Temporal Integrity Constraints with Indeterminacy

Wes Cowley
Department of Computer Science and Engineering
University of South Florida
wcowley@acm.org

Dimitris Plexousakis
Department of Computer Science
University of Crete
dp@csd.uch.gr

Abstract

Temporal integrity constraints specify the way in which a temporal database may be updated in order to maintain semantic integrity with respect to temporal and non-temporal data elements as these change over time. Temporal indeterminacy evolves from uncertainty in the measurement of time and from changes to or differences in the granularity of temporal elements under consideration. We introduce an algebra for indeterminate time intervals and define the semantics of potential satisfaction of temporal integrity constraints. We propose a temporal integrity constraint framework which supports temporal indeterminacy by employing a novel representation for indeterminate time intervals and discuss the optimization of integrity maintenance by the compilation and simplification of constraints.

1 Introduction

Database integrity constraints provide a mechanism for specifying rules in regards to which facts may legally be stored in the database [10]. They specify the legal database states as well as the allowable database state transitions. Although database management systems have traditionally provided support for a small class of mainly structural integrity constraints, there are great benefits stemming from the support of semantic constraints: in its absence, constraints must be implemented at the application level. This results in the integrity constraints being implemented in several application programs with the expected code consistency and maintenance issues. Centralizing integrity constraints at the database level ensures that all programs accessing the database will respect the same constraints. Furthermore, there are potential efficiency gains available when constraints reference data that would not otherwise be retrieved by a given program. Integrity constraints have been an ongoing area of research in the database community for many years.

Another long standing area of database research is temporal support. Typically databases maintain only the current state of the facts stored therein and provide little support for temporal data. The aim of temporal database research is to equip DBMSs with a higher level of support for temporal data, specifically the ability to maintain the history of updates to the database as well as to maintain the history of values which an attribute has had in the modeled domain [26, 17].

With temporal databases comes the need to maintain the integrity of the temporal data through temporal integrity constraints. This is a relatively recent area of research [24, 16, 21, 5, 22, 13]. Temporal integrity constraints must take into account not only the valid values of the temporal attribute themselves, but also the allowable ways in which all attributes, including non-temporal ones, may change over time. Therefore, temporal integrity constraints must potentially take into account the entire history of the database's evolution over time. The problem of constraint expressibility also becomes more complicated because of the need to involve temporal information in the constraint specification language.

Temporal indeterminacy is an inherent problem which arises in maintaining attribute values at different time periods in the modeled domain [11, 18, 14, 9, 4]. It may not always be clear at exactly which point in time an attribute's value changes. This indeterminacy may stem from inherent uncertainty in the modeled reality or from granularity changes. Hence, a recent area of research has focused on how to handle temporal indeterminacy in databases.

Given that there is a need to support both temporal integrity constraints and temporal indeterminacy in temporal databases, it is reasonable to assume that there is a need for temporal integrity constraints to take into account temporal indeterminacy. In this pa-

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

per, we discuss a novel method of representing, manipulating, and comparing indeterminate time intervals as well as what we believe is the first proposed method for incorporating indeterminacy into a temporal integrity constraint enforcement mechanism.

Providing support for temporal indeterminacy with temporal integrity constraints has a number of benefits. In general, more flexibility is allowed in how constraints are specified in the presence of temporal indeterminacy. Specific application areas where temporal indeterminacy and hence the concept of potential satisfaction of temporal integrity constraints are useful include planning, archaeology, astronomy, and genealogy. To the best of our knowledge this is the first work to account for indeterminacy within the context of temporal integrity maintenance.

The remainder of this paper is organized as follows. Section 2 reviews research in temporal integrity constraint specification and the problem of dealing with temporal indeterminacy in databases. Section 3 presents a representation of indeterminate temporal intervals using valid interval stamps, defines an algebra for manipulating valid interval stamps and characterizes the complexity of operations on intervals. That section also includes a brief discussion of the relationships between indeterminate intervals [7]. Section 4 introduces the notion of potential satisfaction of integrity constraints and the integration of temporal indeterminacy into a constraint maintenance framework. Finally, section 5 concludes the paper with a discussion of further research directions.

2 Related Work

The temporal integrity constraint method proposed here is extended from that previously presented for determinate time [22, 23, 21]. There have been several proposals for implementing temporal integrity constraint checking. The work begun by Gertz, Lipeck and Saake appears to be the earliest [24, 13]. They use active rules to monitor progress through a transition graph generated from a Future Temporal Logic (FTL) specification. Chomicki and Toman propose a method based on generating triggers for active RDBMSs from Past Temporal Logic (PTL) [5]. Sistla and Wolfson's method uses Condition/Action rules to implement an and/or graph for evaluating constraints specified in either FTL or PTL [25]. Gal, Etzion, and Segev propose an active database language which includes temporal integrity constraints [12]. Martín and Sistac show a method for deductive databases using SLDNF resolution [19]. Doucet, et al, base their approach on a bitemporal version database [8]. The method we describe here has been developed within the context of the Telos KBMS [20]. We believe it is the first to incorporate temporal indeterminacy.

Dyreson and Snodgrass have shown the most extensive results in temporal indeterminacy [9]. They describe a timestamp with a probability function for describing indeterminacy at the point and interval levels. Their method for representing and manipulating indeterminate timestamps is both more flexible and more complex than that described here. Anger, et al. handle indeterminacy by storing interval constraints between tuples in a temporal constraint network (TCN) [2]. Griffiths and Theodoulidis' method also stores interval relationships between tuples in a TCN [14] and uses Allen's constraint propagation algorithm to check local consistency and derive new relationships [1]. Koubarakis' work represents temporal indeterminacy by local and global constraints on variables representing the end points of the intervals [18]. Gadia, et al. propose a set oriented representation which supports both indeterminacy and incompleteness with a three-valued logic [11]. Their work is the closest to the indeterminate intervals described here, but provides a different set of operators. They do not describe a method for translating between interval constraint notation and their representation, while our work does not examine temporally missing values.

3 Valid Interval Stamps

3.1 Motivation

Indeterminate time may arise in a number of ways and is nearly unavoidable when dealing with the valid time of facts in a database. We are interested in associating a valid time period with tuples in order to support a temporal integrity constraint implementation [22, 23] in which constraints include combinations of the thirteen interval relationships [15, 1]. As such, it is appropriate that the valid time periods are also specified using interval relationships. Because most of these relationships do not precisely constrain the time periods involved, the resulting time stamps are indeterminate. Thus, we must be able to derive indeterminate valid time stamps from interval constraints, manipulate those indeterminate intervals, and translate the results back to interval constraints.

3.2 Points and Intervals

We assume a linear discrete time line [3] bounded by the range of the underlying integer type in which temporal points are represented. For our purposes, we assume that the underlying type is unbounded with respect to the range of the temporal domain being modeled. In practice, overflows will need to be addressed by allowing the granularity to be changed or by changing to a type with a larger range. We do not address either of these solutions in this paper. We further assume distinct elements ∞, $-\infty$, and `nil`. With the exception of the incomparable element `nil`, points are totally ordered by $<$. The relationships $<$, $=$, and $\leq$ between two points, p_1 and p_2, carry the expected meaning. More formally, we will assume a temporal structure $\mathcal{T}$ where $\mathcal{T}$ is characterized by a set of points $\mathcal{P}$, $\mathcal{P} = \{\dots, p_{-1}, p_0, p_1, \dots\}$, and a relation $<$.

The set $\mathcal{P}' = \mathcal{P} \cup \{-\infty, \infty, \texttt{nil}\}$ is isomorphic to the set $\mathcal{Z}' = \mathcal{Z} \cup \{-\infty, \infty, \texttt{nil}\}$. We define a one-to-one, onto, and invertible mapping function, $\iota : \mathcal{P}' \to \mathcal{Z}' : \mathcal{Z}' \to \mathcal{P}'$, as follows:

$$\iota(p) = \begin{cases} p & \text{if } p = -\infty, \infty, \text{or } \texttt{nil} \\ i & \text{for } p_i,\ i \in \mathcal{Z} \end{cases}$$

The relation $<$ between members of the set $\mathcal{P}$ is defined as $p_1 < p_2 \Leftrightarrow \iota(p_1) < \iota(p_2)$ and can be extended to the special elements $-\infty$ and ∞ by observing that $\forall p \in \mathcal{P}\ -\infty < p < \infty$. Finally, $<$ is undefined if either operand is `nil`. Similar translations give the meanings for $=$ and $\leq$. Note that from the definition of ι, $p_i = p_j \Rightarrow i = j$.

In several places in the sequel we will need to determine the relative order of a pair of points on the time line. The functions to do so are defined next.

Definition 1. The functions $\min_{\text{p}}$, $\max_{\text{p}}$, $\min_{\text{vp}}$, $\max_{\text{vp}} : \mathcal{P}' \times \mathcal{P}' \to \mathcal{P}'$ are defined as follows:

$$\min{}_{\text{p}}(p_1, p_2) = \begin{cases} \texttt{nil} & \text{if } p_1 = \texttt{nil} \vee p_2 = \texttt{nil} \\ p_1 & \text{if } p_1 < p_2 \\ p_2 & \text{otherwise} \end{cases}$$

$$\max{}_{\text{p}}(p_1, p_2) = \begin{cases} \texttt{nil} & \text{if } p_1 = \texttt{nil} \vee p_2 = \texttt{nil} \\ p_1 & \text{if } p_2 < p_1 \\ p_2 & \text{otherwise} \end{cases}$$

$$\min{}_{\text{vp}}(p_1, p_2) = \begin{cases} \texttt{nil} & \text{if } p_1 = \texttt{nil} \wedge p_2 = \texttt{nil} \\ p_1 & \text{if } p_2 = \texttt{nil} \vee p_1 < p_2 \\ p_2 & \text{otherwise} \end{cases}$$

$$\max{}_{\text{vp}}(p_1, p_2) = \begin{cases} \texttt{nil} & \text{if } p_1 = \texttt{nil} \wedge p_2 = \texttt{nil} \\ p_1 & \text{if } p_2 = \texttt{nil} \vee p_2 < p_1 \\ p_2 & \text{otherwise} \end{cases}$$

$\min_{\text{p}}$ ($\max_{\text{p}}$) chooses the earliest (latest) operand if both are not `nil`. $\min_{\text{vp}}$ and $\max_{\text{vp}}$ treat the case where exactly one of the operands is `nil` by returning the other point. All four operators are commutative, reflexive, and associative.

Given a point p, we need to refer to its next and previous points on the time line.

Definition 2. The functions previous and next, each with signature $\mathcal{P}' \to \mathcal{P}'$, are defined as follows:

$$\text{previous}(p) = \begin{cases} p & \text{if } p \in \{-\infty, \infty, \texttt{nil}\} \\ \iota^{-1}(\iota(p) - 1) & \text{otherwise} \end{cases}$$

$$\text{next}(p) = \begin{cases} p & \text{if } p \in \{-\infty, \infty, \texttt{nil}\} \\ \iota^{-1}(\iota(p) + 1) & \text{otherwise} \end{cases}$$

In order to discuss periods of time, as opposed to instants, we use intervals.

Definition 3. A *convex interval* is a set of consecutive points. An interval I can be represented by its lowest and highest points $\langle I_s, I_e \rangle$, $I_s \leq I_e$, $I_s = \texttt{nil} \Leftrightarrow I_e = \texttt{nil}$. We say that $p \in I$ iff $I_s \leq p \leq I_e$. The empty interval, which contains no point, is represented by $\langle \texttt{nil}, \texttt{nil} \rangle$ or $\emptyset$. We say that a convex interval is *infinite* if one or both endpoints is ∞ or $-\infty$. $\mathcal{I}$ refers to the set of all convex intervals. Points may be implicitly converted to intervals by the function *interval*: $\mathcal{P} \to \mathcal{I}$, defined as interval$(p) = \langle p, p \rangle$.

We will use the 13 basic interval relationships [15, 1] for comparing intervals for ordering and inclusion. The concept of intersection of convex intervals is important. This carries the same meaning as in set theory, namely that there is at least one point in common. Note that the empty interval, $\langle \texttt{nil}, \texttt{nil} \rangle$ does not intersect with any interval due to the incomparability of `nil`. The infinite interval, $\langle -\infty, \infty \rangle$ intersects with all non-empty intervals. We also need the concept of adjacency of convex intervals, which is defined next.

Definition 4. We say that two intervals I_1 and I_2 are *adjacent* if $\text{next}(I_{1_e}) = I_{2_s} \vee \text{next}(I_{2_e}) = I_{1_s}$.

We will also need the concept of non-convex intervals to talk about sets of non-consecutive points. These are defined simply as a set of convex intervals I_i which neither intersect nor are adjacent.

3.3 Valid Interval Stamps

A *valid interval stamp* (VIS) is an extension of the interval concept. A VIS describes the set of points which are definitely in the interval as well as those which *may* be in the interval. Through this we can represent the indeterminacy inherent in interval constraints.

Definition 5. A *convex valid interval stamp* (CVIS) is a 4-tuple $\langle I_s, D_s, D_e, I_e \rangle$ associated with a fact in a database. D_s, D_e, if not `nil`, are time stamps marking the end points of the determinate interval D; that period of time during which the associated fact is true. I_s and I_e, if not `nil`, represent the extended period of time before and after the determinate interval, respectively, during which the associated fact *may* be true. The indeterminate interval then is the non-convex interval $I = \{I_L, I_H\}$ where I_L and I_H are derived from the CVIS V as follows:

$$I_L = \begin{cases} \emptyset & \text{If } I_s = \texttt{nil} \\ \langle V.I_s, V.I_e \rangle & \text{If } D = \emptyset \\ \langle V.I_s, \text{previous}(V.D_s) \rangle & \text{otherwise} \end{cases}$$

$$I_H = \begin{cases} \emptyset & \text{If } I_e = \texttt{nil} \vee D = \emptyset \\ \langle \text{next}(V.D_e), V.I_e \rangle & \text{otherwise} \end{cases}$$

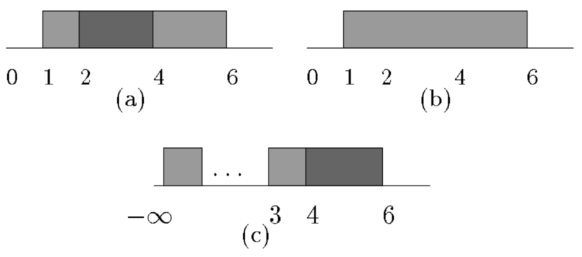

Figure 1: Three examples of CVISs

A CVIS for which both I_s and I_e are `nil` is called *fully determinate*, while one with both D_s and D_e being `nil` is called *fully indeterminate*. The empty VIS is represented as $\langle \texttt{nil}, \texttt{nil}, \texttt{nil}, \texttt{nil} \rangle$ or $\emptyset$. We will use $\mathcal{CV}$ to refer to the set of all CVISs. An interval I can be converted implicitly to a CVIS with the function cvis : $\mathcal{I} \rightarrow \mathcal{CV}$, defined as $\text{cvis}(I) = \langle \texttt{nil}, I_s, I_e, \texttt{nil} \rangle$.

Example 1. Figure 1a shows the CVIS $\langle 1, 2, 4, 6 \rangle$. This contains the points $\{2, 3, 4\}$ in the determinate region and $\{1, 5, 6\}$ in the indeterminate region. Figure 1b shows the same period of time covered by the fully indeterminate interval $\langle 1, \texttt{nil}, \texttt{nil}, 6 \rangle$. Finally, figure 1c shows the left infinite CVIS $\langle -\infty, 4, 6, \texttt{nil} \rangle$ which has no upper indeterminate region.

We will often need to obtain the earliest and latest points in a CVIS independently of whether that point is determinate or not.

Definition 6. We define the functions *upper* and *lower*: $\mathcal{CV} \rightarrow \mathcal{P}$ as $\text{upper}(V) = \max_{\text{vp}}(V.D_e, V.I_e)$ and $\text{lower}(V) = \min_{\text{vp}}(V.D_s, V.I_s)$.

We will also need to know the latest point at which a CVIS V can start and the earliest point at which it can end. To understand these functions, observe that if V has a determinate region then the interval will start no later than $V.D_s$ and end no earlier than $V.D_e$. On the other hand, if V is fully indeterminate then it may start as late as $V.I_e$ and end as earlier as $V.I_s$.

Definition 7. We define two functions, *maxlo* and *minup*: $\mathcal{CV} \rightarrow \mathcal{P}$ as $\text{maxlo}(V) = \min_{\text{vp}}(V.D_s, V.I_e)$ and $\text{minup}(V) = \max_{\text{vp}}(V.I_s, V.D_e)$.

The following extends the definition of membership in an interval. We will also use $p \in V.D$ to represent p's determinate inclusion in V and $p \in V.I$ to represent p's indeterminate inclusion in V.

Definition 8. We denote membership of a point p in a CVIS V as $p \in V$ and define that membership as $p \in V \Leftrightarrow \text{lower}(V) \leq p \leq \text{upper}(V)$.

Next we extend the concept of intersection.

Definition 9. Two CVISs, V_1 and V_2 *intersect* if: $\text{lower}(V_1) \leq \text{upper}(V_2) \wedge \text{lower}(V_2) \leq \text{upper}(V_1)$. We say V_1 and V_2 *determinately intersect* if $V_1.D \neq \emptyset \wedge V_2.D \neq \emptyset \wedge V_1.D$ *intersects* $V_2.D$.

The concept of adjacency can be extended to convex valid interval stamps as follows.

Definition 10. We say that two CVISs, V_1 and V_2 are *adjacent* if $\text{next}(\text{upper}(V_1)) = \text{lower}(V_2) \vee$ $\text{next}(\text{upper}(V_2)) = \text{lower}(V_1)$. We say that V_1 and V_2 are *determinately adjacent* if $V_1.D \neq \emptyset \wedge V_2.D \neq \emptyset \wedge V_1.D$ *adjacent* $V_2.D$. Note that two CVISs which are determinately adjacent may intersect but will not determinately intersect.

Details on how to translate from interval constraints of the form $\mathcal{R}\ I$, where $\mathcal{R}$ is an interval relationship and I is a determinate interval, to a convex valid interval stamp can be found in [6].

We will also need to consider non-convex valid interval stamps.

Definition 11. A *non-convex valid interval stamp* (NVIS) is a finite set of CVISs which meet two conditions. 1) No two CVISs may intersect. 2) If any two CVISs are adjacent, then both must have a definite interval and they must not be determinately adjacent. $\mathcal{NV}$ refers to the set of all NVISs. A CVIS can be implicitly converted to an NVIS with the function *nvis*: $\mathcal{CV} \rightarrow \mathcal{NV}$, defined as $\text{nvis}(C) = \{C\}$.

Definition 12. $\mathcal{V} = \mathcal{CV} \cup \mathcal{NV}$ denotes the set of VISs.

3.4 Operators on Valid Interval Stamps

There are a number of operators on VISs which correspond to the similarly named ones on Boolean expressions and sets. Specifically, we will define conjunction, intersection, union, and difference of VISs below. To avoid confusion, we will annotate the usual operators to emphasize that they carry different semantics than the familiar ones. For example, we use $\overset{v}{\cup}$ for the union of VISs. When operands are known to be convex (non-convex) VISs, we will use $\overset{cv}{\cup}$ ($\overset{nv}{\cup}$). When the familiar semantics from logic or set theory are sufficient, we will use the unannotated operators. The algorithms and correctness proofs are omitted for space reasons and are presented elsewhere [6].

3.4.1 Conjunction

There are times when a single interval constraint may not adequately express the knowledge one has about a fact's valid time. For that we must conjoin multiple constraints. We will do this by way of the VIS conjunction operator[1]: $\overset{v}{\wedge}$. The effect of $\overset{v}{\wedge}$ is to produce

[1]The term *conjunction* refers to the operation on interval constraints which $\overset{v}{\wedge}$ supports. It might be more appropriate to

a determinate region which includes the determinate region of both operands and an indeterminate region which includes only points in the indeterminate region of both operands. The operator uses the knowledge expressed by two VISs in order to increase the accuracy to which the valid time of a fact is known.

In order to compute the resulting VIS, V_r, from the conjunction of two existing CVISs, V_1 and V_2, we must first ensure that the conjunction is satisfiable. For example, there are no dates which satisfy the condition "contains Jan-98 and during Mar-98". On the other hand, "contains Jan-98 and after Nov-97" can be satisfied. Each operand restricts the range of the other.

Definition 13. Two CVISs, V_1 and V_2, are *conjunction compatible*, denoted $V_1 \wedge_{\text{comp}} V_2$, if 1) V_1 intersects V_2, 2) $V_2.D \neq \emptyset \Rightarrow (\text{lower}(V_1) <= V_2.D_s \wedge V_2.D_e <= \text{upper}(V_1))$, and 3) $V_1.D \neq \emptyset \Rightarrow (\text{lower}(V_2) <= V_1.D_s \wedge V_1.D_e <= \text{upper}(V_2))$. We extend this definition to NVISs by saying that two NVISs are *conjunction compatible* if $\forall V_{1_i} \in V_1 \exists V_{2_j} \in V_2$ such that $V_{1_i} \wedge_{\text{comp}} V_{2_j}$ and $\forall V_{2_j} \in V_2 \exists V_{1_i} \in V_1$ such that $V_{2_j} \wedge_{\text{comp}} V_{1_i}$. It should be clear from this definition that $\wedge_{\text{comp}}$ is symmetric and reflexive.

Conjunction compatibility ensures that the indefinite region of each interval encloses the definite region of the other. Otherwise, there would be a point which is in the definite region of one operand but not in the other operand at all. That is the condition which results from inconsistent interval constraints.

Definition 14 ($\overset{v}{\wedge}$). If V_1 and V_2 are VISs such that $V_1 \wedge_{\text{comp}} V_2$ we define their conjunction, denoted by $V_1 \overset{v}{\wedge} V_2$, as the VIS V_r such that $V_r.D = \{p | p \in V_1.D \vee p \in V_2.D\}$ and $V_r.I = \{p | p \in V_1.I \wedge p \in V_2.I\}$. If $\neg(V_1 \wedge_{\text{comp}} V_2)$ then $V_1 \overset{v}{\wedge} V_2 = \emptyset$.

Example 2. Suppose we wish to conjoin the three VISs shown in figure 2:

$$V_1 = \langle -\infty, 14, 21, \infty \rangle$$
$$V_2 = \langle -\infty, 17, 19, 23 \rangle$$
$$V_3 = \langle -\infty, 9, 12, \infty \rangle$$

We will arbitrarily start with $V_1 \overset{cv}{\wedge} V_2$. It's clear that they intersect and that each interval's determinate region falls within the indeterminate region of the other. Hence, they are conjunction compatible. Further, we can see that they determinately intersect. We compute the intermediate result to be $V_r = \langle -\infty, 14, 21, 23 \rangle$. The final valid interval is then $V_r' = V_r \overset{cv}{\wedge} V_3$. We note that V_r and V_3 are conjunction compatible but neither determinately intersect nor are determinately adjacent. The result then is:

$$V_r' = \{\langle -\infty, 9, 12, \texttt{nil} \rangle, \langle 13, 14, 21, 23 \rangle\}$$

think of the effect on a VIS as *strengthens.*

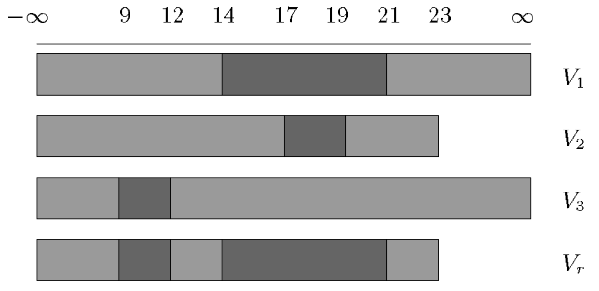

Figure 2: A Graphical Representation of Example 2

Because the order of the interval constraints in a valid time specification should not matter, it is important that the order of VISs with respect to $\overset{v}{\wedge}$ not matter. It can easily be shown that $(\mathcal{V}, \overset{v}{\wedge})$ forms a commutative monoid with identity $\langle -\infty, \texttt{nil}, \texttt{nil}, \infty \rangle$.

3.4.2 Intersection

The intersection of two CVISs is defined similarly to conjunction. The precondition is not as restrictive. It is simply necessary that the intervals intersect by definition 9. On the other hand, the result of intersection is more restrictive than that of conjunction. Specifically, the determinate region produced by conjunction includes the determinate regions of both operands. If a point is determinate in only one of the operands, it is determinate in the result. For intersection, the determinate region includes only those points determinately in *both* operands. If a point is determinately in one operand but indeterminately in the other, the intersection contains that point indeterminately.

Definition 15 ($\overset{v}{\cap}$). We define the intersection of two VISs, V_1 and V_2, denoted by $V_1 \overset{v}{\cap} V_2$, as the VIS V_r such that $V_r.D = \{p | p \in V_1.D \wedge p \in V_2.D\}$ and $V_r.I = \{p | p \in V_1 \wedge p \in V_2 \wedge (p \in V_1.I \vee p \in V_2.I)\}$.

It is straightforward to show that $(\mathcal{V}, \overset{v}{\cap})$ forms a commutative monoid with identity $\langle \texttt{nil}, -\infty, \infty, \texttt{nil} \rangle$.

3.4.3 Union

When an insert operation is performed for a tuple whose non-temporal attributes are the same as an existing tuple either the operation is treated as a new insert, resulting in a pair of tuples which are duplicated except in their valid interval stamps; or the valid interval stamp of the existing tuple is updated to include the valid time specified for the insert. We choose the latter semantics, provided by the union operator.

The union of two VISs is somewhat more complicated than conjunction and intersection. This is because the union of two NVISs may contain overlapping regions on the time line, which must be merged or split

Operator	Complexity
$\wedge_{\text{comp}}$	$O(\|V_1\|\|V_2\|)$
$\overset{nv}{\wedge}$	$O(\|V_1\|\|V_2\|)$
$\overset{nv}{\cap}$	$O(\|V_1\|\|V_2\|)$
$\overset{nv}{\cup}$	$O((\|V_1\|\|V_2\|)^2)$
$\overset{nv}{-}$	$O(\|V_2\|^4\|V_1\|)$

Table 1: Complexity of Operators over NVISs

in order to satisfy the definition. There are also no restrictions on the VISs which can be combined by $\overset{v}{\cup}$.

Definition 16 ($\overset{v}{\cup}$)**.** We define the union of two VISs, V_1 and V_2, denoted by $V_1 \overset{v}{\cup} V_2$, as the VIS V_r such that $V_r.D = \{p | p \in V_1.D \vee p \in V_2.D\}$ and $V_r.I = \{p | p \notin V_r.D \wedge (p \in V_1.I \vee p \in V_2.I)\}$.

Again, it is easy to show that $(\mathcal{V}, \overset{v}{\cup})$ forms a commutative monoid with identity $\langle nil, nil, nil, nil \rangle$.

3.4.4 Difference

When performing a deletion or update operation where the valid time specified is only a portion of the valid time for the affected tuples, we will need to produce the difference of two valid interval stamps. In general, this results in a non-convex interval. Observe that when deleting an interval V_d from another interval V, the resulting non-convex interval contains two portions of V. One is the section of V occurring before V_d, the other is the section occurring after V_d. If either V or V_d is indeterminate, then so is the resulting interval.

Definition 17 ($\overset{v}{-}$)**.** We define the difference of two VISs, V_1 and V_2, denoted by $V_1 \overset{v}{-} V_2$ as the VIS V_r such that $V_r.D = \{p | p \in V_1.D \wedge p \notin V_2\}$ and $V_r.I = \{p | (p \in V_1.D \wedge p \in V_2.I) \vee (p \in V_1.I \wedge p \notin V_2.D)\}$.

Note that $\overset{v}{-}$ is neither commutative nor associative. There is, however, an identity element: $\emptyset$.

3.5 Complexity of VIS Operators

The basic operators on CVISs: $\wedge_{\text{comp}}$, $\overset{cv}{\wedge}$, $\overset{cv}{\cap}$, $\overset{cv}{\cup}$, and $\overset{cv}{-}$, are all constant time. This should be clear from their definitions, which depend solely on the values of the end points and not the duration of the intervals. For an NVIS V, we will use $|V|$ to indicate the number of CVISs $V_i \in V$. Based on an analysis of the algorithms developed to implement the operators, we have the results in table 1 for NVISs. Proofs of these results can be found with the algorithms [6].

3.6 Indeterminate Intervals Relationships

Having defined a representation for indeterminate intervals, we need a way of comparing intervals for ordering and inclusion. We propose an extension of the thirteen relationships between determinate intervals. For each relationship between determinate intervals we have two corresponding relationships, potential and definite, between indeterminate intervals. The following principles give the basis for defining the definite and potential relationships. The specific definitions for each of the twenty-six relationships can be found in previous work [7].

Principle 1. For V_1 definitely $\mathcal{R}$ V_2 to hold, where $\mathcal{R}$ is an interval relationship, we must have: $\nexists V_1', V_2'$ such that $V_1 \wedge_{\text{comp}} V_1' \wedge V_2 \wedge_{\text{comp}} V_2' \wedge \neg((V_1 \overset{cv}{\wedge} V_1').D \,\mathcal{R}\, (V_2 \overset{cv}{\wedge} V_2').D)$. If this principle is met, it is not possible to conjoin any compatible interval constraint to either V_1 or V_2 such that $\mathcal{R}$ will not be satisfied.

Principle 2. For V_1 potentially $\mathcal{R}$ V_2 to hold, where $\mathcal{R}$ is an interval relationship, we must have: $\exists V_1', V_2'$ such that $V_1 \wedge_{\text{comp}} V_1' \wedge V_2 \wedge_{\text{comp}} V_2' \wedge (V_1 \overset{cv}{\wedge} V_1')$ definitely $\mathcal{R}$ $(V_2 \overset{cv}{\wedge} V_2')$. This ensures that it is possible, via the conjunction of additional interval constraints with V_1 and V_2 to arrive at a pair of intervals which definitely satisfy $\mathcal{R}$ by principle 1[2].

Previous work has shown some interesting observations about the compatibility of the various interval relationships [7]. First, we note that the interval relationships between determinate intervals are mutually exclusive. This observation extends to the definite relationships between valid interval stamps. We cannot make the same conclusion regarding potential relationships, however. It is easy to show that, for indeterminate interval V_1, V_2 and indeterminate relationships $\mathcal{R}_1, \mathcal{R}_2$, V_1 potentially $\mathcal{R}_1$ $V_2 \not\Rightarrow \neg(V_1$ potentially $\mathcal{R}_2$ $V_2)$. Finally, we can see from Principles 1 and 2 that, as expected, a definite relationship implies a potential one.

4 Potential Temporal Integrity Constraint Satisfaction

4.1 Constraint Satisfaction

We now examine what it means for a database to satisfy an integrity constraint in the presence of temporal indeterminacy. Definitions for constraint satisfaction have previously been given for the case of determinate intervals [21, 22, 23]. We extend those definitions to allow for temporal indeterminacy as supported by the VIS. In what follows, the notations $r(i_1, i_2)$ and $i_1 \, r \, i_2$ where r is an interval relationship and i_1, i_2 are valid interval stamps will be used interchangeably. First, we define the contents of the knowledge bases (databases) which underlie the integrity constraints.

[2]Note that either or both of V_1', V_2' may be the identity.

Definition 18. A *knowledge base*, KB, comprises a set of propositions, KB_P, defining the validity of predicates over possibly indeterminate time intervals, as well as a set, KB_R, of deductive rules and a set, KB_I of integrity constraints.

We will need the concepts of both object and temporal variable substitution.

Definition 19. An *object variable substitution* σ is a function mapping a variable x_i of sort S_i to an instance of the corresponding class C_i so that $instanceOf(\sigma(x_i), C_i, t) \in KB_P$ for some VIS t. A *temporal variable substitution* τ is a function mapping a temporal variable t of sort $Time$ to an interval in $\mathcal{V}$.

The next definition forms the core of the extension of the temporal integrity constraint method to indeterminate intervals. We show how to determine whether a knowledge base satisfies, either definitely or potentially, a temporal formula. In this definition and the following, we will require a shorthand notation for a common disjunction of interval relationships. We will use the term *covers* as the disjunction between *finishes*, *starts*, *during*, and *equals*, prefixing with *potentially* or *definitely* as appropriate. We will use *covered-by* as the disjunction of the inverses of those four relationships.

With the introduction of temporal indeterminacy, one may not be able to determine with certainty whether a knowledge base KB satisfies a formula or not. In the case where KB may or may not satisfy a formula because of temporal indeterminacy, we will use the symbol $\overset{p}{\models}$ for potential satisfaction. The traditional $\models$ will be used for definite satisfaction.

Definition 20. For base predicates P and Q, object substitution σ and temporal substitution τ:

- If P is ground and of the form $r(i_1, i_2)$ for a determinate interval relationship r and intervals i_1, i_2 then $(KB, \sigma, \tau) \models P$ iff $(KB, \sigma, \tau) \models$ definitely $r(i_1, i_2)$. Similarly, $(KB, \sigma, \tau) \overset{p}{\models} P$ iff $(KB, \sigma, \tau) \models$ potentially $r(i_1, i_2)$.
- If P is ground, then $(KB, \sigma, \tau) \models P$ and $(KB, \sigma, \tau) \overset{p}{\models} P$ iff $P \in KB_P$.
- $(KB, \sigma, \tau) \models P(x, t)$ iff $\exists t' \in \mathcal{V}$ such that $\tau(t)$ definitely covered-by $\tau(t') \wedge (KB, \sigma, \tau) \models P(\sigma(x), \tau(t'))$.
- $(KB, \sigma, \tau) \overset{p}{\models} P(x, t)$ iff $\exists t' \in \mathcal{V}$ such that $\tau(t)$ potentially covered-by $\tau(t') \wedge (KB, \sigma, \tau) \overset{p}{\models} P(\sigma(x), \tau(t'))$.
- $(KB, \sigma, \tau) \models \neg P(x, t)$ iff $\nexists t' \in \mathcal{V}$ such that $\tau(t)$ potentially covered-by $\tau(t') \wedge (KB, \sigma, \tau) \overset{p}{\models} P(\sigma(x), \tau(t'))$.
- $(KB, \sigma, \tau) \overset{p}{\models} \neg P(x, t)$ iff $\nexists t' \in \mathcal{V}$ such that $\tau(t)$ definitely covered-by $\tau(t') \wedge (KB, \sigma, \tau) \models P(\sigma(x), \tau(t'))$.
- $(KB, \sigma, \tau) \models P(x, t_1) \vee Q(x, t_2)$ iff $(KB, \sigma, \tau) \models P(x, t_1) \vee (KB, \sigma, \tau) \models Q(x, t_2)$. Similarly, $(KB, \sigma, \tau) \overset{p}{\models} P(x, t_1) \vee Q(x, t_2)$ iff $(KB, \sigma, \tau) \overset{p}{\models} P(x, t_1) \vee (KB, \sigma, \tau) \overset{p}{\models} Q(x, t_2)$.
- $(KB, \sigma, \tau) \models \forall x/C \; P(x, t)$ iff $(KB, \sigma[x/d], \tau) \models P(x, t)$ for all d such that $instanceOf(d, C, T)$ for some interval T, T definitely covered-by $\tau(t)$.
- $(KB, \sigma, \tau) \overset{p}{\models} \forall x/C \; P(x, t)$ iff $(KB, \sigma[x/d], \tau) \overset{p}{\models} P(x, t)$ for all d such that $instanceOf(d, C, T)$ for some interval T, T potentially covered-by $\tau(t)$.
- $(KB, \sigma, \tau) \models \forall t/Time \; P(x, t)$ iff $\forall T \in \mathcal{V} \, (KB, \sigma, \tau[t/T]) \models P(x, t)$.
- $(KB, \sigma, \tau) \overset{p}{\models} \forall t/Time \; P(x, t)$ iff $\forall T \in \mathcal{V} \, (KB, \sigma, \tau[t/T]) \overset{p}{\models} P(x, t)$.

If P is a derivable predicate defined by a set of deductive rules with bodies $R_1, \ldots, R_k$ and respective time intervals $T_1, \ldots, T_k$, $T_i \in \mathcal{V}$, then:

- $(KB, \sigma, \tau) \models P(x, t)$ iff $(KB, \sigma, \tau) \models \bigvee_{i=1}^{k} R_i \wedge$ (t definitely covered-by T), where $T = \overset{v}{\bigcap}{}_{i=1}^{k} T_i$.
- $(KB, \sigma, \tau) \overset{p}{\models} P(x, t)$ iff $(KB, \sigma, \tau) \overset{p}{\models} \bigvee_{i=1}^{k} R_i \wedge$ (t potentially covered-by T), where $T = \overset{v}{\bigcap}{}_{i=1}^{k} T_i$.

The intuition behind the cases for $(KB, \sigma, \tau) \models \neg P(x, t)$ and $(KB, \sigma, \tau) \overset{p}{\models} \neg P(x, t)$ is that if there is some time interval t' which potentially covers t during which KB potentially entails P then we cannot say that KB definitely entails P's negation. Likewise, we can say that KB may entail the negation of P during some interval t only if there is no time interval t' which definitely covers t during which P is definitely entailed by KB.

The satisfaction of temporal integrity constraints follows from definition 20.

Definition 21. If the temporal variables $t_1, \ldots, t_k$ occur in the constraint C with history time[3] T, then:

- $(KB, \sigma, \tau) \models C\,[\text{at } T]$ iff $(KB, \sigma, \tau) \models C'$, where $C' \equiv C \wedge \bigwedge_{i=1}^{k} (t_i$ definitely covered–by $T)$.

[3] In the context of Telos, history and belief time refer to the same concepts as valid and transaction time, respectively.

- $(KB,\sigma,\tau) \overset{p}{\models} C\,[\text{at } T]$ iff $(KB,\sigma,\tau) \overset{p}{\models} C'$, where $C' \equiv C \wedge \bigwedge_{i=1}^{k}(t_i \text{ potentially covered–by } T)$.

4.2 Compilation and Simplification

In this paragraph, we will discuss an extension of temporal integrity constraint compilation algorithm previously given in [21, 22, 23] to handle valid interval stamps. First, we define what is meant by an update and transaction in the context of the Telos language [20].

Definition 22. An *update* is an instantiated literal whose sign determines whether it is an insert or a deletion. A *transaction* is an arbitrary set of updates.

Next, we define how to determine updates possibly affecting the validity of a constraint.

Definition 23. Let t and T be VISs. An update $U(,\_,\_,t)$ is an *affecting update* for a constraint $C\,[\text{at } T]$ if and only if there exists a literal $L(\_,\_,\_)$ in C such that L unifies with the complement of U, and the indeterminate intersection $(\overset{v}{\cap})$, $t * T$, of intervals t and T is non-empty. A transaction $X = \{U_1, \ldots, U_m\}$ is called an *affecting transaction* for a constraint $C\,[\text{at } T]$ if and only if at least one of $U_1, \ldots U_m$ is an affecting update for the constraint.

The notion of dependence between constraints and deductive rules, defined next, does not depend on the temporal intervals involved.

Definition 24. A literal L *directly depends* on a literal K if and only if there exists a rule of the form $\forall x/C_1 \ldots \forall x_n/C_n\,(F \Rightarrow A)$ such that there exists a literal in F unifying with K with most general unifier θ and $A\theta = L$. A literal L *transitively depends* (or, simply, depends) on literal K if and only if it directly depends on K or depends on a literal M that directly depends on K.

The concerned class of a literal is used to narrow the set of constraints to be checked at runtime for each update. This is not strictly necessary for constraint checking, but is used as an optimization.

Definition 25. A *concerned class* for a literal L is a most specialized class CC such that inserting or deleting an instance of CC can affect the truth of L and the time intervals of L and CC are potentially unifiable[4]. The *concerned set* for a literal L is the set of distinct concerned classes for L.

Now we arrive at the compiled form of a constraint, the Parameterized Simplified Structure (PSS).

[4]By *potentially unifiable* we mean that there is some assignment which can be made to the intervals such that they are potentially equal.

Definition 26. Given a temporal constraint $C\,[\text{at}\,T]$ expressed in DNF and a literal L occurring positively (negatively) in C, the *parameterized simplified structure* of C with respect to L is a 6-tuple $(L, Params, CS, T,$
$T', SF)$ where $Params$ is the list of instantiation variables of L, CS is the concerned set of L, T and T' are the history and belief time intervals of the constraint respectively, and SF is the simplified form of the constraint that suffices to be evaluated when a deletion from (insertion to) L takes place. SF is derived by replacing instantiation variables with parameters, conjoining history time variables with potentially during(t_i, T), conjoining belief time variables with potentially during(t_i, T'), replacing L with *True* when the update is an insertion or *False* when the update is a delete, applying first order logic absorption rules, and applying temporal simplification rules.

Example 3. As an example of the application of the above definition to integrity constraints, consider the following dynamic constraint expressing the property that "salaries should never decrease".

$$\begin{array}{l}\forall p/\texttt{Employee}\ \forall s,s'/\texttt{Integer}\ \forall t_1,t_2/\texttt{Time} \\ \quad (salary(p,s,t_1) \wedge salary(p,s',t_2) \wedge \\ \quad \texttt{potentially before}(t_1,t_2) \\ \qquad \Rightarrow (s \leq s'))\ (\texttt{at 02/01/99..*})\end{array}$$

The constraint is expressed in the assertion language of Telos [20], a many-sorted first-order language with classes corresponding to sorts (e.g., the class `Employee`). `Time` is a built in class of time intervals. The example serves to convey the idea behind compilation and simplification of constraints.

Applying the simplification steps of definition 26 to this constraint will generate the following simplified form (capitalized variables denote parameters):

$$\begin{array}{l}\forall s/\texttt{Integer}\ \forall t_1/\texttt{TimeInterval} \\ \quad (salary(p,s,t_1)\ \wedge \\ \quad (t_1\ \texttt{potentially during 02/01/1999..*})\ \wedge \\ \quad (t_1\ \texttt{potentially before}\ T_2) \\ \qquad \Rightarrow (s \leq S'))\end{array}$$

4.3 Temporal Simplification

The last phase of the constraint compilation algorithms is temporal simplification. Consider a conjunction of the form potentially during$(t, i_1) \wedge r_1(t, i_2)$ where r_1 is one of the interval relationships or the negation of one of the relationships and i_1, i_2 are known intervals. Such a constraint often arises from the building of the PSS. Using the derived interval relationship, r_2, between i_1 and i_2 one can in some cases either show that the conjunction is unsatisfiable or find a simpler form of the conjunction, $r(t, i_3)$ where r and i_3 are derived from a simplification table. The simplification tables as well as details on their development have been omitted for space reasons. They can be found in [6].

Example 4. Suppose we have the temporal constraint:

$$C = \text{potentially during}(t, i_1) \wedge \text{potentially starts}(t, i_2)$$

where $i_1 = \langle 10, 20, 30, 40 \rangle$ and $i_2 = \langle 5, 20, 35, \texttt{nil} \rangle$. In [7], definition 20 gives the *potentially finishes* relationship as:

$$V_1.I_e = \texttt{nil} \wedge V_2.I_e = \texttt{nil} \wedge V_1.D_e = V_2.D_e \wedge \\ V_2.D_s < \text{lower}(V_1)$$

From this we can see that i_1 potentially finishes i_2. From C we see that r_1 is potentially starts. By consulting the simplification table we find the simplification rule:

```
Let $i_1' = \langle \texttt{nil}, \text{next}(\text{lower}(i_1)), \text{upper}(i_1), \texttt{nil} \rangle$
and $i_3 = i_1' \overset{cv}{\cap} i_2$.
If $\text{lower}(i_3) < \text{upper}(i_3)$ then
    $C' = \text{potentially starts}(t, i_3)$
otherwise inconsistent
```

and we proceed as follows:

$$i_1' = \langle \texttt{nil}, 11, 40, \texttt{nil} \rangle$$
$$i_3 = i_1' \overset{cv}{\wedge} i_2 = \langle \texttt{nil}, 11, 35, \texttt{nil} \rangle$$
$$C' = \text{potentially starts}(t, i_3)$$

4.4 Graph Construction and Run Time Evaluation

The remainder of the integrity constraint compilation algorithms, specifically the construction of the dependency graph, the computation of its transitive closure, and the incremental maintenance of both, do not require changes from the original definition [23] to handle indeterminate intervals. Graph construction proceeds by assigning each PSS to a vertex and drawing directed arcs from a vertex v_1 to a vertex v_2 when the integrity constraint or deductive rule associated with v_2 directly depends on the deductive rule associated with v_1. Definition 24 does not rely on the valid time interval of the rules.

The time intervals associated with the rules and the associated literals come into play during the run time evaluation of the dependency graph. The original evaluation algorithm [23] does not need to be changed to incorporate indeterminate intervals except for the choice between definite or potential constraint satisfaction semantics. The algorithm assumes that before each update all constraints are satisfied. When an update is made to the database the algorithm identifies each PSS in the graph associated with a literal which unifies with a literal involved in the update. All integrity constraints which are associated with these PSSs are potentially affected and must be evaluated using definition 21.

Complexity results and a performance analysis for the dependency graph maintenance and constraint evaluation phases have been previously produced for the determinate version of the algorithms [23]. Because the operations used in the extension to indeterminate intervals have a constant time complexity it is not expected that these results will be significantly different for the extension.

5 Conclusions and Future Directions

By using a novel representation for indeterminate intervals and an extension of the interval relationships to indeterminate time, a previously proposed method for temporal integrity constraint enforcement has been extended to accommodate temporal indeterminacy. Potential integrity constraint satisfaction allows a constraint designer more flexibility in specifying integrity constraints so as to delay violation until it is certain that no series of updates can be made which will cause the database to satisfy the constraints under definite constraint satisfaction semantics. We believe that for applications in which temporal indeterminacy is inherent this flexibility will prove important. To the best of our knowledge this is the first work to account for indeterminacy within the context of temporal integrity maintenance.

As far as complexity is concerned, the operations on valid interval stamps which are involved in the potential constraint satisfaction semantics are constant time, so should not increase the previously presented complexity results for determinate temporal integrity constraints.

There are several directions for further research. Foremost will be to produce an implementation of both the determinate version of the temporal integrity constraint algorithms, as well as the extension presented here. From a more theoretical standpoint, it would be interesting to examine whether the indeterminate interval relationships could be applied to an extension of Allen's constraint propogation algorithm [1] and to reformulate the indeterminate interval relationships without reference to the endpoints. The determinate interval relationships can be stated in terms of *before* or *meets*, for example. Finally, it is worth investigating how the complexity and expressiveness of potential constraint satisfaction would change if the indeterminate intervals had an associated probability function, similar to Dyreson and Snodgrass [9] and also to fuzzy logic based approach of Bouzid and Mouaddib [4].

References

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

[2] Frank D. Anger, Ramon A. Mata-Toledo, Robert A. Morris, and Rita V. Rodriguez. A re-

lational knowledge base with temporal reasoning. In *Proceedings of the Florida AI Research Symposium*, pages 147–151, 1988.

[3] Johan van Benthem. *The Logic of Time.* Kluwer Academic Publishers, Dordrecht, Holland, 1991.

[4] Maroua Bouzid and Abdel-Illah Mouaddib. Uncertain temporal reasoning for the distributed transportation scheduling problem. In *Proceedings of the 5th Int. Workshop on Temporal Representation and Reasoning*, pages 21–28, 1998.

[5] Jan Chomicki and David Toman. Implementing temporal integrity constraints using an active DBMS. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):566–582, 1995.

[6] Wes Cowley. Temporal integrity constraints with temporal indeterminacy. Master's thesis, University of South Florida, November 1999.

[7] Wes Cowley and Dimitris Plexousakis. An interval algebra for indeterminate time. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000).* To appear.

[8] Anne Doucet, Marie-Christine Fauvet, Stéphane Gançarski, Geneviève Jomier, and Sophie Monties. Using database versions to implement temporal integrity constraints. In *Proceedings of the Int. Workshop on Constraint Databases*, pages 219–233, 1997.

[9] Curtis E. Dyreson and Richard T. Snodgrass. Supporting valid-time indeterminacy. *ACM Transactions on Database Systems*, 23(1):1–57, 1998.

[10] J. Florentin. Consistency auditing of databases. *Computer Journal*, 17(1):52–58, 1974.

[11] Shashi K. Gadia, Sunil S. Nair, and Yiu-Cheong Poon. Incomplete information in relational temporal databases. In *Proceedings of the 18th Int. Conference on Very Large Databases*, pages 395–406, 1992.

[12] Avigdor Gal, Opher Etzion, and Arie Segev. A language for the support of constraints in temporal active databases. In *Proceedings ILPS'95–Workshop on Constraints, Databases and Logic Programming*, pages 42–58, 1995.

[13] M. Gertz and U.W. Lipeck. Deriving optimized integrity monitoring triggers from dynamic integrity constraints. *Data and Knowledge Engineering*, 20(2):163–194, 1996.

[14] Antony Griffiths and Babis Theodoulidis. SQL+i: Adding temporal indeterminacy to the database language SQL. In *Proceedings of the British National Conference on Databases*, pages 204–221, 1996.

[15] C. L. Hamblin. Instants and intervals. In J. T. Fraser, F. C. Haber, and G. H. Müller, editors, *The Study of Time*, pages 324–331, New York, NY, USA, 1972. Springer-Verlag.

[16] K. Hulsmann and G Saake. Theoretical foundations of handling large substitution sets in temporal integrity monitoring. *Acta Informatica*, 28(4):365–407, 1991.

[17] Christian S. Jensen, et al. A consensus glossary of temporal database concepts. *ACM SIGMOD Record*, 23(1):52–64, March 1994.

[18] Manolis Koubarakis. Database models for infinite and indefinite temporal information. *Information Systems*, 19(2):141–174, 1994.

[19] Carme Martín and Jaume Sistac. Applying transition rules to bitemporal deductive databases for integrity constraint checking. In *LID '96*, pages 111–128, 1996.

[20] John Mylopoulos, Vinay Chaudhri, Dimitris Plexousakis, Adel Shrufi, and Thodoros Topaloglou. Building knowledge base management systems. *VLDB Journal*, 5(4):238–263, 1996.

[21] Dimitris Plexousakis. Integrity constraint and rule maintenance in temporal deductive knowledge bases. In *Proceedings of the 19th Int. Conference on Very Large Databases*, pages 146–157, 1993.

[22] Dimitris Plexousakis. Compilation and simplification of temporal integrity constraints. In *Proceedings of the 2nd Int. Workshop on Rules in Database Systems*, pages 260–274, 1995.

[23] Dimitris Plexousakis. *On the Efficient Maintenance of Temporal Integrity in Knowledge Bases.* PhD thesis, University of Toronto, 1996.

[24] G. Saake and U. Lipeck. Foundations of temporal integrity monitoring. In C. Roland et al., editor, *Temporal Aspects in Information Systems*, pages 235–249. North Holland, 1988.

[25] A. Prasad Sistla and Ouri Wolfson. Temporal triggers in active databases. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):471–486, 1995.

[26] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings, Redwood City, Cal., 1993.