having the same solutions as the original one, but such that no constraint can be further refined. We also require that constraints in the minimal network are only in terms of the granularities appearing in the original network. Minimal networks are useful in several tasks such as deriving a solution for the given network or comparing two networks. In the following we start illustrating methods to convert constraints in terms of different granularities. The conversion of

constraints is an essential step of the approximate algorithm.

4.1 Conversion of constraints in different granularities

Consider the problem of converting a given TCG_1 in terms of μ_1 into a logically implied TCG_2 in terms of μ_2. (TCG_1 *logically implies* TCG_2 if any pair of time instants satisfying TCG_1 also satisfy TCG_2.) If we only have a total order of granularities with uniform ticks, like e.g., `minute`, `hour`, and `day`, then the conversion method 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.

Moreover, given an arbitrary TCG_1, and a granularity μ, it is not always possible to find a logically implied TCG_2 in terms of μ. For example, $[0,0]\,\texttt{day}$ does not logically imply $[m,n]\,\texttt{b-day}$ no matter what m and n are. The reason is that $[0,0]\texttt{day}$ is satisfied by any two events that happen during the same day, whether the day is a business day or a weekend day.

In our framework, we allow the conversion of a TCG in a constraint network $\mathcal{CN}$ into another TCG if the resulting constraint is implied by the set of all the TCGs in $\mathcal{CN}$. More specifically, a TCG $[m,n]\,\mu$ on arc (X,Y) in a network $\mathcal{CN}$ is allowed to be converted into $[m',n']\,\nu$ as long as $[m',n']\,\nu$ is *implied* by $\mathcal{CN}$, i.e., for any pair of values t_X and t_Y assigned to X and Y respectively, if (t_X,t_Y) satisfies $[m,n]\,\mu$ and t_X and t_Y belong to a solution of $\mathcal{CN}$, then (t_X,t_Y) also satisfies $[m',n']\,\nu$.

Example 2 Consider a network with three variables X, Y and Z as shown in Figure 4.

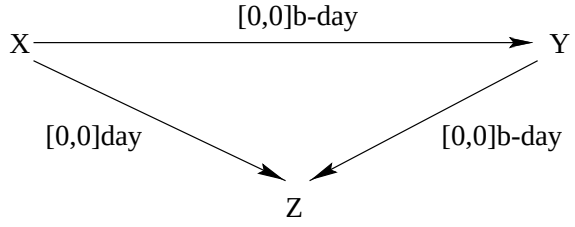

Figure 4: A network to illustrate conversion conditions.

It is clear that we may convert $[0,0]\,\texttt{day}$ on (X,Z) to $[0,0]\,\texttt{b-day}$ since for any events x and z assigned to X and Z respectively, if they belong to a solution of the whole structure, these two events must occur within the same *business day.* □

To guarantee that only allowed conversions are performed, given a constraint network, we assign to each variable X a temporal type μ_X obtained as the greatest lower bound (with respect to the finer-than

relation)[2] of all the temporal types appearing in TCGs involving X. Then, a TCG on variables X and Y can be converted in terms of a target temporal type ν only if ν covers a span of time equal or larger than the span of time covered by μ_X and by μ_Y. (The *span* of time covered by a temporal type μ is defined as the set $\bigcup_{i \in \mathcal{I}} \mu(i)$.) It is clear that any conversion performed under this condition is an *allowed* conversion.

Example 3 Consider the constraint network in Figure 4. Both μ_X and μ_Z are `b-day`. Hence, the constraint $[0,0]$`day` can be converted in terms of `b-day` since the temporal type `day` covers a span of time that includes that of `b-day`. Similarly, consider the network in Figure 2. The TCG in terms of `hour` cannot be converted in terms of `b-day` or `b-week` without considering the other constraints in the network. However, both μ_{X_2} and μ_{X_3} evaluate to the temporal type obtained from `hour` by dropping all ticks not included in a business day. Since both `b-day` and `b-week` cover the same span of time as this type, the constraint $[0,8]$`hour` can be converted in terms of `b-day` and `b-week`, obtaining $[0,1]$`b-day` and $[0,0]$`b-week`, respectively. □

From a practical point of view a specific representation of types should be chosen so that *glb*s can be effectively computed, and covered spans of time can be effectively compared.

In Figure 5 we propose a conversion method that is sufficiently general to apply to all temporal types, provided that the aforementioned condition is satisfied. This method is based on the two functions $mindist()$ and $maxdist()$. Intuitively, $mindist(\mu_1, m, \mu_2)$ denotes the minimal distance (in terms of the number of ticks of μ_2) between all pairs of instants in a tick of μ_1 and in the mth tick after it, respectively. For example, $mindist($`b-week`$, 1,$ `day`$) = 3$, i.e., the minimum distance in terms of days between two events that occur in two different business weeks is 3 (one instant on Friday and the other on Monday). The definition restricts the considered pairs of instants to those in which the first instant is included in μ_X and the second in μ_Y. Indeed, only these pairs are candidate solutions for the constraint on the arc (X, Y), and this restriction improves the precision of the conversion as well as it detects some inconsistencies. For simplicity, in our examples, we assume that μ_X and μ_Y cover the same span of time as μ_1, so that the restriction does not influence the result. The value $maxdist(\mu_1, n, \mu_2)$ is the corresponding maximum distance. For example, $maxdist($`b-week`$, 1,$ `day`$) = 11$, and $maxdist($`b-day`, 1,`day`$) = 3$.

The values of these functions cannot be automatically obtained for general (infinite) temporal types,

[2] We proved that TTS_1 forms a lattice with respect to the finer-than relation. Note also that for types μ_1 and μ_2 in TTS_1 with `second` $\trianglelefteq \mu_1$ and `second` $\trianglelefteq \mu_2$, we have `second` $\trianglelefteq glb(\mu_1, \mu_2)$ and `second` $\trianglelefteq lub(\mu_1, \mu_2)$. It follows that `second` groups-into the *glb* of any finite set of the considered temporal types.

INPUT: a network $\mathcal{CN}$ with a TCG $[m, n]\,\mu_1$, with $n \geq 0$, associated with an arc (X, Y), a target type μ_2, s.t. $\forall i, t\ (t \in \mu_X(i) \cup \mu_Y(i) \Rightarrow \exists j\ t \in \mu_2(j))$.

OUTPUT: a logically implied TCG $[\overline{m}, \overline{n}]\mu_2$ for (X, Y) or **undefined**.

METHOD:

Step 1 **if** $m < 0$ **then** $\overline{m} = -maxdist(\mu_1, |m|, \mu_2)$
else $\overline{m} = mindist(\mu_1, m, \mu_2)$

Step 2 $\overline{n} = maxdist(\mu_1, n, \mu_2)$

Step 3 **if** either $\overline{m}$ or $\overline{n}$ is **undefined**, **then return undefined**
else return $[\overline{m}, \overline{n}]\,\mu_2$.

where

$mindist(\mu_1, m, \mu_2) = min(S)$ if $S \neq \emptyset$, **undefined** otherwise, where
$S = \{\lceil t_2 \rceil^{\mu_2} - \lceil t_1 \rceil^{\mu_2} \mid \lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ are both defined, and $\lceil t_2 \rceil^{\mu_1} - \lceil t_1 \rceil^{\mu_1} \geq m\}$;

$maxdist(\mu_1, n, \mu_2) = max(R)$ if $R \neq \emptyset$, **undefined** otherwise, where
$R = \{\lceil t_2 \rceil^{\mu_2} - \lceil t_1 \rceil^{\mu_2} \mid \lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ are both defined, and $\lceil t_2 \rceil^{\mu_1} - \lceil t_1 \rceil^{\mu_1} \leq n\}$;

$\mu_Z = glb(\nu_1, \ldots, \nu_k)$ if $\nu_1, \ldots, \nu_k$ are the types in the TCGs of $\mathcal{CN}$ involving node Z.

Figure 5: A general method for the conversion of constraints

however they can be computed efficiently when the involved temporal types are periodic, as in most practical cases. The computation of *mindist*() and *maxdist*() is more involved when we want to check the condition, appearing in their specification, on $\lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ being both defined. Note that omitting this check leads to a still sound but less precise conversion. The other condition involving μ_X and μ_Y appears in the input description. This cannot be ignored since it guarantees an allowed conversion. As we have observed earlier, this condition can be automatically checked for certain type specifications, as for example those using periodic descriptions.

Example 4 By exploiting the periodicity of `b-day`, we can compute *mindist*(`b-day`,k,`day`) and *maxdist*(`b-day`,k,`day`), assuming that μ_X and μ_Y cover the same span of time as `b-day`, as follows:

$$mindist(\texttt{b-day},k,\texttt{day}) = k + 2 * (k \text{ DIV } 5)$$

$$maxdist(\texttt{b-day},k,\texttt{day}) = \begin{cases} 0 & \text{if } k = 0 \\ k + 2 * (1 + ((k-1) \text{ DIV } 5)) & \text{otherwise} \end{cases}$$

□

The method imposes $n \geq 0$ for the input constraint. This is not a limitation since any constraint $[-n, -m]\,\mu$ on (X, Y) with $m, n > 0$ can be expressed as $[m, n]\,\mu$ on (Y, X). When only the lower bound is negative, it is sufficient to consider its absolute value and treat it exactly as the upper bound except for reversing the sign of the result. For example, for $[-1, 1]$ `week` we derive the upper bound 13 for `day` according to the method. Since the absolute value for -1 is 1 we derive $[-13, 13]$ `day` as the implied constraint.

Theorem 4 *The general conversion method is correct: Any constraint obtained as output is implied by the original network.*

In application contexts where many temporal types are used and the information about the relationship between all pairs of types is unavailable or impractical to obtain, we propose an alternative conversion method. This method, however, involves a loss of precision compared with the one given above.

In most cases, we know the relationship between each temporal type and a reference (primitive) type μ that groups into all the other types of the network, as e.g., `second`. In Figure 6 we give an alternative conversion method that exploits only this information.

The conversion method uses the new functions *minsize*(), *maxsize*(), and *mindist-prim*(). Intuitively, $minsize(\mu_2, k)$ and $maxsize(\mu_2, k)$ denote respectively the minimum and maximum length of k consequent ticks of μ_2, expressed in ticks of the primitive type. For example, $minsize(\texttt{month}, 1) = 28$, and $maxsize(\texttt{month}, 1) = 31$ if `day` is the primitive type. *mindist-prim*(μ_1, m) denotes the minimal

INPUT: a network $\mathcal{CN}$, a TCG $[m,n]\,\mu_1$, with $n \geq 0$, associated with an arc (X,Y), a primitive type μ, a target type μ_2, s.t. $\forall i,t\ (t \in \mu_X(i) \cup \mu_Y(i) \Rightarrow \exists j\ t \in \mu_2(j))$.

OUTPUT: a logically implied TCG $[\overline{m},\overline{n}]\mu_2$ for (X,Y) or **undefined**.

METHOD:

if $\exists i, i', i'', m'$ s.t. $m \leq m' \leq n$, $\mu_1(i) \cap \mu_X(i') \neq \emptyset$, and $\mu_1(i+m') \cap \mu_Y(i'') \neq \emptyset$ **then**

Step 1 **if** $m < 0$ **then** $\overline{m} = -min(R)$, **where**

$R = \{r \mid minsize(\mu_2, r+1) \geq maxsize(\mu_1, |m|+1)\}$

else $\overline{m} = min(R) - 1$, **where**

$R = \{r \mid maxsize(\mu_2, r) > mindist\text{-}prim(\mu_1, m)\}$

Step 2 $\overline{n} = min(S)$, **where**

$S = \{s \mid minsize(\mu_2, s+1) \geq maxsize(\mu_1, n+1)\}$

Step 3 **return** $[\overline{m},\overline{n}]\,\mu_2$.

else return undefined

where

$minsize(\nu, r) = min(Q)$ and $maxsize(\nu, r) = max(Q)$, with

$Q = \{k \mid \exists j, i,\ k = min\{k' \mid (\nu(j) \cup \cdots \cup \nu(j+r-1)) \subseteq (\mu(i) \cup \cdots \cup \mu(i+k'-1))\}\}$

$mindist\text{-}prim(\mu_1, m) = min\{d \mid \exists i, j\ \mu(j) \cap \mu_1(i) \neq \emptyset$ and $\mu(j+d) \cap \mu_1(i+m) \neq \emptyset\}$

Figure 6: Conversion of constraints through a primitive type

distance in terms of ticks of the primitive type between all pairs of instants in a tick of μ_1 and in the mth tick after it, respectively. The conversion method returns **undefined** when it is not possible to evaluate this distance for any pair of instants that are a candidate solution respectively for variables X and Y.

Steps 1 and 2 compute respectively the new minimum and maximum in terms of the target type. When both m and n are non-negative, the maximum and minimum distances identified by the constraint in terms of the primitive type are $maxsize(\mu_1, n+1)$ and $mindist\text{-}prim(\mu_1, m)$, respectively. 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. Similar to the general method, when the lower bound is negative, we consider its absolute value and we treat it exactly as the upper bound except for reversing the sign of the result.

Theorem 5 *The conversion method in Figure 6 is correct: Any constraint obtained as output is implied by the original network.*

We assume that the procedure implementing this method can access a table containing the values of $minsize(\nu, k)$, $maxsize(\nu, k)$, $mindist\text{-}prim(\nu, k)$ for each considered type ν and positive integer k limited by some constant. In general these values are available and, for most granularities, they can be automatically derived. 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 Figure 6). If the implementation of the condition in the main **if** statement turns out to be difficult for a particular type specification, the condition can be ignored, resulting in a still sound but less precise conversion.

The use of an intermediate granularity (the primitive type) in the conversion introduces an approximation, since the information about the structure of the source granularity is lost once the spans of time corresponding to the least and upper bounds of the original constraint are computed in terms of the primitive type.

Example 5 Consider the conversion of the TCG $[1, 1]$`month` into a TCG in terms of `hour` using `second` as the primitive type. The minimum span of time to cover two instants being 1 month apart is 2 seconds (the last second of a month and the next second). Hence, mindist-prim(`month`,1)=1, since these 2 seconds are contiguous. Once we have obtained this value we lose the structure of `month`, and, hence, the fact that the 2 seconds were overlapping 2 months. Then, the conversion method looks for the minimum number of ticks of the target type (`hour`) covering 2 seconds, and the answer is obviously 1

(whose corresponding distance between ticks is 0). Hence, the method in Figure 6 returns the TCG $[0, 24 * 62 - 1]$`hour`, which is indeed implied by the original TCG as indicated by Theorem 5. However, it is not the tightest constraint that a conversion could give. The *direct* conversion method given in Figure 5 does not suffer from this problem since it does not use an intermediate temporal type; it will return $[1, 24 * 62 - 1]$`hour`. □

4.2 The constraint propagation algorithm

In Figure 7 we provide a constraint propagation algorithm that we use to obtain approximate solutions of the consistency and minimal network problems.

INPUT: a network $\mathcal{CN} = (W, A, ,)$ with M the set of temporal types appearing in , .

OUTPUT: **inconsistent** or a network $\mathcal{CN}'$ with the same solutions as $\mathcal{CN}$, but s.t. for each TCG $[m, n]\,\mu$ on (X, Y) in $\mathcal{CN}$, a tighter TCG $[\overline{m}, \overline{n}]\,\mu$ (i.e., $\overline{m} \geq m$ and $\overline{n} \leq n$) is in $\mathcal{CN}'$ for the same arc.

METHOD:

Initialization: For each $\mu \in M$, let $C_\mu = (W, A_\mu, , _\mu)$, where $A_\mu = \{(X, Y) \mid , (X, Y)$ contains a TCG in terms of $\mu\}$ and $, _\mu(X, Y) = [m, n]\,\mu$ if $[m, n]\,\mu$ is in $, (X, Y)$.

Step 1 Apply path consistency to each C_μ discarding any resulting TCG with $-\infty$ or $+\infty$ in its range.

Step 2 For each $\mu, \nu \in M$ and each TCG $[m, n]\,\mu$ in C_μ, perform the conversion in terms of ν if it is allowed, applying one of the conversion methods illustrated above, and discarding any resulting TCG with $-\infty$ or $+\infty$ in its range. If an allowed conversion returns **undefined**, then $, _\mu(X, Y) = \{False\}$. Otherwise, insert the resulting TCG in $, _\nu(X, Y)$ taking the intersection of the range of values if another TCG is present in $, _\nu(X, Y)$. When the intersection is empty, $, _\nu(X, Y) = \{False\}$.

Step 3 **if** *False* appears in any $, _\mu(X, Y)$ with $\mu \in M$ **then return inconsistent**. **if** new TCGs have been derived **then goto** Step 1, **else return** $\mathcal{CN}' = (W, A', , ')$ where $A' = \bigcup_{\mu \in M} A_\mu$ and for each $(X, Y) \in A'$, $'(X, Y) = \bigcup_{\mu \in M} , _\mu(X, Y)$

Figure 7: The constraint propagation algorithm

The algorithm consists of an initialization followed by a two-step main loop. Step 3 establishes the termination condition. The initialization partitions the TCGs in $\mathcal{CN}$ into groups, each group having

TCGs in terms of the same temporal type. Then, the propagation of the constraints in $\mathcal{C}_\mu$ is a problem known as the Simple Temporal Problem [DMP91]. The first step of the main loop is the application of a path consistency algorithm within each $\mathcal{C}_\mu$. Path consistency is a simple technique that eliminates any inconsistency in each subnetwork involving 3 variables. Intuitively, if X_1, X_2, X_3 are variables in the constraint network the constraints on arcs (X_1, X_2) and (X_2, X_3) are composed and the result is intersected with the constraint on the direct arc (X_1, X_3). Empty intersections denote an inconsistency. The algorithm applies an appropriate strategy to consider all subnetworks of size 3. Path consistency for general networks only detects local inconsistencies, but it has been proved complete for consistency in the Simple Temporal Problem. Hence, if one of our $\mathcal{C}_\mu$ networks is inconsistent the path consistency step will detect it.

Since constraints expressed in a granularity could imply constraints in other granularities, Step 2 tries to convert the TCGs 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, each TCG in $\mathcal{C}_\mu$ is converted into one in terms of ν, which is added into $\mathcal{C}_\nu$. If another TCG on the same arc is present in $\mathcal{C}_\nu$, a single TCG having as range of values the intersection of the ranges allowed by the two TCGs is kept in $\mathcal{C}_\nu$. Steps 1 (path consistency) and 2 (conversion) are repeated until no new TCGs appear in any group. An inconsistency can be detected in two ways: First, if an allowed conversion returns **undefined**, this means that the TCG given as input cannot be satisfied, and, hence, the whole network is inconsistent. Second, when the intersection between a derived TCG and an original or previously derived TCG on the same variables is empty, no assignment can satisfy the constraints for the corresponding arc and the network is inconsistent.

Example 6 Consider the application of the algorithm to the constraint network in Figure 2. At the first iteration, Step 1 only derives the constraint $X_2 - X_1 \in [-1, 6]$ in terms of `b-day`. Then, this and the other 2 constraints in `b-day` are converted in terms of `b-week` and `hour`, generating 6 new constraints. The remaining 2 original constraints ($X_3 - X_2 \in [0, 8]$ `hour` and $X_3 - X_1 \in [2, 2]$ `b-week`) are also converted in terms of the other 2 granularities. The result of Step 1 at the second iteration is shown in Figure 8 (We omit the constraints in terms of `hour`).

Note that, at this point, all constraints have been converted in terms of other granularities and constraint propagation has been applied to each "single-granularity" network until no further refinement was possible. Considering the global set of constraints, however, some refinement can still be obtained. Indeed, the conversion of $X_1 - X_0 \in [-1, -1]$ `b-week` into $X_1 - X_0 \in [-9, -1]$ `b-day`, performed by the algorithm in Step 2 at the second iteration, allows to refine the constraint on X_0 and X_1. The third

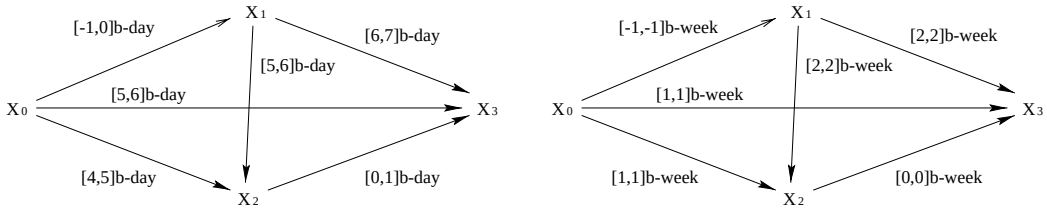

Figure 8: An intermediate step of the propagation algorithm.

iteration does not modify any constraint and the algorithm terminates. Figure 9 shows the final result in terms of constraints in `b-day`. The constraints in `b-week` are the same as in Figure 8, and we omit the constraints in `hour`.

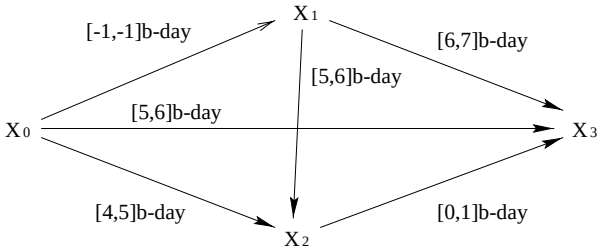

Figure 9: The constraints in `b-day` as returned by the algorithm.

□

We say that the algorithm is *sound* if any assignment satisfying the given constraint network $\mathcal{CN} = (W, A, ,)$ also satisfies $\mathcal{CN}' = (W, A', , ')$ as returned by the algorithm. Hence, when the algorithm returns **inconsistent** there must be no assignment satisfying $\mathcal{CN}$.

Theorem 6 *The approximate propagation algorithm is sound and terminates. Let c be the time taken to convert a constraint, n be the number of variables in the network, $|M|$ be the number of temporal types appearing in the explicit constraints, and w be the maximum cardinality of the ranges in the TCGs (original and derived by conversion). Then, the algorithm requires in the worst case time $O(c * n^5 * |M|^2 * w)$.*

It is important to interpret this upper bound on the complexity of the algorithm isolating the components due to the different steps of the algorithm. Step 1 (path consistency) takes time $O(n^3 * |M|)$, while Step 2 (conversion) takes time $O(c * n^2 * |M|)$. Note that, in practice, the time c can be considered a constant factor. Hence, the overall complexity can be expressed as $O((n^3 + cn^2) * |M| * I)$ where I is

the number of iterations. When $|M| = 1$, i.e., the constraints are all expressed in the same granularity, the problem becomes an STP; hence, a single iteration is sufficient, and the algorithm takes time $O(n^3)$. When $|M| \geq 2$ we give, as an upper bound for the number of iterations, $n^2 * |\mathcal{M}| * w$ where w is the maximum cardinality of the ranges in the TCGs (original and derived by conversion). This bound assumes that in the worst case at each iteration only one constraint is tightened and by only one unit. In this bound, we cannot consider only the original constraints, since new constraints are derived by conversion and they participate in successive iterations. For example, even if the number of restrictions for $[0,0]$`month` is 0, if the granularity `day` appears in the same network, a constraint $[0,30]$`day` is generated, whose number of potential restrictions is 30. However, we believe that this bound can be substantially improved.

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.

We note that the NP-hardness of the consistency checking implies that a *complete*, sound algorithm for constraint propagation on constraint networks is unlikely to be polynomial. A *complete* algorithm here is one that always derives the minimal network. Indeed, if such a polynomial algorithm existed, consistency checking would be polynomial since the tightest constraint between each pair of variables in an inconsistent constraint network is {false} (i.e., not satisfiable).

5 Related work

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, MMCR92]), is quite 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 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 (a) and (b) on the relationships among types (see page 5). The result is a set of temporal types totally ordered by the *finer-than* 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 each value is translated in terms of a basic granularity. In this paper we propose what we consider a more natural semantics for constraints with granularities, which must be supported by different algorithms.

A 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 time granularities 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 (b) on type relationships, while additional restrictions on type relationships are specified through logic axioms.

With respect to 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. 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.

6 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 with respect to similar constraint networks without time granularities (or with a single granularity).

We do not discuss in this paper another type of constraints that are related to granularities, namely, temporal functional dependencies with granularities [WBBJ97]. Intuitively, a temporal functional dependency specifies that certain attributes do not change values during a tick of a certain temporal type. For example, the teaching assistant of a course does not change during a `semester`. [WBBJ97] concentrated on the elimination of redundancies introduced by such constraints in a temporal relational model.

We have applied the proposed framework for time granularities in several areas: federated temporal databases [WJS95], logical design of temporal databases with multiple granularities [WBBJ97], querying temporal databases with semantic assumptions [BWBJ95], and mining large event sequences for complex temporal relationships [BWJ96]. In particular, constraint propagation in a network with multiple granularities is used in [BWJ96] to significantly prune the search space during a data mining process. Working prototypes have been developed for the last two applications that we just mentioned. We refer the interested reader to the cited papers.

Regarding temporal constraints with granularities, the constraint propagation algorithm can also be applied in several artificial intelligence application areas where temporal constraint satisfaction algorithms are already employed, as, for example, in scheduling and planning. There are also some proposals on enhancing the deductive power of temporal databases exploiting known constraint propagation techniques [BCTP95]. Even if we did not investigate this research direction, we believe that the methods proposed in this paper would be an intuitive extension to support multiple granularities.

With respect to temporal logics, an interesting topic would be the study of metric temporal logics along the work of [MON96], but extending these formalisms to deal with 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*, San Jose, CA, 1995, pp. 257–268.

[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*, Montreal, Canada, 1996, pp. 68–78.

[BCTP95] V. Brusoni, L. Console, P. Terenziani, and B. Pernici, Extending Temporal Relational Databases to Deal with Imprecise and Qualitative Temporal Information, in *Proc. of Int. Workshop on Temporal Databases*, Zurich, Switzerland, Springer-Verlag, 1995, pp. 3-22.

[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 (1993) pp. 141-171.

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

[CSS94] R. Chandra, A. Segev, and M. Stonebraker, Implementing calendars and temporal rules in next generation databases, in *Proc. of ICDE*, 1994, pp. 264-273.

[Dea89] T. Dean, Artificial intelligence: Using temporal hierarchies to efficiently maintain large temporal databases, JACM, 36 (1989) pp. 687-718.

[DMP91] R. Dechter, I. Meiri, and J. Pearl, Temporal constraint networks, Artificial Intelligence, 49 (1991) pp. 61-95.

[TSQL2] *The TSQL2 Temporal Query Language*, R. T. Snodgrass ed., Kluwer Academic Pub., 1995.

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

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

[Hob85] J.R. Hobbs, Granularity, in *Proc. of IJCAI*, Los Angeles, CA, 1985, pp. 432–435.

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

[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*, Tokyo, Japan, 1992, pp. 702–712.

[MON96] A. Montanari. *Metric and Layered Temporal Logic for Time Granularity*, ILLC Dissertation Series 1996-02, University of Amsterdam, 1996.

[NS92] M. Niezette and J. Stevenne, An efficient symbolic representation of periodic time, in *Proc. of CIKM*, Baltimore, 1992, pp. 161–168.

[WBBJ97] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia, Logical design for temporal databases with multiple granularities, ACM TODS, (1997).

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

Proofs

Proof of Theorem 1

We prove that for each pair of temporal types μ_1 and μ_2, there exist a unique least upper bound, $lub(\mu_1, \mu_2)$, and a unique greatest lower bound, $glb(\mu_1, \mu_2)$. Before doing that, we introduce the following notation: Given two non-empty sets S_1 and S_2 of elements in $\mathcal{A}$, $S_1 \ll S_2$ holds if each number in S_1 is strictly less than each number in S_2. (Formally, $S_1 \ll S_2$ if $\forall x \in S_1 \forall y \in S_2 (x <_{\mathcal{A}} y)$.) Moreover, we say that a set $\mathbf{S}$ of non-empty sets of elements in $\mathcal{A}$ is *monotonic* if for each pair of sets S_1 and S_2 in $\mathbf{S}$ either $S_1 \ll S_2$ or $S_2 \ll S_1$. It is easily seen that a temporal type μ can be viewed as a monotonic set of non-empty sets of elements in $\mathcal{A}$, namely $\{\mu(i) | \mu(i) \neq \emptyset\}$. On the other hand, a monotonic set $\mathbf{S}$ of non-empty sets of elements in $\mathcal{A}$, such that each element can be indexed by a different $i \in \mathcal{I}$ preserving the order imposed by the monotonicity (we call this property *indexability*), can be viewed as a temporal type μ, where for each $i \in \mathcal{I}$, $\mu(i)$ is the set in $\mathbf{S}$ indexed by i, or $\mu(i) = \emptyset$ if i is not used in the above indexing. Let μ_1 and μ_2 be two temporal types and $\mathbf{S}_1$ and $\mathbf{S}_2$ the sets corresponding to μ_1 and μ_2, respectively, as discussed above.

Let $\mathbf{S}_{glb} = \{S_1 \cap S_2 \mid S_1 \in \mathbf{S}_1, S_2 \in \mathbf{S}_2 \text{ and } S_1 \cap S_2 \neq \emptyset\}$. Since $\mathbf{S}_1$ and $\mathbf{S}_2$ are both monotonic and indexable by $\mathcal{I}$, $\mathbf{S}_{glb}$ is also monotonic and indexable by $\mathcal{I}$. It is easily seen that the type μ corresponding to $\mathbf{S}_{glb}$ is the unique $glb(\mu_1, \mu_2)$.

Consider now $lub(\mu_1, \mu_2)$. Let $\mathbf{S}_b = \{S \subseteq \mathcal{A} \mid \emptyset \neq \mu_k(j) \subseteq S \text{ for some } k \in \{1,2\} \text{ and } j\}$ and $\mathbf{S}_{ub} = \{S \in \mathbf{S}_b \mid \text{for each } k \in \{1,2\} \text{ and each } i \in \mathcal{I}, \mu_k(i) \ll S \text{ or } S \ll \mu_k(i) \text{ or } \mu_k(i) \subseteq S\}$. Intuitively, $\mathbf{S}_b$ consists of all the sets that contain at least one tick of μ_1 or μ_2, and $\mathbf{S}_{ub}$ consists of

the sets S such that S is in $\mathbf{S}_b$ and each tick of μ_1 and μ_2 it is either entirely in or entirely out of S. Finally, let $\mathbf{S}_{lub} = \{S \mid S \in \mathbf{S}_{ub} \text{ and } \nexists S' \in \mathbf{S}_{ub} \text{ with } S' \subset S\}$. Clearly, $\mathbf{S}_{lub} \subseteq \mathbf{S}_{ub} \subseteq \mathbf{S}_b$.

The following three steps suffice to show that μ_1 and μ_2 have a unique least upper bound: (a) We first prove that the set $\mathbf{S}_{lub}$ is monotonic and indexable by $\mathcal{I}$. By (a), there exists μ that corresponds to $\mathbf{S}_{lub}$. We then establish that (b) $\mu_1 \preceq \mu$ and $\mu_2 \preceq \mu$ and (c) for all μ', if $\mu_1 \preceq \mu'$ and $\mu_2 \preceq \mu'$, then $\mu \preceq \mu'$.

For (a), we first prove that the sets in $\mathbf{S}_{lub}$ are disjoint. That is, given sets S and S' in $\mathbf{S}_{lub}$, we prove that $x \in S$ and $x \in S'$, for some x, implies $S = S'$. By the fact that S is in $\mathbf{S}_{lub}$, $x \in S$ implies $x \in \mu_k(j)$ for some j and $k \in \{1, 2\}$. Since $x \in \mu_k(j)$ and $x \in S$, we know $\mu_k(j) \not\ll S$ and $S \not\ll \mu_k(j)$. It follows from the condition defining $\mathbf{S}_{ub}$ that $\mu_k(j) \subseteq S$. Analogously, $\mu_k(j) \subseteq S'$. We now consider a particular relation θ: For each $k' \in \{1, 2\}$, $i \in \mathcal{I}$ and non-empty set S_1 of elements in $\mathcal{A}$, let $\mu_{k'}(i)\theta S_1$ be true iff $\mu_{k'}(i) \not\ll S_1 \wedge S_1 \not\ll \mu_k(i)$. We denote with $C(\mu_k(j))$ the set obtained by recursively applying the following step $C^n(\mu_k(j)) = C^{n-1}(\mu_k(j)) \cup S^n$ where $C^0(\mu_k(j)) = \mu_k(j)$ and S^n is the union of all ticks $\mu_{k'}(i)$ such that $\mu_{k'}(i)\theta C^{n-1}(\mu_k(j))$. We show $S = S'$ by proving that (i) $C(\mu_k(j)) \in \mathbf{S}_{ub}$ and (ii) $C(\mu_k(j)) \subseteq S$ and $C(\mu_k(j)) \subseteq S'$. Indeed, (i) and (ii) imply $C(\mu_k(j)) = S$ and $C(\mu_k(j)) = S'$ since otherwise (i) and (ii) contradict the fact that S and S' are both in $\mathbf{S}_{lub}$.

Consider (i). $C(\mu_k(j))$ is in $\mathbf{S}_b$ since $\mu_k(j) \subseteq C(\mu_k(j))$. Moreover, given an arbitrary tick $\mu_{k'}(i)$, if $\mu_{k'}(i)$ is not a subset of $C(\mu_k(j))$, then $\mu_{k'}(i)\theta C^n(\mu_k(j))$ is not true for any $n \geq 0$. It follows that $\mu_{k'}(i) \ll C(\mu_k(j))$ or $C(\mu_k(j)) \ll \mu_{k'}(i)$. Hence, $C(\mu_k(j))$ is in $\mathbf{S}_{ub}$. Consider (ii). We know that $C^0(\mu_k(j)) = \mu_k(j) \subseteq S$. Consider $C^1(\mu_k(j))$. Assume $C^1(\mu_k(j)) \not\subseteq S$. Thus there exists $\mu_{k'}(i)$ such that $\mu_{k'}(i) \subseteq C^1(\mu_k(j))$ but $\mu_{k'}(i) \not\subseteq S$. Hence, $\mu_{k'}(i)\theta C^0(\mu_k(j))$, i.e., $\mu_{k'}(i) \not\ll C^0(\mu_k(j))$ and $C^0(\mu_k(j)) \not\ll \mu_{k'}(i)$. Since $C^0(\mu_k(j)) \subseteq S$, it follows that $\mu_{k'}(i) \not\ll S$ and $S \not\ll \mu_{k'}(i)$. Since, in addition, $\mu_{k'}(i) \not\subseteq S$, this means that S is not in $\mathbf{S}_{ub}$, which contradicts the fact that S is in $\mathbf{S}_{lub}$. Continuing by induction with the same reasoning, we have that $C(\mu_k(j)) \subseteq S$. By symmetry, we have $C(\mu_k(j)) \subseteq S'$.

To conclude step (a), assume $\mathbf{S}_{lub}$ is not monotonic. Let $S \neq S'$ be in $\mathbf{S}_{lub}$ such that $S \not\ll S'$ and $S' \not\ll S$. Then (1) there exists $x \in S'$ such that $\{x\} \not\ll S$ and $S \not\ll \{x\}$, or (2) there exists $x \in S$ such that $\{x\} \not\ll S'$ and $S' \not\ll \{x\}$. For case (1), since x is in S', as we have shown before, there must exist k and j such that $x \in \mu_k(j)$ and $\mu_k(j) \subseteq S'$. By definition of $\mathbf{S}_{ub}$, either $\mu_k(j) \subseteq S$ or $\mu_k(j) \ll S$ or $S \ll \mu_k(j)$. Since S and S' are disjoint, $\mu_k(j) \not\subseteq S$. Therefore, $\mu_k(j) \ll S$ or $S \ll \mu_k(j)$. This contradicts the fact that $\{x\} \not\ll S$ and $S \not\ll \{x\}$. Case (2) leads to a contradiction by symmetry. This concludes the proof for (a) since the indexability of $\mathbf{S}_{lub}$ by $\mathcal{I}$ is trivial.

For (b), by construction, for each tick in μ_1 and μ_2, $\mathbf{S}_{ub}$ contains at least one set S equal or including

that tick. The minimization in $\mathbf{S}_{lub}$ takes the smallest of these sets. Hence, the type μ corresponding to $\mathbf{S}_{lub}$ is coarser than μ_1 and μ_2.

For (c), suppose that μ' is a type coarser than μ_1 and μ_2. We show that $\mu \preceq \mu'$. Consider any $i \in \mathcal{I}$ such that $\mu(i) \neq \emptyset$. Since $\mu(i) \in \mathbf{S}_{lub}$ and hence $\mu(i) \in \mathbf{S}_b$, it follows that $\mu(i)$ contains a tick $\mu_k(j)$ for some $k \in \{1,2\}$ and $j \geq 1$. Since $\mu_k(j) \subseteq \mu(i)$ and $\mu(i) \in \mathbf{S}_{lub}$, $\mu(i) = C(\mu_k(j))$ as in the proof for (a). Since $\mu_k \preceq \mu'$, there exists r such that $\mu_k(j) \subseteq \mu'(r)$. We now show that $\mu(i) = C(\mu_k(j)) \subseteq \mu'(r)$. Suppose otherwise, i.e., $C(\mu_k(j)) \not\subseteq \mu'(r)$. By the definition of $C(\mu_k(j))$ and since $C^0(\mu_k(j)) = \mu_k(j) \subseteq \mu'(r)$, there exists integer $n > 0$ such that $C^{n-1}(\mu_k(j)) \subseteq \mu'(r)$ but $C^n(\mu_k(j)) \not\subseteq \mu'(r)$. Hence, there exists $\mu_{k_1}(l)$ in $C^n(\mu_k(j)) - C^{n-1}(\mu_k(j))$ such that $\mu_{k_1}(l) \not\subseteq \mu'(r)$. By definition, $\mu_{k_1}(l) \theta C^{n-1}(\mu_k(j)$, i.e., $\mu_{k_1}(l) \not\ll C^{n-1}(\mu_k(j)$ and $C^{n-1}(\mu_k(j) \not\ll \mu_{k_1}(l)$. Since $C^{n-1}(\mu_k(j)) \subseteq \mu'(r)$ we have $\mu_{k_1}(l) \not\ll \mu'(r)$ and $\mu'(r) \not\ll \mu_{k_1}(l)$. Since $\mu_{k_1} \preceq \mu'$ there exists $r' \neq r$ such that $\mu_{k_1}(l) \subseteq \mu'(r')$. Therefore, we derive $\mu'(r') \not\ll \mu'(r)$ and $\mu'(r) \not\ll \mu'(r')$, a contradiction since each temporal type is monotonic. Thus, $\mu(i) = C(\mu_k(j)) \subseteq \mu'(r)$ and hence $\mu \preceq \mu'$.

Proof of Theorem 2

First note that if a temporal type system has restriction (b), then it is closed. Indeed, since types are restricted to be comparable, *glb* and *lub* are always among the given types. That is, if $\mu_1 \preceq \mu_2$, then $glb(\mu_1, \mu_2) = \mu_1$ and $lub(\mu_1, \mu_2) = \mu_2$.

For restrictions (1) and (3), we first independently show that if both temporal types μ_1 and μ_2 satisfy the restriction, then the *glb* and *lub* constructed in the proof of Theorem 1 also satisfy the restriction.

Consider restriction (1) which says that ticks are not allowed to have gaps within them. From the construction of *glb*, each tick of *glb* is obtained as intersection of a tick of μ and a tick of ν. Since these ticks have no gaps, i.e., they consists of a sequence of consecutive instants, their intersection is also a sequence of consecutive instants. Hence, *glb* satisfies this restriction. Consider now *lub*. According to the construction of *lub*, it is easy to see that each tick of *lub* it is the union of sets $S_1, \ldots, S_k$ (where k can also be infinite) corresponding to ticks of μ_1 and μ_2 and such that $S_i \cap S_{i+1} \neq \emptyset$ for each $1 \leq i < k$. Since each tick of μ_1 and μ_2 has no gap, each S_i contains consecutive instants and the same holds for the union of these sets.

Consider restriction (3), which states that ticks must cover all the absolute time. Since the set of ticks of both μ_1 and μ_2 cover all the absolute time, the set of ticks of *glb* that is obtained as intersection of the corresponding sets obviously covers all the absolute time. Moreover, by definition of *lub*, each

tick of μ_1 is contained in a tick of lub, and, since μ_1 covers all the absolute time, so does lub.

We can now conclude that any system having one or more restrictions among (b), (1), and (3) is closed if no additional restrictions (i.e., neither (2) nor (4)) are applied. Consider a system with restrictions (2) and (3). Note that restriction (2) is superseded by (3), since any type that covers all the absolute time must have contiguous ticks; hence, that system is closed. Finally, note that, as shown earlier, any system with restriction (b) is closed; hence, the system with restrictions (2) and (b) is closed, as well as the one with (4) and (b).

Proof of Theorem 3

Given a set of constraints, is there an assignment to the variables that satisfies the constraints? We show that answering this question is at least as hard as solving the SUBSET SUM problem (a knapsack variant). Consider a set of positive integers $n_1, \ldots, n_k$ and s. The SUBSET SUM problem consists in finding a subset such that the sum of its numbers is s. For each instance of this problem we can construct an instance of our problem as follows: Let $W = X_1, \ldots, X_{k+1}, V_1, \ldots, V_k, U_1, \ldots, U_k$. Consider the granularities `n-month` for $n = n_1, \ldots, n_k$ defined by grouping each consecutive n ticks of `month` into a single tick. For each $i = 1, \ldots, k$, create the following two constraints:
$(X_i, X_{i+1}) \in [0, n_i]$`month`.
$(X_1, X_{k+1}) \in [s, s]$`month`.
We now add another set of constraints with the only purpose to rule out all values between 1 and $n_i - 1$ in the first of the above two constraints. For each $i = 1, \ldots, k$ we add:
$(V_i, X_i) \in [0, 0]$`n`$_i$`-month`,
$(V_i, X_i) \in [n_i - 1, n_i - 1]$`month`,
$(U_i, X_{i+1}) \in [0, 0]$`n`$_i$`-month`.
$(U_i, X_{i+1}) \in [n_i - 1, n_i - 1]$`month`.
This set of constraints implies the disjunctions:
$(X_i, X_{i+1}) \in [0, 0]$`month` $\vee$ $[n_i, n_i]$`month` for each $i = 1, \ldots, k$. (This is similar to Figure 3.) If an assignment is found to satisfy the whole set of constraints specified above, the distance between the value of X_{i+1} and X_i will be 0 or n_i for each $i = 1, \ldots, k$. The set of indices i for which that value is different from 0 determines the subset of $\{n_1, \ldots, n_k\}$ being a solution of the SUBSET SUM problem. It is also easy to show that if an assignment is not found such a subset does not exist. Since the SUBSET SUM problem is NP-hard, and the transformation in the consistency problem can be done in polynomial time, determining consistency is also NP-hard.

Proof of Theorem 4

Assume that the conversion method of Figure 5 gives as output the TCG $[\overline{m}, \overline{n}]\mu_2$ for the given input TCG $[m, n]\mu_1$ associated with the arc (X, Y) in the network $\mathcal{CN}$ and the target type μ_2 satisfying the conditions specified in the method. Suppose, for proof by contradiction, $[\overline{m}, \overline{n}]\mu_2$ on (X, Y) is not implied by the network $\mathcal{CN}$. It follows that there exist two values x and y that assigned respectively to X and Y belong to a solution of the given network, satisfying the constraint $[m, n]\,\mu_1$ between X and Y, but not satisfying the constraint $[\overline{m}, \overline{n}]\,\mu_2$ between the same variables. By definition, if the constraint is not satisfied, one of the following must hold: (a) $\lceil x \rceil^{\mu_2}$ or $\lceil y \rceil^{\mu_2}$ are not defined, (b) $\lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} > \overline{n}$, (c) $\lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} < \overline{m}$.

Suppose (a) holds and, in particular, $\lceil x \rceil^{\mu_2}$ is not defined. From the definition of $\lceil\ \rceil$, this means that there does not exist j such that $x \in \mu_2(j)$. However, since x is part of a solution of the given network, $x \in \mu_X(k)$ for some positive integer k. Then, the condition $\forall i, t\ (t \in \mu_X(i) \cup \mu_Y(i) \Rightarrow \exists j\ t \in \mu_2(j))$ imposed by the method on its input, guarantees $\exists j\ x \in \mu_2(j)$, leading to a contradiction. The same arguments apply to $\lceil y \rceil^{\mu_2}$.

Suppose (b) holds. Let $\overline{n}' = \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2}$. Hence, $\overline{n}' > \overline{n}$. From the computation of $\overline{n}$ by the method we have $\overline{n} = max(Q)$, where $Q = \{r \mid \exists t_1, t_2,\ \lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ are both defined, $\lceil t_2 \rceil^{\mu_1} - \lceil t_1 \rceil^{\mu_1} \leq n$ and $r = min\{r' \mid \lceil t_2 \rceil^{\mu_2} - \lceil t_1 \rceil^{\mu_2} \leq r'\}\}$. Let $t_1 = x$ and $t_2 = y$. Clearly, $\lceil x \rceil^{\mu_X}$ and $\lceil y \rceil^{\mu_Y}$ are both defined since x and y are part of a solution. $\lceil y \rceil^{\mu_1} - \lceil x \rceil^{\mu_1} \leq n$ since (x, y) satisfies the TCG $[m, n]\mu_1$, and $min\{r' \mid \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} \leq r'\} = \overline{n}'$. Then, $\overline{n}' \in Q$ and, hence, $\overline{n}' \leq \overline{n} = max(Q)$. This is a contradiction since we assumed $\overline{n}' > \overline{n}$.

Finally, suppose (c) holds. Let $\overline{m}' = \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2}$. Hence, $\overline{m}' < \overline{m}$. We first consider the case when $m \geq 0$. From the computation of $\overline{m}$ by the method, we have $\overline{m}' < min(Q)$, where $Q = \{s \mid \exists t_1, t_2,\ \lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ are both defined, $\lceil t_2 \rceil^{\mu_1} - \lceil t_1 \rceil^{\mu_1} \geq m$ and $s = max\{s' \mid \lceil t_2 \rceil^{\mu_2} - \lceil t_1 \rceil^{\mu_2} \geq s'\}\}$. Let $t_1 = x$ and $t_2 = y$. Clearly, $\lceil x \rceil^{\mu_X}$ and $\lceil y \rceil^{\mu_Y}$ are both defined since x and y are part of a solution. $\lceil y \rceil^{\mu_1} - \lceil x \rceil^{\mu_1} \geq m$ since (x, y) satisfies the TCG $[m, n]\mu_1$, and $max\{s' \mid \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} \geq s'\} = \overline{m}'$. Then, $\overline{m}' \in Q$ and, hence, $\overline{m}' \geq \overline{m} = min(Q)$. This is a contradiction since we assumed $\overline{m}' < \overline{m}$. When $m < 0$, the fact that $\overline{m}' < \overline{m}$ can be interpreted as the distance between x and y in terms of ticks of μ_2 being greater (in absolute value) than the bound given by the method. The violation of the bound and the corresponding proof are, in this case, equivalent to point (b) above, since the values of x and y can be exchanged reversing the sign of the bounds. The conversion method, indeed, treats the negative lower bound as a positive upper bound reversing the sign of the result.

Proof of Theorem 5

Assume that the conversion method of Figure 6 gives as output the TCG $[\overline{m}, \overline{n}]\mu_2$ for the given input TCG $[m, n]\mu_1$ associated with the arc (X, Y) in the network $\mathcal{CN}$ and the target type μ_2 satisfying the conditions specified in the method. Suppose, for proof by contradiction, $[\overline{m}, \overline{n}]\mu_2$ on (X, Y) is not implied by the network $\mathcal{CN}$. It follows that there exist two values x and y that assigned respectively to X and Y belong to a solution of the given network, satisfying the constraint $[m, n]\,\mu_1$, but not satisfying the constraint $[\overline{m}, \overline{n}]\,\mu_2$. By definition, if the constraint is not satisfied, one of the following facts must hold: (a) $\lceil x \rceil^{\mu_2}$ or $\lceil y \rceil^{\mu_2}$ are not defined, (b) $\lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} > \overline{n}$, (c) $\lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} < \overline{m}$.

Suppose (a) holds and, in particular, $\lceil x \rceil^{\mu_2}$ is not defined. From the definition of $\lceil\,\rceil$, this means that there does not exist j such that $x \in \mu_2(j)$. However, since x is part of a solution of the given network, $x \in \mu_X(k)$ for some positive integer k. Then, the condition $\forall i, t\ (t \in \mu_X(i) \cup \mu_Y(i) \Rightarrow \exists j\ t \in \mu_2(j))$ imposed by the conversion method on its input, guarantees $\exists j\ x \in \mu_2(j)$, leading to a contradiction. The same arguments apply to $\lceil y \rceil^{\mu_2}$.

Suppose (b) holds. Let $\overline{n}' = \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2}$. Hence, $\overline{n}' > \overline{n}$. From the computation of $\overline{n}$ by the conversion method we have $minsize(\mu_2, \overline{n}+1) \geq maxsize(\mu_1, n+1)$. From the definition of $minsize()$ and $maxsize()$ this means that any $\overline{n}+1$ ticks of μ_2 are sufficient to cover the maximal span of time covered by $n+1$ ticks of μ_1. Since the constraint $[m, n]\mu_1$ is satisfied by (x, y), we have $\lceil y \rceil^{\mu_1} - \lceil x \rceil^{\mu_1} \leq n$, i.e., x and y are contained in no more than $n+1$ ticks of μ_1 (the maximal distance being n). Since $minsize(\mu_2, \overline{n}+1) \geq maxsize(\mu_1, n+1)$, we conclude that x and y are covered by no more than $\overline{n}+1$ ticks of μ_2, and therefore the distance between the ticks of μ_2 containing x and y cannot be greater than $\overline{n}$. This contradicts $\lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} = \overline{n}' > \overline{n}$.

Finally, suppose (c) holds. Let $\overline{m}' = \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2}$. Hence, $\overline{m}' < \overline{m}$. We first consider the case when $m \geq 0$. From the computation of $\overline{m}$ by the method we have $\overline{m}' < min(R) - 1$, where $R = \{r \mid maxsize(\mu_2, r) > mindist(\mu_1, m)\}$, and hence $\overline{m}' + 1 < min(R)$. It follows that $maxsize(\mu_2, \overline{m}' + 1) \leq mindist(\mu_1, m)$ since, otherwise, $\overline{m}'+1$ is in Q. Since the constraint $[m, n]\mu_1$ is satisfied by (x, y), we have $\lceil y \rceil^{\mu_1} - \lceil x \rceil^{\mu_1} \geq m$. Then, $mindist(\mu_1, m) \leq mindist(\mu_1, \lceil y \rceil^{\mu_1} - \lceil x \rceil^{\mu_1})$, and hence $maxsize(\mu_2, \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} + 1) \leq mindist(\mu_1, \lceil y \rceil^{\mu_1} - \lceil x \rceil^{\mu_1})$. By the definition of $maxsize()$ and $mindist()$ it is easy to show that $mindist(\nu_1, \lceil t_2 \rceil^{\nu_1} - \lceil t_1 \rceil^{\nu_1}) \leq t_2 - t_1 \leq maxsize(\nu_2, \lceil t_2 \rceil^{\nu_2} - \lceil t_1 \rceil^{\nu_2})$ for any values t_1, t_2 and temporal types μ and ν (provided that the necessary $\lceil\,\rceil$ functions are defined for those values). Applying these inequalities to our data we easily derive $y - x + 1 \leq maxsize(\mu_2, \lceil y \rceil^{\mu_2} - \lceil x \rceil^{\mu_2} + 1)$ and $y - x \geq mindist(\mu_1, \lceil y \rceil^{\mu_1} - \lceil x \rceil^{\mu_1})$. This results in $y - x + 1 \leq y - x$ which is a contradiction. When $m < 0$ the fact that $\overline{m}' < \overline{m}$ can be interpreted as the distance between x and y in terms of ticks of μ_2

being greater (in absolute value) than the bound given by the conversion method. The violation of the bound and the corresponding proof are, in this case, equivalent to point (b) above, since the values of x and y can be exchanged reversing the sign of the bounds. The conversion method, indeed, treats the negative lower bound as a positive upper bound reversing the sign of the result.

Proof of Theorem 6

Soundness. The algorithm is *sound* if any assignment satisfying the input network $\mathcal{CN} = (W, A, ,)$, also satisfies $\mathcal{CN}' = (W, A', , ')$ as returned by the algorithm. Hence, when the algorithm returns **inconsistent**, there must be no assignment satisfying $\mathcal{CN}$. Consider the case in which the algorithm returns a network $\mathcal{CN}'$: Soundness in this case is trivial, since (1) path consistency is known to be sound, (2) we proved that any output of conversion methods is implied by $\mathcal{CN}$, and (3) intersecting a TCG obtained from conversion with an original or previously derived TCG in the same granularity and for the same arc is a sound step. Consider now the case when the algorithm returns **inconsistent**. This implies $, _\nu(X, Y) = \{False\}$ for some arc (X, Y) and $\nu \in M$. This can be due to (1) the intersection of the ranges of two TCGs in the same granularity and for the same arc is empty, or (2) **undefined** is returned by an allowed conversion. The first case trivially implies that the input network is inconsistent since it implies that there is no assignment to variables X and Y in W satisfying two TCGs on (X, Y) which are either originally given or logically implied by $\mathcal{CN}$. For case (2) we need to consider a specific conversion method. Consider first the general method illustrated in Figure 5. **undefined** is returned by the method when either of the functions $mindist()$ or $maxdist()$ is undefined. $mindist(\mu_1, m, \mu_2)$ is undefined when $S = \emptyset$ where $S = \{\lceil t_2 \rceil^{\mu_2} - \lceil t_1 \rceil^{\mu_2} \mid \lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ are both defined, and $\lceil t_2 \rceil^{\mu_1} - \lceil t_1 \rceil^{\mu_1} \geq m\}$. Note that, since the conversion is *allowed*, the condition on the target type guarantees that if $\lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ are both defined then also $\lceil t_1 \rceil^{\mu_2}$ and $\lceil t_2 \rceil^{\mu_2}$ are defined. Then, $S = \emptyset$ if and only if there is no pair of instants t_1, t_2 such that (1)$\lceil t_1 \rceil^{\mu_X}$ and $\lceil t_2 \rceil^{\mu_Y}$ are both defined, and (2) $\lceil t_2 \rceil^{\mu_1} - \lceil t_1 \rceil^{\mu_1} \geq m$. This means that any pair (t_1, t_2) satisfying the input TCG $[m, n]\mu_1$ on arc (X, Y) is such that either $\lceil t_1 \rceil^{\mu_X}$ or $\lceil t_2 \rceil^{\mu_Y}$ is not defined. However, since μ_X and μ_Y are the *glb*s of the temporal types of TCGs involving respectively X and Y in $\mathcal{CN}$, any solution of $\mathcal{CN}$ must assign to X and Y instants covered respectively by μ_X and by μ_Y. Since the input TCG is either an original constraint or a logically implied one we conclude that if $S = \emptyset$ then $\mathcal{CN}$ is inconsistent. A similar argument applies for $maxdist()$. Finally, consider the alternative conversion method of Figure 6. According to this method, **undefined** is returned if and only if for all positive integers i, i', and i'' and each m' with $m \leq m' \leq n$ either $\mu_1(i) \cap \mu_X(i') = \emptyset$ or $\mu_1(i + m') \cap \mu_Y(i'') = \emptyset$. For any pair (t_1, t_2) satisfying the input TCG $[m, n]\mu_1$ on arc (X, Y) there exists $i = \lceil t_1 \rceil^{\mu_1}$ and $i + m' = \lceil t_2 \rceil^{\mu_1}$. However, if the above condition

holds, either there is no i' such that $t_1 \in \mu_X(i')$ or there is no i'' such that $t_2 \in \mu_Y(i'')$. Since μ_X and μ_Y are the *glb*s of the temporal types of TCGs involving respectively X and Y in $\mathcal{CN}$, any solution of $\mathcal{CN}$ must assign to X and Y instants covered respectively by μ_X and by μ_Y. We conclude that, since the input TCG is either an original constraint or a logically implied one, if the method returns **undefined** (i.e., the above condition holds) then $\mathcal{CN}$ is inconsistent.

Termination.

Step 1 (path consistency of an STP) is known to terminate. Step 2 trivially terminates. Step 3 is the critical part: we have to show that we cannot infinitely iterate between steps 1 and 2. Since we do not allow $+\infty$ nor $-\infty$ in the explicit TCGs and we discard any derived TCG containing $+/-\infty$, any explicit or implicit constraint between two variables will have only integer values. Consider $S = \sum_{i=1}^{k}(t_e^i - t_b^i)$ where k is the number of TCGs after the first iteration, and t_e^i and t_b^i are respectively the ending and beginning values in the range of TCG_i. It is easily seen that S is monotonically decreasing at each iteration, and, since it cannot be negative, this means that the algorithm terminates in a finite number of iterations.

Complexity.

Step (1) in the worst case takes time $O(n^3 \cdot |M|)$, where n is the number of variables (nodes in the graph) and $|M|$ is the number of temporal types appearing in the explicit constraints. Step (2) in the worst case takes time $O(c \cdot n^2 \cdot |M|)$, where c is the constant time required to translate a constraint from a granularity to an other one. Hence, their combination takes time $O(n^3 \cdot |M|)$. It is easily seen that both steps can only reduce the range of values in the constraints. In the worst case, at each iteration (steps 1 + 2), only one constraint range is reduced. If each constraint is reduced by only one unit in terms of its granularity the upper bound on the number of restrictions is the the maximum cardinality of the ranges in the TCGs (original and derived by conversion). Thus, the upper bound on the number of iterations is $n^2 \cdot |\mathcal{M}| \cdot w$, where w is the above upper bound on the number of restrictions for each constraint. We can conclude that the overall worst case complexity is $O(n^5 \cdot |M|^2 \cdot w)$.