Temporal Query Languages: a Survey*

Jan Chomicki†

January 24, 1995

Category: Survey
Area: Temporal Databases

Abstract

We define formal notions of *temporal domain* and *temporal database*, and use them to survey a wide spectrum of temporal query languages. We distinguish between an abstract temporal database and its concrete representations, and accordingly between abstract and concrete temporal query languages. We also address the issue of incomplete temporal information.

1 Introduction

A *temporal database* is a repository of temporal information. A *temporal query language* is any query language for temporal databases. In this paper we propose a formal notion of temporal database and use this notion in surveying a wide spectrum of temporal query languages.

The need to store temporal information arises in many computer applications. Consider, for example, records of various kinds: financial [37], personnel, medical [98], or judicial. Also, monitoring data, e.g., in telecommunications network management [4] or process control, has often a temporal dimension.

There has been a lot of research in temporal databases in the past 15 years, as evidenced by a recent book [110]. We think that the field of temporal databases is mature enough to allow a unifying formal framework. Our proposed framework attempts to integrate the research on temporal databases with database theory and logic research. The framework is also relevant for the research on knowledge representation in the area of Artificial Intelligence. We hope that it will serve as a foundation for further, more systematic advances in

*An early version of this paper was presented as an invited tutorial at the *12th ACM Symposium on Principles of Database Systems*, May 1993, Washington, D.C. A later version appeared in *Proc. First International Conference on Temporal Logic*, July 1994, Bonn, Germany, Lecture Notes in Artificial Intelligence 827, Springer-Verlag.

†Address: Computing and Information Sciences, Kansas State University, Manhattan, KS 66506, USA, E-mail: chomicki@cis.ksu.edu.

the field of temporal databases. We have tried to use generally accepted database theory or logic terminology as much as possible and to limit the number of specialized temporal database terms to a minimum. For these reasons we have decided not to follow the recently proposed temporal database glossary [59].

The well-known ANSI/SPARC architecture [116] distinguishes three different levels in a database: physical, conceptual, and external. In the context of temporal databases, we propose to split the conceptual level even further and distinguish between an *abstract* temporal database and a *concrete* one. Intuitively, an abstract temporal database captures the formal, representation-independent meaning of a temporal database, while a concrete one provides a specific, finite representation for it in terms of a specific temporal data model. Accordingly, we study *abstract* query languages whose semantics is defined for abstract temporal databases and *concrete* query languages defined for specific concrete representations. Among the languages of the first group are: first-order logic, temporal logic, relational algebra, and a number of deductive languages; among those of the second – several languages described in [110] as well as a recently proposed temporal extension to SQL [108, 107]. We study the following issues in particular: formal semantics, expressiveness, query processing and its computational complexity, and representation-independence (for concrete query languages). We also address the implications for query languages of allowing *incomplete* information in temporal databases. We list many open problems.

The present survey is different from other existing surveys [12, 87, 88, 101, 112, 109], mainly because of the emphasis on a single unifying formal framework applicable to a wide spectrum of languages. [12] discusses the technical issues involved in representing infinite temporal databases, while the scope of [88, 109] is limited to concrete temporal query languages. [87, 101, 112] survey a number of different temporal formalisms and compare them with respect to a set of informally defined criteria. Our survey is also unique by including the discussion of recent work on constraint databases and constraint query languages [69]. A framework similar to ours has been independently proposed in [76] in the context of incomplete temporal information. However, the latter paper is not a survey and its scope is limited to constraint databases.

We do not claim that the present survey is exhaustive. So many different temporal data models and query languages have been proposed that it is not possible to describe them in detail in a single paper. However, we have attempted to extract the most important features of those models and languages, and present them in a single unified framework. We believe that this framework is also applicable to the approaches that are not explicitly covered here. For lack of space, we do not address several important issues for temporal query languages like query optimization techniques [82, 102] or physical storage structures [86, 74, 41]. We also limit our discussion to the relational model of data. A recent survey of query languages for temporal object-oriented databases appears in [105].

The plan of the paper is as follows. In section 2 we introduce a formal framework for temporal databases. In particular we specify the notions of *temporal domain*, *abstract* temporal database, and *concrete* temporal database. In section 3 we introduce a number of general properties of interest of temporal query languages. In section 4 we discuss *abstract* temporal query languages, and in section 5 a number of *concrete* temporal query languages. In section 6 we discuss incomplete temporal information, and in section 7 we summarize related work in Artificial Intelligence. In section 8 we draw conclusions.

2 Temporal databases

Within a formal framework for temporal databases the following issues need to be addressed:

- *choice of temporal domains:* points vs. intervals; linear vs. branching, dense vs. discrete, bounded vs. unbounded time;
- definitions of *abstract* and *concrete* temporal database;
- *query languages:* formal semantics, expressiveness, implementation.

In this section we address the first two issues. Temporal query languages are discussed in sections 3, 4, and 5.

2.1 Temporal domains

Temporal ontology

Here basically two options exist: *points* (instants) vs. *intervals*. Researchers in logic [118] and Artificial Intelligence [6] have been aware of this distinction for a long time. However, in temporal databases intervals are often confused with sets of points, so we will discuss this issue in several places in this paper. For now, the following natural language example should help to clarify the difference.

Example 2.1 Suppose that I have just made a car trip from Washington to New York. It lasted 5 hours, from 2 to 7pm. Saying "I was driving from Washington to New York from 2 to 7pm" I mean that I was driving from Washington to New York at *every time instant* in the interval $\langle 2, 7\rangle$. Here, the notion of an interval serves just as a description of a set of points. Clearly, it is also true that I was driving from Washington to New York during the interval $\langle 3, 5\rangle$ or any other subinterval of the interval $\langle 2, 7\rangle$. I can also say "I was driving from Washington to New York at 5pm". The situation is different when I say "I drove from Washington to New York from 2 to 7pm". Now, I mean only the interval $\langle 2, 7\rangle$ itself, and none of its subintervals nor any points belonging to it. In the first case, the view of time is *point-based* (intervals are just sets of points), in the second *interval-based.*

In the database context the point-based view is predominant and we concentrate on it here. We use the term *instant* for a time point. However, most of the AI research (for an up-to-date survey see [91]) takes the interval-based view. In the point-based view intervals are obtained as pairs of points. In the interval-based view it is common to have designators for interval endpoints. Thus usually moving between both views is easy in the first-order case (the propositional case is different but it does not concern us here).

Mathematical structure

As a rule some kind of mathematical structure is imposed on time instants. The most common is *order*: partial or total (linear). The orders studied in temporal databases are usually linear. However, exceptions exist: a linear transitive order that is not irreflexive or asymmetric can be used to model *cyclic time* and a partial order that satisfies *left-linearity* ("no branching to the left") – to model *branching time* [42]. (Linear-time cyclicity can also be handled using *ultimately periodic sets*: see the discussion later in this section.) In this survey we consider only linear orders, as they are prevalent in most temporal database applications.

We propose here to view a temporal domain as a *first-order structure* with some given *signature* (a set of constant, function, and relation symbols). Typical elements of signatures are:

- the binary order relation symbol "$<$",
- the constant symbol "0": to denote the origin or a standard reference point of a temporal domain,
- the unary *successor* function symbol "s": to capture the succession of time points,
- the binary function symbols "$+$"(plus) or "$-$"(minus): to capture relative distance of time points,
- the binary relation symbol "$\equiv_k$" for *congruence* modulo k: to capture *periodicity*.

The standard temporal domains in this context are: natural numbers $\mathbf{N} = (N, 0, <)$, integers $\mathbf{Z} = (Z, 0, <)$, rationals $\mathbf{Q} = (Q, 0, <)$, and reals $\mathbf{R} = (R, 0, <)$. Usually, we assume that equality is available in the temporal domain. (Unexpectedly, this is rather a strong assumption. It is violated, for example, by TSQL2, a temporal extension to SQL2 described in section 5.) Sometimes the temporal domains considered have universes that are finite or bounded subsets of the standard domains.

In most temporal database applications, it is commonly assumed that time is discrete and isomorphic to natural numbers [109]. In the AI view [5, 35], however, time is usually assumed to be dense. Moreover, continuous time has turned out to be extremely valuable in mathematics and physics, and is now the standard in the nascent area of *hybrid systems* [55]. The emerging consensus [40] is that many different temporal domains should be supported in a truly general temporal DBMS. The notion of a *constraint formula* [58, 69] makes it possible to finitely represent dense sets in computer storage.

Time is only one instance of an *interpreted* database domain. Other include space, physical quantities like temperature etc. In fact, the issues occurring in the representation of temporal information have their analogues in other application areas in which the storage of interpreted data is essential.

Time granularity

All of the temporal domains mentioned so far are "flat". To handle multiple time granularities, e.g., days vs. weeks, it is necessary to consider multiple interrelated temporal domains. An instant in a "higher-level" domain corresponds to a contiguous set of instants in another, "lower-level" domain. We do not consider multiple time granularities in this survey, as they have been treated recently in [129].

2.2 Abstract temporal databases

We propose here a formal notion of an *abstract* temporal database which captures the representation-independent meaning of a temporal database. An abstract temporal database may be viewed in several different but equivalent ways. The model-theoretic view, which is the most basic, treats an abstract temporal database as a first-order structure. The snapshot view treats it as a function that maps every instant to a set of tuples. Finally, the timestamp view treats it as a mapping associating a set of instants with every tuple.

For simplicity, we assume first a single temporal dimension, a single temporal domain $\mathcal{T}$, and a single data domain $\mathcal{U}$ (the latter domain contains standard database constants). We show then how to lift these restrictions. We will work in the context of the relational data model but the definition can be generalized to other data models formulated within a first-order framework, e.g., certain object-oriented models described by Beeri [15]. Moreover, we assume that the database schema is fixed and consists of a fixed set of relations.

Model-theoretic view

Consider a relation P of arity n. To model the fact that the tuples of P correspond to facts holding at some time instant, we introduce an *abstract temporal relation* P' of arity $n+1$ whose tuples have the following meaning:

$(a_1, \ldots, a_n, t) \in P'$ iff $P(a_1, \ldots, a_n)$ holds at instant t where $a_1, \ldots, a_n \in \mathcal{U}$.

More formally: it is well known that a relational database may be viewed as a finite structure $D = (\mathcal{U}, P_1, \ldots, P_k)$ for the first-order language L_D containing relation symbols for all the relations $P_1, \ldots, P_k$ in the database and constant symbols for all the elements of $\mathcal{U}$. A corresponding *abstract temporal database* (called a *temporal structure*) is a structure $D' = (\mathcal{U}, \mathcal{T}, P'_1, \ldots, P'_k)$ for the two-sorted first-order language L'_D containing a new temporal relation symbol for every abstract temporal relation P'_i and constant symbols for at least all the elements of $\mathcal{U}$ (and possibly some elements of $\mathcal{T}$ as well). The arity of P'_i is the arity of P_i plus 1. The extra argument of P'_i (last by convention) is called *temporal*, other arguments are called *data*. The language L'_D contains also the relation and function symbols, e.g., "$<$" or "$+$", from the language of the temporal domain $\mathcal{T}$ (those symbols are not interpreted in D' but have a fixed meaning). Using such a language for knowledge representation and reasoning was also proposed in the area of Artificial Intelligence [8].

Note that the concept of abstract temporal database is parameterized by the temporal domain $\mathcal{T}$ and no assumptions about $\mathcal{T}$ are made. We are going to say that a database D'

is *finite* if it consists of finite relations. Thus, the above definition allows finite and infinite databases. In fact, in some applications, e.g., dealing with infinite periodic data, it is more natural to consider infinite databases [12, 26, 64].

Snapshot view

In the snapshot view, an abstract temporal database (called a *snapshot* database) is a set of functions $\{f_{P_1}, \ldots, f_{P_k}\}$ such that for each instant $t \in \mathcal{T}$

$$f_{P_i}(t) = \{(a_1, \ldots, a_n) : P_i(a_1, \ldots, a_n) \text{ holds at } t\}.$$

A snapshot database may be thought of as consisting of *snapshot relations*, one for every relation symbol. Each such relation has arity two and is a non-1NF relation (the values of the second attribute are sets of tuples).

Timestamp view

In the timestamp view, an abstract temporal database (called a *timestamp* database) is a set of functions $\{g_{P_1}, \ldots, g_{P_k}\}$ such that

$$g_{P_i}(a_1, \ldots, a_n) = \{t : P_i(a_1, \ldots, a_n) \text{ holds at } t\}.$$

A timestamp database may be thought of as consisting of *timestamp relations*, one for every relation symbol. A timestamp relation has a data attribute for every data argument of the corresponding relation symbol, and a timestamp attribute. Each timestamp relation is also a non-1NF relation (the values of the timestamp attribute are sets of instants).

It should be clear now that all the three views presented above have the same expressive power. Below, we show in Table 1 a European history database viewed as a timestamp relation. In Table 2, we present a fragment of the corresponding snapshot relation and in Table 3 a relation that is a part of the corresponding temporal structure.

Multiple temporal dimensions

Multiple temporal dimensions are necessary to model *intervals* (as pairs of points) or multiple kinds of time, e.g., *valid time* (the time when a fact holds in the real world) and *transaction time* (the time when a fact is recorded in the database) [106]. For simplicity we assume a single temporal domain. To handle multiple temporal dimensions, the notion of temporal database needs to be appropriately generalized in an obvious way. In the model-theoretic view, we introduce the notion of *relation type* [96]: An abstract temporal relation with n data and k temporal attributes is of *type* $[n, k]$.

The different interpretations of multidimensional temporal databases can be captured by additional axioms serving as integrity constraints. For example, a constraint may state that the beginning of an interval always precedes its end. The snapshot and timestamp views can be modified along the same lines as the model-theoretic view.

There is no consensus on the number of temporal dimensions that need to be supported in a temporal database. In addition to valid and transaction times, [31] postulates *reference time* and [72] – *event time*. However, adding temporal dimensions results in considerably more complicated query languages that are also harder to implement, and has a negative influence on the computational complexity of query processing. We will illustrate these points later in the paper.

Properties of abstract temporal databases

We claim that any of the above definitions of an abstract temporal database provides a representation-independent meaning for any database defined within any of the extant temporal data models that are based on the relational model [110]. Any model-specific database (called hereafter a *concrete* temporal database) is then just a *representation* of an abstract temporal database. Two concrete temporal databases are equivalent if they represent the same abstract temporal database.

The notions of an abstract temporal relation and its associated type clarifies the notoriously confusing issue of whether a temporal data model is point- or interval-based. For instance, consider only valid-time temporal databases. Then a model is point-based if facts are associated with single time points, interval-based – if they are associated with intervals (represented as pairs of points). Thus in the first case the semantics of concrete temporal databases is expressed in terms of abstract temporal relations of type $[n, 1]$, while in the latter relations of type $[n, 2]$ are necessary.

Moreover, as we show later in this section the notion of abstract temporal database makes possible the study of integration and interoperability of different temporal data models within a single, precise framework. Finally, the semantics of many query languages and integrity constraints can be defined directly on abstract temporal databases, without any concern for the way they are represented. We discuss this topic in section 4. In particular, integrity constraints can be just first-order axioms if the model-theoretic view is adopted.

2.3 Concrete temporal databases

The notion of abstract temporal database is not sufficient to deal with temporal database applications. There is a fundamental reason for it: an abstract temporal database may be infinite, while only finite objects can be explicitly represented in computer storage. Moreover, many extant temporal database models provide already specific (and often incompatible) notions of temporal database. Therefore, it is necessary to consider *concrete* temporal databases that are specific finite representations of abstract temporal databases. For simplicity we discuss mostly the case of one temporal dimension.

There are two properties of classes of concrete temporal databases that are of primary interest: data expressiveness and succinctness. *Data expressiveness* of a temporal data model is the set of abstract temporal databases representable within it [14, 12]. Thus different temporal data models can be formally compared in terms of expressiveness. Data expressiveness should be distinguished from *query expressiveness* discussed in section 3. The second property is *succinctness*: how much space is necessary to represent a given abstract temporal relation. Again, this may serve as a basis for comparing different temporal data models.

Concrete timestamp databases

Out of the three different views of an abstract temporal database, the timestamp view leads to the most natural and useful notion of concrete database. Timestamps can be infinite sets but such sets can often be implicitly represented using *timestamp formulas:* first-order formulas with one free variable in the language of the temporal domain. Example: $0 < t < 5 \ \vee \ t > 10$. An example concrete timestamp database is shown in table 4.

We may ask what subsets of the temporal domain can serve as timestamps, i.e., can be defined by timestamp formulas. For example, if the temporal domain is $(N, <)$, then the timestamps are all (and only) finite or co-finite subsets of N. A more interesting case is that of Presburger arithmetic $(N, 0, +, <)$ where the timestamps are all ultimately-periodic subsets of N. For example, the set of natural numbers with period 7 beginning with 0 ("Sundays"), is described by the Presburger formula

$$\exists y.\, t = y + y + y + y + y + y + y$$

Another equivalent formulation is $t \equiv_7 0$ where "$\equiv_k$" means congruent modulo k. (This is an example of a *congruence formula.*) Consequently, if we allow Presburger or congruence formulas as timestamps, then infinite ultimately periodic abstract temporal databases can be finitely represented [64]. (An abstract temporal database D' is *ultimately periodic* if there is an instant t_0 and a natural number d such that for all $P_i, a_1, \ldots, a_n$ and $t > t_0$, if $P_i(a_1, \ldots, a_n, t)$ holds in D', then $P_i(a_1, \ldots, a_n, t + d)$ also holds in D'.)

Infinite periodic sets are useful because they can define *calendars*, e.g., the set of all business days or the set of all Sundays. Even finite periodic sets are better represented as infinite sets plus constraints guaranteeing finiteness. For example, consider the set of all Sundays in a given year. [94] describes adding a symbolic calendar level on top of a timestamp database.

The main technical tool in the theory of timestamp databases is *quantifier elimination.* (This notion is covered in many logic textbooks, e.g., [43].) A theory is said to *admit quantifier elimination* if for every formula in the language of this theory an equivalent quantifier-free formula can be effectively constructed. Additionally, we require that quantifier-free formulas thus obtained can be effectively tested for satisfaction. This additional requirement is especially important as we want to be able to effectively test whether a specific instant belongs to a timestamp defined by a timestamp formula. Fortunately, the theories of all the temporal domains that we proposed earlier in this section admit quantifier elimination.

Quantifier elimination may give different results for different temporal domains, even if they have the same signature.

Example 2.2 Consider the formula:

$$\exists t_1 .\, t < t_1 < 6.$$

In $(N, <)$, quantifier elimination yields the formula $t < 5$. In $(Q, <)$, quantifier elimination yields $t < 6$.

Quantifier elimination plays two distinct roles in timestamp databases. First, it makes possible limiting the attention to quantifier-free timestamp formulas. (However, sometimes quantifier elimination leads to an exponential blowup in the size of the formula and therefore is not performed [16].) Second, it is a basic tool of query evaluation (see section 4).

Constraints are atomic timestamp formulas. Consider a tuple w in a timestamp relation, whose quantifier-free timestamp formula is ϕ. The formula ϕ can be transformed into a disjunctive normal form $\phi_1 \vee \cdots \vee \phi_m$. Subsequently, negation can be eliminated from every disjunct because we are dealing with linear orders, giving $\phi' = \phi'_1 \vee \cdots \vee \phi'_m$. As a result every disjunct ϕ'_j, $1 \leq j \leq m$, becomes a conjunction of constraints. Now we can replace the tuple w by m tuples $w_1, \ldots, w_m$. Each new tuple w_j, $1 \leq j \leq m$, has data components identical to w and has ϕ'_j as the timestamp formula. Thus, disjunction and negation can be eliminated from timestamp formulas. A finite timestamp relation in this form is a *generalized relation* [68, 69, 100].

We call a timestamp formula $\phi(t)$ *separable* if $\phi(t)$ is a conjunction of the formulas of the form $t\theta c$ where $\theta \in \{=, <, >\}$ and $c \in \mathcal{T}$.

If we want to admit more than one time dimension, e.g., to represent intervals, then we need to have timestamps which are formulas with *two or more* free variables. Example: $t_1 = 0 \wedge t_2 = 5$. This should be distinguished from ranges of instants which are timestamps defined by single-variable formulas, e.g., $0 \leq t \wedge t \leq 5$. Multi-dimensional timestamp formulas can describe, for example, the set of all subintervals of an interval: $0 \leq t_1 < t_2 \leq 5$.

Most existing temporal data models [110] choose (a variant of) the timestamp representation. However, it is usually assumed that timestamps are finite (or at least bounded) sets. Using timestamp formulas to implicitly represent infinite sets makes it possible to skirt this limitation. It is an open question whether timestamps could be defined in a richer language than the first-order theory of the time domain. We should mention that in some approaches [52, 111] timestamps are associated not with tuples but with attribute values.

Finite temporal databases

For snapshot databases or temporal structures there is no natural notion of implicit finite representation. Therefore, if they are to be used as concrete databases, one has to enforce the restriction that they be finite and describe only a *finite* subset of the time domain. Also, as should be obvious from the examples above, both snapshot databases and temporal structures are quite wasteful in terms of space usage in typical applications.

Logic programs

An abstract temporal database, being an infinite structure, can often be finitely represented using a *logic program* consisting of a set of deductive (Horn) rules and a finite database. The abstract temporal database corresponding to such a program is its least Herbrand model [85]. For example, "Sundays" can be represented by the following program:

$$sunday(0).$$
$$sunday(s^7(T)) :- sunday(T).$$

[26] introduced this kind of representation for abstract temporal databases over the time domain $(N, 0, s)$ where "s" is the unary successor symbol. The syntax of logic programs is restricted by requiring that the successor symbol "s" is the only function symbol and can occur only in one argument of relations. The resulting language, which is an extension of *Datalog* (the language of function-free logic programs), is called $Datalog_{1S}$. ($Datalog_{1S}$ is also discussed as a query language in section 4.) An equally expressive logic programming language, Templog, whose syntax is based on temporal logic was proposed in [1, 9, 10, 11]. There is a simple syntactic translation between $Datalog_{1S}$ and Templog [12].

It is interesting to note that data expressiveness of $Datalog_{1S}$ is identical to that of the timestamp representation with Presburger timestamps: both express exactly ultimately periodic abstract temporal databases [12]. However, the representation using $Datalog_{1S}$ may be exponentially more succinct than the one using timestamps because in the former periodicity is implicit while in the latter it is explicit [27].

2.4 Interoperability

Our framework for temporal databases makes it possible to formulate the issue of temporal database interoperability in a very natural way. (This part represents ongoing work.)

Suppose that we have two temporal data models Δ_1 and Δ_2. Assume first that they use the same underlying temporal domains. The *meaning* of Δ_1 (resp. Δ_2) is defined as a *total* mapping h_1 (resp. h_2) from concrete temporal databases defined under Δ_1 (resp. Δ_2) to abstract temporal databases. The inverse mappings h_1^{-1} and h_2^{-1} may be *partial* because not necessarily every abstract temporal database is representable in the given data model (for example the model can be capable of representing only finite temporal databases). Also, the mappings may not be uniquely defined, which is not a problem because any inverse mapping is equally good. Assuming "$\circ$" denotes function composition, the database $d_1 = h_1^{-1} \circ h_2(d_2)$, when it is defined, represents the concrete database under Δ_1 corresponding to a concrete database d_2 under Δ_2. The database d_1 can then be queried using the query languages defined for Δ_1, providing thus the access to the database d_2. It should be pointed out that the corresponding concrete relations in both d_1 and d_2 represent abstract relations of the same type.

In this way interoperability is given a formal basis which also provides a general direction for the implementation. The data expressiveness of a data model sets exact limits on interoperability. E.g., if h_1^{-1} is not defined on $h_2(d_2)$, then there is no hope to use d_2 under Δ_1.

Notice that in the case when the semantics of a query language are defined directly on abstract temporal databases, there are no inverse mappings to deal with and the whole process is considerably simplified.

If the heterogenous databases have different underlying temporal domains, then additionally one has to construct a coercion between the domains. Such a coercion is often very natural, for example between $\mathbf{Z}$ and $\mathbf{Q}$.

3 Properties of query languages

The semantics of *abstract* query languages is defined with respect to abstract temporal databases, while that of *concrete* query languages – with respect to concrete temporal databases.

In the next two sections we survey the following properties of abstract and concrete temporal query languages:

- declarative semantics,
- closed-form evaluation,
- representation-independence,
- query expressiveness,
- data complexity of query evaluation, and
- efficient implementation.

We explain these notions below.

A semantics for a query language is *declarative* if it assigns a meaning to a query without referring to the way how the query is evaluated. Preferably, such semantics should be *logical* and provide a precise notion of a *model* of a query.

A query can be evaluated in *closed form* if the result of the query can be represented in the language of the database. This is trivially satisfied for languages over relational databases but for other kinds of databases, e.g., containing constraints, becomes a non-trivial property.

Representation-independence means that the answer to a query should be the same for every two concrete databases representing the same abstract temporal database (this property was identified in [52]). It should be clear that every abstract query language is representation-independent if it is correctly implemented. However, for concrete query languages representation independence needs to be separately proved.

Query expressiveness was defined in [18] as follows. Considering only yes-no queries, two queries are said to be *equivalent* if they return the same answer for every database. Now a query language L_1 is *at least as expressive* as another query language L_2 if for every query formulated in L_2, there is an equivalent query in L_1 (L_1 is *more expressive* than L_2 if additionally there is a query in L_1 for which there is no equivalent query in L_2). This notion of expressiveness is called *query expressiveness* [14] and is formally defined to be the class of sets of abstract temporal databases for which the queries in the language evaluate to true. This notion makes it possible to abstract away the features of a specific temporal data model and compare query languages from different data models. One has to be very careful, however, not to compare queries over concrete relations whose abstract equivalents are of different types, for example queries defined over relations representing interval data with those defined over relations representing point data. The notion of query expressiveness has been very important in the theory of query languages for relational databases [17, 3, 2].

Data complexity of query evaluation was defined in [19, 125] to mean the computational complexity of the set of finite databases for which a given, fixed query evaluates to true. One can also study *combined complexity* where the query is also a part of the input. However, data complexity seems to measure better the computational effort necessary for evaluating queries formulated in a given language. This notion has also been extensively used in database theory [17, 2].

4 Abstract query languages

4.1 Relational calculus

The first-order language L'_D of an abstract temporal database (assuming the model-theoretic view) can be used as a query language, as suggested in [64, 115]. It is commonly known as the *domain relational calculus*. Its semantics is the standard Tarskian semantics [43] which is obviously declarative. The answer to a first-order query is the set of valuations that make the query formula true in the given database.

Example 4.1 Consider the query *"list all countries that lost and regained independence"*. This query can be formulated in first-order logic as follows (I is a shorthand for *Independent*):

$$\exists t_1, t_2, t, s_1, s_2. \forall s.\ I(x, s_1, t_1) \wedge I(x, s_2, t_2) \wedge \neg I(x, s, t) \wedge t_1 < t < t_2.$$

Example 4.2 The functional dependency that *"an employee can have only one salary at a given time"* is simply expressed as:

$$\forall x, s, t, s', t'.\ Emp(x, s, t) \wedge Emp(x', s', t') \wedge x = x' \wedge t = t' \Rightarrow s = s'.$$

Another type of constraint may limit the range of the time instants that appear in a temporal relation. For example, suppose the constraint is *"a transaction can be done only on business days"* (i.e., Monday through Friday) and the time point 0 corresponds to a Sunday. This constraint can be expressed as:

$$\forall x, t.\ Transaction(x, t) \Rightarrow (t \not\equiv_7 0 \wedge t \not\equiv_7 6).$$

[62] uses first-order logic to formulate *a taxonomy* of temporal databases. The resulting formulas can be viewed as *constraint dependencies* [13] that generalize the traditional dependencies [116, 126, 67].

First-order logic can also be used as a concrete query language. This is straightforward for finite temporal structures. For finite snapshot relations (an example is in table 2), relational calculus requires second-order constructs for dealing with sets. Such a language was proposed in [129].

Example 4.3 Consider again the query *"list all countries that lost and regained independence"* from example 4.1. In the second-order relational calculus it can be formulated as follows (assume *sn* is the snapshot relation):

$$\exists t_1, t_2, t, s_1, s_2, X_1, X_2.\ sn(t_1, X_1) \wedge sn(t_2, X_2) \wedge t_1 < t < t_2 \wedge$$
$$(x, s_1) \in X_1 \wedge (x, s_2) \in X_2 \wedge (\forall X.\ sn(t, X) \Rightarrow \forall s.\ (x, s) \notin X).$$

Another proposal to use second-order calculus in temporal databases is [111]. However, the data model of that paper is not snapshot- but timestamp-based. Timestamps are associated not with tuples but with attribute values. In general, the issue of second-order calculus and the corresponding extensions of the relational algebra has been extensively researched in the context of data models with non-1NF relations or complex values [2]. As far as we know the results obtained in the course of this research have not been transferred to temporal databases.

For concrete timestamp databases, there are essentially two possibilities to implement query evaluation:

1. translate the query to relational algebra (for example using the algorithm of [122]) and use generalized versions of relational algebra operations described in the next subsection [76], or

2. directly evaluate the query in closed form (i.e., the result should also be a timestamp relation) using quantifier elimination procedures for the temporal domain and the data domain.

For the second approach to work, the theory of the data domain has to admit quantifier elimination. [69] proposed an elegant quantifier elimination algorithm for infinite domains with inequality constraints that would be applicable here (we do not assume any mathematical structure of the data domain). But still more work needs to be done in order to see whether quantifier elimination can be implemented with an efficiency approaching that of standard relational database operations.

We discuss now the issue of *data complexity* of first-order logic queries over concrete temporal databases with various temporal domains.

[69] analyzed the computational complexity of various *constraint query languages*. Their results apply to finite timestamp databases with timestamp formulas that are conjunctions of constraints (atomic formulas). Any fixed number of temporal dimensions over a single temporal domain $\mathcal{T}$ is allowed. They characterize the data complexity of processing of first-order logic queries as being in:

- LOGSPACE if $\mathcal{T}$ is a countably infinite dense linear order, e.g., $\mathcal{T} = \mathbf{Q}$,
- NC if $\mathcal{T}$ is real arithmetic, i.e., $\mathcal{T} = (R, 0, +, *, <)$.

[64] considered timestamp formulas that are constraints in Presburger arithmetic, i.e., $\mathcal{T} = (N, 0, +, <)$. They show that data complexity of first-order queries is in PTIME. Their results also apply to any fixed number of temporal dimensions.

4.2 Relational algebra

The semantics of relational algebra is defined set-theoretically for arbitrary, not necessarily finite, relations, so it fits well with the model-theoretic view of abstract temporal databases.

Under the snapshot view, relational algebra operations can only be used *pointwise*, i.e., on the snapshots corresponding to the *same* time instants in different relations. One needs to consider also the generalized versions of relational algebra operations that simultaneously apply to *all* instants. (This is analogous to the *filter* or *apply_to_all* operation proposed first in the area of functional programming and applied to databases with complex values in [15].) Also, additional operations that relate different instants are necessary. Such an approach has been pursued in [95] in the context of the temporal domain **N**. This proposal does not address any implementation issues, so snapshot relations are not required to be finite.

Under the timestamp view, relational algebra operations need to be generalized to deal with timestamp relations [68, 76, 96]. For example, the natural join $R(A,T) \bowtie S(B,T)$ is defined as

$$R(A,T) \bowtie S(B,T) = \{(a, b, \phi_1 \wedge \phi_2) | (a, \phi_1) \in R, (b, \phi_2) \in S\}$$

where ϕ_1 and ϕ_2 are timestamp formulas. Projecting out a temporal attribute out is implemented by quantifier elimination.

As in the standard relational algebra, one can consider constant timestamp relations. While in the former case they are finite, in the latter even if they are finite they can represent infinite abstract temporal relations. For example, one can define a singleton unary constant relation

$$Time = \{(t = t)\}.$$

consisting of all time instants.

Example 4.4 The query *"list all countries that lost and regained independence"* can be formulated in relational algebra (with renaming) as:

$$\pi_X(P - \pi_{X,T}(I(X, S, T))).$$

where

$$P = \pi_{X,T}(\sigma_{T_1<T<T_2}(I(X,S_1,T_1) \bowtie I(X,S_2,T_2) \bowtie Time(T))).$$

PTIME-evaluable algebras were proposed by [64] for the temporal domain $(N, 0, +, <)$ and by [68] for $(Q, <)$. Developing efficient implementation methods and query optimization techniques that appropriately generalize those developed for relational databases is very much an open problem. Some recent work in constraint databases [70, 16] may be applicable here. Also, the possibility of generalizing the methods of query processing developed for finite temporal databases [82, 102]) should be explored.

4.3 Temporal logic

Here we consider first abstract temporal relations of type $[n, 1]$. Instead of the first-order language L'_D, we can use a temporal extension of L_D, denoted by $tl(L_D)$. This language contains binary temporal connectives **since** and **until** with the following meaning:

- A **since** B is true at instant i iff for some j, $j < i$, B is true at instant j, and for every k, $j < k \leq i$, A is true at instant k.
- A **until** B is true at instant i iff for some j, $j > i$, B is true at instant j and for every k, $i \leq k < j$, A is true at instant k.

We will use "$\blacklozenge A$" (*sometime in the past* A) as a shorthand for "*true* **since** A" and "$\lozenge A$" (*sometime in the future* A) as a shorthand for "*true* **until** A". A comprehensive recent reference for temporal logic is [50].

Example 4.5 The query *"list all countries that lost and regained independence"* from example 4.1 can be formulated in temporal logic as follows:

$$\exists s_1, s_2.\ \lozenge(\blacklozenge I(x, s_1) \land \lozenge I(x, s_2) \land \forall s.\ \neg I(x, s)).$$

Temporal logic is of interest because of its ubiquity in different areas of computer science. In databases temporal logic has been used as a language for querying *finite* snapshot databases [47, 48, 115], and for formulating temporal integrity constraints [84, 24, 103].

One possible implementation method for temporal logic queries over timestamp databases is to translate them to first-order logic queries and use the implementation for first-order logic queries described earlier. It may also be possible to develop query evaluation and optimization methods that are specific to temporal logic.

We believe that the central research issue here is *query expressiveness* of temporal logic. Is $tl(L_D)$ equally expressive as L'_D (assuming the signature of the temporal domain $\mathcal{T}$ contains only the order relation symbol $<$)? This seems to be one of the most interesting open research problems in the area of temporal databases. There are some relevant early results due mainly to H. Kamp. He showed that propositional temporal logic is equally expressive as monadic first-order logic [66] over Dedekind-complete temporal domains (e.g., **N**, **Z**, and **R**). Later, he showed that $tl(L_D)$ is strictly less expressive than $L_D^{\mathcal{T}}$ for arbitrary temporal databases [65]. His construction, however, used in an essential way infinite temporal databases and was applicable only to dense temporal domains. We conjecture that the separation between first-order and temporal logics holds even for finite databases and discrete domains.

Another important result is that of [49] who proved that over **N** the connective **until** suffices in the propositional case; **since** can be eliminated. However, their proof uses Kamp's result [66], so the same question for first-order temporal logic remains open.

If abstract temporal relations are of type $[n, k]$ for $k > 1$, the above **since/until** temporal logic is no longer applicable. The choice of a more powerful temporal logic is determined by the semantics of the temporal attributes. If they are meant to represent intervals, then some kind of *interval* temporal logic is called for [118, 56, 127]. If they are meant to represent multiple kinds of time, then some form of *multi-dimensional* temporal logic should be used [45, 92]).

4.4 Inductive query languages

The temporal query languages discussed so far are all *first-order*. However, there are many natural queries that are not first-order but inductive or even second-order.

Example 4.6 The following is an example of an inductive temporal query that may very well be posed by an epidemiologist:

> Find all the persons at risk where "being at risk" is defined in the following way: a person is *at risk* at a given time if she has been earlier infected or she has been in contact with someone already at risk.

Notice that having been in contact with someone who only became at risk later should not result in classifying the person as at risk. It should be clear that the above query is not first-order because of the inherent recursion.

Inductive temporal queries can be formulated in a number of *logic programming languages*. Those languages extend *Datalog*, the language of function-free logic programs, in various ways. We consider the following:

- *Datalog*$^{<_Z}$ [99, 100]: *Datalog* with integer order constraints,
- *Datalog*$^{<_Q}$ [69, 68]: *Datalog* with rational order constraints,
- *Datalog*$_{1S}$ [26, 23, 27, 25]: *Datalog* with a unary successor symbol in one argument,
- *Datalog* with a unary successor symbol and linear arithmetic constraints [14].

Strictly speaking, the first two languages were proposed for temporal relations with an arbitrary number of temporal attributes and no data attributes, i.e., temporal relations of type $[0, k]$, $k > 0$. But the generalization to arbitrary temporal relations of type $[n, k]$, $n \geq 0$, $k \geq 0$, is straightforward. On the other hand, $Datalog_{1S}$ allows only temporal relations of type $[n, 1]$, $n \geq 0$. The language of [14] allows temporal relations of arbitrary type. Moreover, $Datalog_{1S}$ is applicable only to finite temporal structures, while the remaining languages are applicable to concrete timestamp databases.

Example 4.7 The formulation in $Datalog^{<_Z}$ (or $Datalog^{<_Q}$) of the query from example 4.6 is very natural:

$$\begin{array}{l} atRisk(X, T) \leftarrow infected(X, T'), T' < T. \\ atRisk(X, T) \leftarrow contacted(X, Y, T'), atRisk(Y, T'), T' < T. \end{array}$$

The rules in this example are not range-restricted. This is usually the case in constraint languages and it does not lead to difficulties: the evaluation mechanism for such languages is more general than standard bottom-up evaluation [116]. The formulation in $Datalog_{1S}$ is less transparent because the order $<$ relation symbol is not directly available:

$$\begin{array}{l} atRisk(X, T+1) \leftarrow infected(X, T). \\ atRisk(X, T+1) \leftarrow atRisk(X, T). \\ atRisk(X, T+1) \leftarrow contacted(X, Y, T), atRisk(Y, T). \end{array}$$

Because of the syntactic restrictions on the number of arguments in which the successor function symbol may appear, $Datalog_{1S}$ cannot express some $Datalog^{<_Z}$ queries.

The above deductive languages can be extended with stratified negation [7] in clause bodies. Such an extension of $Datalog_{1S}$ will be denoted *Stratified* $Datalog_{1S}\neg$. The addition of negation makes the deductive languages capable of expressing also all first-order logic queries over abstract temporal databases of type $[n, 1]$.

Example 4.8 The query from example 4.1 can be expressed in $Datalog^{<_Z}$ with stratified negation as:

$$\begin{array}{l} someI(X, T) \leftarrow I(X, S, T). \\ query(X) \leftarrow I(X, S_1, T_1), I(X, S_2, T_2), \neg someI(X, T), T_1 < T < T_2. \end{array}$$

The semantics of logic programs without negation is given by their least Herbrand models [121]. In the presence of stratified negation, the semantics is given by perfect models [97]. $Datalog_{1S}$ is a subset of Prolog, so its implementation requires only an adaptation of existing logic programming implementation techniques [25]. On the other hand, the implementation of constraint languages like $Datalog^{<_Z}$ and $Datalog^{<_Q}$ requires a quantifier elimination procedure. For such languages, the proofs of closed-form evaluation or termination of bottom-up computation are often quite involved [69, 99, 100].

We discuss now the issue of the *data complexity* of the deductive languages. [69, 99, 100] obtained closed-form evaluation and PTIME computability for $Datalog^{<_Q}$ and $Datalog^{<_Z}$. They also considered the extension of $Datalog^{<_Q}$ with negation but under a different, *inflationary* semantics [73], which also has closed-form evaluation and PTIME computability. Extending $Datalog^{<_Z}$ with stratified (or inflationary) negation is problematic, as one can define then a relation coding the successor function symbol and obtain all Turing-computable functions [99, 100]. It is, however, possible to extend $Datalog^{<_Z}$ with congruence constraints while preserving closed-form evaluation and PTIME computability [114]. Data complexity of $Datalog_{1S}$ (*Stratified* $Datalog_{1S}\neg$) queries on finite temporal structures is PSPACE-complete [26, 22] but there are subsets of $Datalog_{1S}$ for which queries can be evaluated in PTIME [23]. *Datalog* with a unary successor symbol and linear arithmetic constraints [14] is a very expressive language and termination of query evaluation can not be guaranteed.

Another way to obtain inductive queries is to extend logic query languages with *fixpoint operators*. In fixpoint query languages, queries are defined as least (or greatest) solutions of equations $X = \phi(X)$ where ϕ is a logic formula and X is a relation or a set. Two best known fixpoint query languages are: least fixpoint queries on finite relational databases [18] and temporal fixpoint calculus (*propositional* temporal logic extended with fixpoint operators) [124, 47]. Both languages add extra expressive power to the underlying logic languages. For example, the first language makes possible the expression of transitive closure queries which are not expressible in first-order logic. The second language is capable of defining the property EVEN true of every *even-numbered* state (in the case of $\mathcal{T} = (N, <)$ a temporal database may be viewed as a sequence of states), which is not expressible in propositional temporal logic [130]. In the context of temporal databases, it is natural to consider the fixpoint extensions of both first-order logic and temporal logic. This is a topic of current and future research.

We have conjectured earlier that temporal logic is strictly less expressive than first-order logic. Does a similar separation hold for the fixpoint extensions of both languages? Moreover, it is worthwhile to study the relationship between deductive, fixpoint and *second-order* languages. [11] showed that propositional temporal fixpoint calculus and Monadic Stratified $Datalog_{1S}\neg$ are equally expressive and that there is a similar correspondence between the positive fixpoint calculus and Monadic $Datalog_{1S}$. Propositional temporal fixpoint calculus and *second-order* monadic logic (with quantification over sets of time points) are known to be equally expressive [113].

[115, 48] have proposed to extend relational algebra with recursively-defined operators. These operators are essentially limited propositional temporal fixpoint operators. They are, however, restricted to finite snapshot databases (or finite temporal structures). Moreover, the algebra in [115] does not even achieve the full power of propositional fixpoints, as it is unable to express EVEN. This version of relational algebra is equally expressive as the **since/until** temporal logic discussed earlier. [48] additionally proposed to introduce a special *time* relation that keeps track of the flow of time. Using this relation and arithmetic selection conditions, EVEN can be expressed. The exact expressive power and the data complexity of the resulting query language have not been studied, however.

5 Concrete Query Languages

In this section we discuss a number of temporal query languages whose semantics is defined with respect to concrete temporal databases. We then summarize the main limitations of those languages.

5.1 TQuel

TQuel, proposed in [104], is a well-known temporal query language derived from Quel. It supports a single temporal domain which is discrete, infinite and multi-level (there is a way to refer to a specific day, hour etc.), and two temporal dimensions: valid time and transaction time. For simplicity, however, we will only consider valid time with a single granularity.

The data model of TQuel is a variant of the timestamp representation. Specifically, a timestamp is an interval $\langle a, b\rangle$ where a is the start and b is the end of the interval. Associating the interval $\langle a, b\rangle$ with a fact $p(\bar{x})$ (or, equivalently with a tuple $\bar{x}$ in the relation corresponding to p) means that $p(\bar{x})$ holds for every t, $a \leq t \leq b$. If $b = \infty$, then $\langle a, b\rangle = \{t : a \leq t\}$.

The intervals are required to be maximal, so the timestamps of identical facts are coalesced if they denote overlapping intervals. This requirement imposes quite a burden on database *update* procedures (an insertion may trigger a coalescing operation). On the other hand, a timestamp representation in which maximality is not enforced may be arbitrarily less succinct than the one used in TQuel. The intervals are represented in TQuel using two additional attributes `From` and `To` in every relation.

It is important to see that the data model of TQuel is point-based, not interval-based. Intervals serve only as a representational device. The truth values of facts are associated with points, not intervals. For example, it is impossible to represent in the database the situation where a fact is true of an interval but not of some of its subintervals. Formally, a TQuel relation (with valid time only) represents an abstract temporal relation of type $[n, 1]$. TQuel relations are finite. The data expressiveness of TQuel is the same as that of timestamp relations with separable timestamp formulas (defined in section 2).

The semantics of TQuel queries is given indirectly by a translation to Quel. TQuel presently has no declarative, logical semantics, but such semantics is in principle possible.

Example 5.1 Assuming every country had only one capital, the query *"list all countries that lost and regained independence"* from example 4.1 can be written in TQuel as follows:

```
range of O1 is IndependentCountries
range of O2 is IndependentCountries
retrieve (O1.Name)
valid at begin of O1
where O1.Name=O2.Name
when (end of O1) precede (begin of O2)
```

The negation is the example above is not necessary as it is assumed that the timestamps represent maximal intervals that can not be coalesced. In fact, the language does not contain negation. Also, the temporal attributes attributes `From` and `To` are treated in a special way: they are not directly referenced in queries. Instead each pair of values for those attributes is viewed as an interval which can be compared to other intervals using a predefined set of interval relationships (like `precede` in the example above).

The query expressiveness of TQuel depends on the repertoire of interval relationships. With a sufficiently rich set of those relationships [5] it can be shown that TQuel queries can simulate temporal logic queries. To show that temporal logic can simulate TQuel, it is enough to express interval relationships in temporal logic which is not difficult. However, to obtain the power of full-fledged logical negation TQuel (like Quel [116]) uses sequences of queries. The example query above is not correct if any country has multiple capitals, as each capital gives rise to a different, disjoint interval. So in such a case a sequence of TQuel queries with intermediate relations is necessary.

It should be pointed out that the existence of a translation from TQuel to Quel which can in turn be translated to relational calculus does not prove by itself that TQuel is subsumed by relational calculus, as claimed by [30]. That is so because the translation yields relational calculus queries over relations of type $[n, 2]$, while the point-based semantics of TQuel require that it be compared with the calculus over relations of type $[n, 1]$. One has to show that the relations of type $[n, 2]$ obtained in this translation are in fact first-order definable from relations of type $[n, 1]$ (which fortunately is the case because interval endpoints are first-order definable).

TQuel is essentially first-order so it is incapable of expressing inductive temporal queries. The data complexity of TQuel queries has not been analyzed but is clearly polynomial. Due to the maximality requirement on timestamps, a given abstract temporal relation is uniquely represented as a TQuel relation. This guarantees representation independence of TQuel queries.

The syntax and semantics of TQuel become quite cumbersome when transaction time is also considered. There is no support for more than two temporal dimensions.

5.2 TSQL2

TSQL2 [108, 107] is a proposed extension to SQL2 [46]. The proposal refrains from making a commitment to a specific temporal domain (except that it is linearly ordered) and consequently does not allow testing for equality of time instants. Such an equality could lead to queries that give different answers for discrete and dense temporal domains. Instead of equality a family of relation symbols $\approx_g$ is provided where g is is any granularity. The meaning of $x \approx_g y$ is: "x and y happened in the same granule of a given granularity g". Valid and transaction time as well as multiple time granularities are supported.

Like TQuel TSQL2 is point-based, not interval-based. TSQL2 relations are very much like TQuel relations except that facts are timestamped not with (maximal) intervals but with *finite unions* of maximal intervals. Every fact has exactly one timestamp. The data expressiveness of TSQL2 is identical to that of TQuel (ignoring multiple time granularities).

We are not aware of any comprehensive description, even informal, of the semantics of TSQL2 queries. This makes it impossible to establish formal properties of this language. However, the description of its syntax [108] leaves some doubts whether the TSQL2 query language is well matched with the data model. In particular, TSQL2 designers still use the set of interval relationships of [5]. For unions of intervals this set is not sufficient and should be augmented, e.g., along the lines of [80, 81, 83].

5.3 HRDM

HRDM (Historical Relational Data Model) [28, 29], based on some earlier foundational work [33, 32], is one of the most influential temporal data models. A similar model was proposed in [51, 52].

HRDM supports a single, discrete, and infinite temporal domain, and a single time dimension. HRDM relations are finite. The treatment of relation attributes is not uniform: some of them are designated as parts of the *key*. Non-key attributes can take values that are *functions* whose domains are subsets of the temporal domain (thus HRDM provides a limited form of *non-1NF* relations). It is not clear, however, which subsets of the temporal domain can serve as domains of functions, nor how they are specified, which makes it difficult to assess the data expressiveness of the model or the computational complexity of answering queries. (Under the assumption that the domains of functions are required to be finite, HRDM relations can only represent finite abstract temporal relations.) Specific ways of efficiently storing functions in HRDM relations are also proposed. For example, if a function is constant, storing one value together with the domain of the function is sufficient. Stepwise constant functions are handled similarly.

HRDM has an algebra obtained from the relational algebra by redefining most of the relational operators in the context of the HRDM data model. The semantics of the HRDM relational algebra operations are defined set-theoretically. As noted in [30] the algebra has a rather limited expressive power because it can not express queries that relate database snapshots at different time instants. Consequently, it can not express the query from example 4.1. Because of its essentially first-order character, the algebra can not also express inductive temporal queries. The data complexity of the algebra and representation-independence of queries have not been analyzed. The extension to SQL proposed in [52] was shown there not to be representation-independent.

5.4 Backlogs

A rather different temporal data model, based on *backlog relations,* was proposed in [61, 60]. The model supports a single, discrete, and infinite temporal domain, and two temporal dimensions: valid and transaction-time (which are not independent, unlike in TQuel). Backlog relations store not raw data but rather requests to change the data. Therefore only *transaction* time instants are stored. To find out whether a specific fact was true at a given *valid* time instant, the appropriate backlog relation has to be scanned to make sure that the appropriate tuple was inserted and not subsequently deleted (modification is handled similarly). Backlog relations are finite and can represent only finite abstract temporal relations. An example backlog relation B representing a part of the European history database introduced in section 2 is shown in Table 5. This relation has five attributes: the first contains the consecutive number of the update operation, the second – the name of the operation itself, the third – transaction time, the fourth and fifth – the relevant data (country, capital).

Assuming every transaction can generate only a single request to insert or delete a specific fact, a given abstract temporal relation is uniquely represented by a backlog relation. This guarantees representation independence of query languages over backlogs. Moreover, since the matching *insert-delete* pair in a backlog relation corresponds to the endpoints a TQuel interval, the succinctness of backlogs is identical to that of TQuel.

The backlog model can support any relational query language, for example relational algebra or calculus. In particular, the query *"list all countries that lost and regained independence"* from example 4.1 can be written in the algebra (with renaming) as:

$$\sigma_{O_1=Insert \wedge O=Delete \wedge O_2=Insert \wedge T_1<T<T_2}(R)$$

where

$$R = B(Id_1, O_1, T_1, X, S_1) \bowtie B(Id, O, T, X, S) \bowtie B(Id_2, O_2, T_2, X, S_2))$$

and B is the backlog relation in Table 5.

The data complexity of processing relational algebra or calculus queries to backlog relations is clearly polynomial. [60] describes many techniques for incremental evaluation of queries in this model.

5.5 Limitations of temporal data models and query languages

There are many more different temporal data models and query languages. Some of them are, like HRDM, not fully relational by allowing non-1NF relations or limited forms of object identity [30]. The recent book [110] presents at least 12 different temporal data models that are extensions of the relational model. A specific temporal data model is obtained, as above, by choosing a single fixed temporal domain, adopting a specific framework for concrete temporal databases (usually of the timestamp variety), and defining one or more query languages that are applicable only to this particular framework. If a timestamp framework is selected, then it is often assumed that the timestamps are finite or bounded sets. Consequently, the abstract temporal databases represented have to be finite. Moreover, the signatures of the time domain contain at most the order relation symbol. A serious limitation of the expressiveness of the temporal query languages discussed in the book [110] (except for the languages discussed in one of its chapters [12]) is their inability to express inductive temporal queries.

Virtually all temporal data models are mutually incompatible. This situation seriously hinders further systematic progress in the area of temporal databases, as remarked in [63]. Their solution, the provision of a single unifying temporal data model to which other models could be mapped, is not sufficient. They still fall short of defining a *representation-independent* abstract semantics for temporal databases. In fact, their model uses another notion of concrete temporal database, admittedly simpler and more general than others. This model is still limited as it deals with a single temporal domain and timestamps that are finite sets. We also think that the model is unnecessarily complicated. It violates Ockham's razor[1] by introducing a new type of entity - the "bitemporal element". The objectives of [63] can be achieved in a much simpler way by adopting the framework described here in section 2.

Current work on the interoperability of temporal databases, e.g., [129], addresses similar concerns as the present paper. The snapshot and timestamp views are identified without, however, referring to the underlying model-theoretic view. Only finite snapshot databases and a single, fixed temporal domain are considered. On the other hand, it should be pointed out that [129] deals with the issue of multiple time granularities that we do not address here.

By treating a temporal database as a special kind of first-order structure the transfer of results and techniques from mathematical logic and database theory becomes possible. Database theory in particular supplies the concepts of *query expressiveness* and *data complexity* that are necessary for the analysis of the mathematical properties of query languages. We think that there is no single notion of *completeness* of query languages for temporal databases. [30] proposed first-order logic (the language L'_D) as a "temporally-complete" query language. But in this language inductive temporal queries are not expressible. Moreover, the claim of [30] that temporal logic is equally expressive as first-order logic has not been substantiated yet. There is strong evidence to the contrary [65]. We conjecture that their claim does not hold even for finite databases. In the context of relational databases it has been shown in [18, 19, 17, 3] and other papers that there is no single complete query language but rather many different classes of query languages related to one another in intricate ways. We believe the situation in temporal databases is similar.

6 Incomplete temporal information

Temporal information may often be incompletely specified. For example, only a partial ordering of events may be given. One way of dealing with such problems is to give up linearity and well-known mathematical structures like $\mathbf{N}$ and $\mathbf{R}$, and develop a temporal logic based on events [118]. Another: to generalize the notion of a temporal database. We describe the latter solution as it has been predominant in the database area.

To deal with incomplete temporal information, several authors proposed to apply the classic framework of [57, 54] to *timestamp* databases. [57] modelled incomplete information in the context of the relational data model using *marked nulls* – placeholders standing for

[1] "A rule stating that entities should not be multiplied needlessly, which is interpreted to mean that the simplest of two or more competing theories is preferable or that an explanation for unknown phenomena should first be attempted in terms of what is already known." *Webster's II New Riverside University Dictionary.*

some value in the domain. The same null value may appear in different columns and different rows of a table. Moreover, every row has a (quantifier-free) local condition associated with it that contains some nulls of this row. Finally, the entire table has a (quantifier-free) global condition relating nulls in different rows (this was proposed in [54]).

It should be clear now that there is a close correspondence between local conditions in tables and quantifier-free timestamp formulas. Koubarakis [75, 77, 76, 78] pursued this correspondence in full generality in the context of the temporal domains $\mathbf{Q}$ and $\mathbf{Z}$. In his approach, there may be one or two temporal dimensions (i.e., one or two variables in timestamp formulas). Timestamp formulas which already contain variables (e.g., $5 < t$) may additionally contain nulls (e.g., $5 < c \wedge c = t$ where c is a null). We call such timestamp formulas *indefinite*. Indefinite timestamp formulas define indefinite timestamps which are just sets of timestamps. An *indefinite timestamp table* is a finite set of tuples with indefinite timestamp formulas (local conditions) and a global condition.

The semantics $sem(T)$ of a indefinite temporal table T is defined in two steps [76]. First, $rep(T)$ is the set of timestamp relations obtained by substituting domain values for nulls in T in such a way that the global condition of T is satisfied. Second, $sem(T)$ is the set of abstract temporal relations corresponding to the timestamp relations in $rep(T)$.

Example 6.1 Suppose we do not know the exact time of the split of Czechoslovakia into two different countries but we know that it was before 1993. This incomplete knowledge can be represented as the table in Table 6 with the global condition $c < 1993$.

As far as query languages for incomplete temporal databases are concerned, the semantics of *abstract* query languages discussed in section 4 generalize easily. The semantics of an indefinite timestamp table T is given by a set $sem(T)$ of abstract temporal relations, thus an answer to a query Q is also an indefinite timestamp table representing the set of abstract temporal relations corresponding to the answers to Q obtained separately for the individual members of $sem(T)$. In addition, modal queries can be asked: which facts are *certain* (true in every member of $sem(T)$), and which are *possible* (true in some member of $sem(T)$)?

Example 6.2 Consider Table 6. The query

$$\exists s.possible(I(\text{Czechoslovakia}, s, 1992))$$

returns True, while

$$\exists s.certain(I(\text{Czechoslovakia}, s, 1992))$$

returns False because the split may have occurred in an earlier year. On the other hand, the query

$$\exists s.certain(I(\text{Slovakia}, s, 1993))$$

returns True because Slovakia was for certain independent in 1993.

The implementation of the abstract query languages for incomplete timestamp databases raises new problems because of the presence of global conditions and timestamp formulas with nulls. Also, modal queries need to be supported. [75, 77] studied relational calculus and algebra for incomplete timestamp databases. He has shown [78] than the complexity of query processing for such databases is no worse than that for incomplete relational databases [54].

[119] obtained the corresponding lower bounds in a much more restricted framework. Only single values, null or non-null, can constitute timestamps. (Consequently, only finite abstract temporal databases can be represented in this framework.) Nulls can be related through a global conjunctive condition. There may be arbitrarily many temporal dimensions but not surprisingly the complexity of evaluating queries crucially depends on this number. [119] shows that the data complexity of obtaining *certain* answers to first-order queries is:

- in PTIME for one-dimensional time (for disjunctive queries the original proof is non-constructive, although it can be made constructive [120]),
- co-NP-complete for n-dimensional time ($n \geq 2$).

[119] has also many very interesting results about the combined complexity of evaluating queries that we can not discuss here because of the lack of space. The results of [119] apply to $\mathbf{Z}$, $\mathbf{Q}$, as well as finite orders.

[38, 53, 52] pursued the problem of incomplete temporal information for *concrete* query languages. They didn't base their approaches on [57] but to some degree their approaches can be recast using the framework of [76]. In the context of the discrete temporal domain $\mathbf{Z}$, [38] postulated to represent an indefinite point by a *possible interval*, which could be viewed as an indefinite timestamp formula (e.g., $t = c \wedge 0 < c < 5$ meaning "*some* instant between 0 and 5"), and an indefinite interval by a pair of possible intervals. Null values appearing in timestamp formulas associated with different tuples can not be compared, even for equality. Thus the incomplete temporal relation from example 6.1 can not be represented in their framework. On the other hand, a quadratic query evaluation algorithm becomes possible for queries expressed in a (slightly restricted) subset of TQuel [39]. [38] also proposed to incorporate several kinds of *probabilistic* information in incomplete temporal databases and described a possible implementation. In the context of the discrete temporal domain $\mathbf{N}$, [53, 52] proposed to represent an indefinite timestamp by a pair (*lower bound*, *upper bound*) where both bounds are finite unions of intervals. This can be easily represented using an indefinite timestamp formula. For this approach, precise expressiveness and complexity bounds on query processing have still to be determined.

[20] was historically the first to deal with the problem of incomplete temporal information. His approach, however, does not completely fit in the framework presented above. In his approach a temporal database records information about events (points or intervals) and their relationships. Characteristically, the relationships are just attribute values, e.g., `''before''` is a constant that may appear in a database. This makes it possible to formulate queries asking for all relationships between two events in a purely first-order language. The set of allowed relationships is user-defined but the derivable relationships

between events should be specified in a special, essentially Horn, clausal form. This makes query evaluation efficient (polynomial time). However, the derivation of disjunctions is not possible, which makes the aproach incomplete for disjunctive queries. Also, [20] discussed only the evaluation of atomic queries. Some of the above work can be recast using indefinite timestamp tables. Point events can be viewed as null values and their ordering captured by a global condition [21].

7 Related work in AI

We briefly summarize here the differences between the database and the AI perspectives on the representation and processing of temporal information.

First, most AI approaches are restricted to the propositional case (the non-temporal part is propositional). The exceptions include [79, 34]. Second, they often take an interval-based view of time that originates in [5]. The approach of [5] was to associate every proposition with an interval in which it holds. [5] proposed an algebra of intervals based on 13 basic kinds of interval relationships, e.g., `before`, and operations on those relationships which included Boolean operations and composition. Moreover, two intervals could be related through a set of relationships which represented a disjunction. Thus a rich array of disjunctive information could be represented.

Example 7.1 The fact that two intervals I_1 and I_2 are disjoint can be represented as

$$I_1\{\texttt{before}, \texttt{after}\}I_2.$$

This can also be expressed as a logical formula (assume I^- is the beginning and I^+ the end of an interval, so $I^- < I^+$):

$$I_1^+ < I_2^- \vee I_2^+ < I_1^-.$$

Because of the possibility of formulating sets of interval relationships as quantifier-free formulas (as above), the algebra of [5] is subsumed in terms of expressive power by the framework of [78, 76] described in section 6. Determining satisfaction of a set of interval relationships turned out to be NP-complete [128]. Thus more restricted PTIME-algebras were proposed [128, 117, 93]. (It would be interesting to compare those algebras with tractable subclasses of indefinite timestamp databases identified in [119].) Later works [35, 71, 89] introduced constructs for representing distance information. Again, this can be represented in a first-order framework, assuming the signature of the time domain contains successor or addition.

The third difference is in the query languages supported and query evaluation. Allowed queries typically involve establishing a relationship between points or intervals. The representation languages are propositional, thus unable to represent a relational database.

Consequently, first-order queries involving quantifiers are not supported. Queries are evaluated using constraint satisfaction algorithms. It is an open question whether the above approaches can be generalized to first-order logic.

Finally, AI research is often concerned not only with the representation and querying of temporal information but also with *temporal reasoning*: drawing conclusions using additional assumptions like persistence [79, 34] or defaults, as well as special procedures like abduction [44, 36].

8 Conclusions

In this survey we have proposed a single and uniform formal framework for studying temporal databases and temporal query languages. We have applied this framework to a wide spectrum of query languages. We have also identified a number of important mathematical properties of query languages. We believe that the framework can serve as a foundation for further, more systematic, advances in the area of temporal databases.

Acknowledgments

We are very grateful to Marianne Baudinet, Michael Boehlen, Surajit Chaudhuri, Curtis Dyreson, Manolis Koubarakis, Peter Revesz, Rick Snodgrass, David Toman, and Pierre Wolper for sharing with us their insights in and knowledge of temporal databases. The members of the fall 1994 database theory class at KSU are thanked for their incisive comments.

References

[1] M. Abadi and Z. Manna. Temporal Logic Programming. *Journal of Symbolic Computation*, 8(3), September 1989.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[3] S. Abiteboul and V. Vianu. Expressive Power of Query Languages. In *Theoretical Studies in Computer Science*. Academic Press, 1992.

[4] I. Ahn. Database Issues in Telecommunications Network Management. In *ACM SIGMOD International Conference on Management of Data*, 1994.

[5] J.F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[6] J.F. Allen and P.J. Hayes. Moments and Points in an Interval-Based Temporal Logic. *Computational Intelligence*, 5:223–238, 1989.

[7] K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In Minker [90], pages 89–148.

[8] F. Bacchus, J. Tenenberg, and J.A. Koomen. A Non-Reified Temporal Logic. *Artificial Intelligence*, 52(1):87–108, 1991.

[9] M. Baudinet. Temporal Logic Programming is Complete and Expressive. In *ACM Symposium on Principles of Programming Languages*, 1989.

[10] M. Baudinet. A Simple Proof of the Completeness of Temporal Logic Programming. In L. Fariñas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming.* Oxford University Press, 1992.

[11] M. Baudinet. On the Expressiveness of Temporal Logic Programming. *Information and Computation*, 1994. To appear.

[12] M. Baudinet, J. Chomicki, and P. Wolper. Temporal Deductive Databases. In Tansel et al. [110], pages 294–320.

[13] M. Baudinet, J. Chomicki, and P. Wolper. Constraint-Generating Dependencies. In *International Conference on Database Theory*, Prague, Czech Republic, January 1995. Short version in: Proc. 2nd Workshop on Principles and Practice of Constraint Programming, 1994.

[14] M. Baudinet, M. Niézette, and P. Wolper. On the Representation of Infinite Temporal Data and Queries. In *ACM Symposium on Principles of Database Systems*, 1991.

[15] C. Beeri. A formal approach to object-oriented databases. *Data and Knowledge Engineering*, 5:353–382, 1990.

[16] A. Brodsky, J. Jaffar, and M.J. Maher. Towards Practical Constraint Databases. In *International Conference on Very Large Data Bases*, 1993.

[17] A.K Chandra. Theory of Database Queries. In *ACM Symposium on Principles of Database Systems*, pages 1–9, 1988.

[18] A.K. Chandra and D. Harel. Computable Queries for Relational Databases. *Journal of Computer and System Sciences*, 21:156–178, 1980.

[19] A.K. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.

[20] S. Chaudhuri. Temporal Relationships in Databases. In *International Conference on Very Large Data Bases*, 1988.

[21] S. Chaudhuri, 1994. Personal communication.

[22] J. Chomicki. *Functional Deductive Databases: Query Processing in the Presence of Limited Function Symbols.* PhD thesis, Rutgers University, New Brunswick, New Jersey, January 1990. Also Laboratory for Computer Science Research Technical Report LCSR-TR-142.

[23] J. Chomicki. Polynomial-Time Computable Queries in Temporal Deductive Databases. In *ACM Symposium on Principles of Database Systems*, Nashville, Tennessee, April 1990.

[24] J. Chomicki. History-less Checking of Dynamic Integrity Constraints. In *IEEE International Conference on Data Engineering*, Phoenix, Arizona, February 1992.

[25] J. Chomicki. Depth-Bounded Bottom-Up Evaluation of Logic Programs. *Journal of Logic Programming*, 1995. To appear.

[26] J. Chomicki and T. Imieliński. Temporal Deductive Databases and Infinite Objects. In *ACM Symposium on Principles of Database Systems*, Austin, Texas, March 1988.

[27] J. Chomicki and T. Imieliński. Finite Representation of Infinite Query Answers. *ACM Transactions on Database Systems*, 18(2):181–223, June 1993.

[28] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. In *IEEE International Conference on Data Engineering*, 1987.

[29] J. Clifford and A. Croker. The Historical Relational Data Model (HRDM) Revisited. In Tansel et al. [110], pages 6–27.

[30] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM Transactions on Database Systems*, 19(1):64–116, March 1994.

[31] J. Clifford and T. Isakowitz. On the Semantics of (Bi) Temporal Variable Databases. In *International Conference on Extending Database Technology*, Cambridge, UK, March 1994.

[32] J. Clifford and A.U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In *ACM SIGMOD International Conference on Management of Data*, 1985.

[33] J. Clifford and D.S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[34] T.L. Dean and D.V. McDermott. Temporal Data Base Management. *Artificial Intelligence*, 32(1):1–55, 1987.

[35] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.

[36] M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal Reasoning with Abductive Event Calculus. In *European Conference on Artificial Intelligence*, 1992.

[37] W. Dreyer, A.K. Dittrich, and D. Schmidt. Research Perspectives for Time Series Management Systems. *SIGMOD Record*, 23(1):10–15, March 1994.

[38] C. E. Dyreson and R.T. Snodgrass. Historical Indeterminacy. In *IEEE International Conference on Data Engineering*, 1993.

[39] C.E. Dyreson, 1994. Personal communication.

[40] C.E. Dyreson, M.D. Soo, and R.T. Snodgrass. The TSQL2 Data Model for Time. A TSQL2 Commentary, March 1994.

[41] R. Elmasri, G.T.J. Wuu, and V. Kouramajian. The Time Index and the Monotonic B+-tree. In Tansel et al. [110], pages 433–456.

[42] E.A. Emerson. Temporal and Modal Logic. In van Leeuwen [123], chapter 16, pages 995–1072.

[43] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.

[44] K. Eshghi. Abductive Planning with Event Calculus. In *International Conference on Logic Programming*, 1988.

[45] M. Finger. Handling Database Updates in Two-Dimensional Temporal Logic. *Journal of Applied Non-Classical Logic*, 1992.

[46] International Organization for Standardization. Database Language SQL. ISO/IEC 9075:1992, 1992.

[47] D. Gabbay. The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In B. Banieqbal, B. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398, pages 409–448. Springer-Verlag, LNCS 398, 1989.

[48] D. Gabbay and P. McBrien. Temporal Logic and Historical Databases. In *International Conference on Very Large Data Bases*, 1991.

[49] D. Gabbay, A. Pnueli, S. Shelah, and S. Stavi. On the Temporal Analysis of Fairness. In *ACM Symposium on Principles of Programming Languages*, 1980.

[50] D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press, 1994.

[51] S.K. Gadia. A Homogenous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[52] S.K. Gadia. Temporal Databases: A Prelude to Parametric Data. In Tansel et al. [110], pages 28–66.

[53] S.K. Gadia, S.S. Nair, and Y.C. Poon. Incomplete Information in Relational Temporal Databases. In *International Conference on Very Large Data Bases*, 1992.

[54] G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer-Verlag, LNCS 554, 1991.

[55] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*. Springer-Verlag, LNCS 736, 1993.

[56] J.Y. Halpern and Y. Shoham. A Propositional Modal Logic of Time Intervals. *Journal of the ACM*, 38(4):935–962, October 1991.

[57] T. Imielinski and W. Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.

[58] J. Jaffar and J-L. Lassez. Constraint Logic Programming. In *ACM Symposium on Principles of Programming Languages*, 1987.

[59] C.S. Jensen, J. Clifford, S. Gadia, A. Segev, and R. Snodgrass. A Glossary of Temporal Database Concepts. In Tansel et al. [110], pages 621–633.

[60] C.S. Jensen and L. Mark. Differential Query Processing in Transaction-Time Databases. In Tansel et al. [110], pages 457–496.

[61] C.S. Jensen, L. Mark, and N. Roussopoulos. Incremental Implementation Model for Relational Databases with Transaction Time. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):461–473, 1991.

[62] C.S. Jensen and R.T. Snodgrass. Temporal Specialization. In *IEEE International Conference on Data Engineering*, pages 594–603, 1992.

[63] C.S. Jensen, M.D. Soo, and R.T. Snodgrass. Unification of Temporal Data Models. In *IEEE International Conference on Data Engineering*, 1993.

[64] F. Kabanza, J-M. Stevenne, and P. Wolper. Handling Infinite Temporal Data. In *ACM Symposium on Principles of Database Systems*, pages 392–403, Nashville, Tennessee, April 1990.

[65] H. Kamp. Formal properties of 'now'. *Theoria*, 37:227–273, 1971.

[66] J.A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[67] P.C. Kanellakis. Elements of Relational Database Theory. In van Leeuwen [123], chapter 17, pages 1073–1158.

[68] P.C. Kanellakis and D. Q. Goldin. Constraint Programming and Database Query Languages. In *Conference on Theoretical Aspects of Computer Software*. Springer-Verlag, 1994.

[69] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint Query Languages. In *ACM Symposium on Principles of Database Systems*, pages 299–313, Nashville, Tennessee, April 1990. To appear in Journal of Computer and System Sciences.

[70] P.C. Kanellakis, S. Ramaswamy, D.E. Vengroff, and J.S. Vitter. Indexing for Data Models with Constraints and Classes. In *ACM Symposium on Principles of Database Systems*, 1993.

[71] H. Kautz and P. Ladkin. Integrating Metric and Qualitative Temporal Reasoning. In *National Conference on Artificial Intelligence*, 1991.

[72] S-K. Kim and S. Chakravarthy. Modeling Time: Adequacy of Three Distinct Time Concepts for Temporal Databases. In *International Conference on Entity-Relationship Approach*, Arlington, Texas, December 1993.

[73] P.G. Kolaitis and C.H. Papadimitriou. Why not Negation by Fixpoint? *Journal of Computer and System Sciences*, 43:125–144, 1991.

[74] C.P. Kolovson. Indexing Techniques for Historical Databases. In Tansel et al. [110], pages 418–432.

[75] M. Koubarakis. Representation and Querying in Temporal Databases: the Power of Temporal Constraints. In *IEEE International Conference on Data Engineering*, 1993.

[76] M. Koubarakis. Database Models for Infinite and Indefinite Temporal Information. *Information Systems*, 19(2):141–174, March 1994.

[77] M. Koubarakis. Foundations of Indefinite Constraint Databases. In *International Workshop on Principles and Practice of Constraint Programming*, Orcas Island, Washington, May 1994. LNCS 874, Springer-Verlag.

[78] M. Koubarakis. *Foundations of Temporal Constraint Databases*. PhD thesis, National Technical University of Athens, 1994.

[79] R. Kowalski and M. Sergot. A Logic-based Calculus of Events. *New Generation Computing*, 4:67–95, 1986.

[80] P. Ladkin. Primitives and Units for Time Specification. In *National Conference on Artificial Intelligence*, pages 354–359, 1986.

[81] P. Ladkin. *The Logic of Time Representation*. PhD thesis, University of California, Berkeley, 1987.

[82] T.Y. Cliff Leung and R.R. Muntz. Stream Processing: Temporal Query Processing and Optimization. In Tansel et al. [110], pages 329–355.

[83] G. Ligozat. On Generalized Interval Calculi. In *European Conference on Artificial Intelligence*, pages 234–240, 1992.

[84] U.W. Lipeck and G. Saake. Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems*, 12(3):255–269, 1987.

[85] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2nd edition, 1987.

[86] D. Lomet and B. Salzberg. Transaction-Time Databases. In Tansel et al. [110], pages 388–417.

[87] R. Maiocchi and B. Pernici. Temporal Data Management Systems: A Comparative View. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):504–524, December 1991.

[88] L.E. Jr. McKenzie and R.T. Snodgrass. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4):501–543, December 1991.

[89] I. Meiri. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *National Conference on Artificial Intelligence*, 1991.

[90] J. Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1988.

[91] A. Montanari and B. Pernici. Temporal Reasoning. In Tansel et al. [110], pages 534–562.

[92] A. Montanari and B. Pernici. Towards a Temporal Logic Reconstruction of Temporal Databases. In *Proc. International Workshop on an Infrastructure for Temporal Databases*, Arlington, Texas, June 1993.

[93] B. Nebel and H-J. Bürckert. Reasoning About Temporal Relations: a Maximal Tractable Subclass of Allen's Interval Algebra. In *National Conference on Artificial Intelligence*, 1993.

[94] M. Niézette and J-M. Stévenne. An Efficient Symbolic Representation of Periodic Time. In *International Conference on Information and Knowledge Management*, 1992.

[95] M.A. Orgun and H. A. Müller. A Temporal Algebra Based on an Abstract Model. In M.E. Orlowska and M. Papazoglou, editors, *Advances in Database Research: Proceedings of the 4th Australian Database Conference*, pages 301–316. World Scientific, 1993.

[96] J. Paredaens, J. Van den Bussche, and D. Van Gucht. Towards a Theory of Spatial Database Queries. In *ACM Symposium on Principles of Database Systems*, 1994.

[97] T. C. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In Minker [90], pages 193–216.

[98] J.C.G. Ramirez, L.A. Smith, and L.L. Patterson. Medical Information Systems: Characterization and Challenges. *SIGMOD Record*, 23(3):44–53, September 1994.

[99] P.Z. Revesz. A Closed Form for Datalog Queries with Integer Order. In *International Conference on Database Theory*, pages 187–201. Springer-Verlag, LNCS 470, 1990.

[100] P.Z. Revesz. A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science*, 116:117–149, 1993.

[101] J.F. Roddick and J.D. Patrick. Temporal Semantics in Information Systems - A Survey. *Information Systems*, 17(3):249–267, 1992.

[102] A. Segev. Join Processing and Optimization in Temporal Relational Databases. In Tansel et al. [110], pages 356–387.

[103] A.P. Sistla and O. Wolfson. Temporal Triggers in Active Databases. *IEEE Transactions on Knowledge and Data Engineering*, 1995. To appear.

[104] R. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[105] R. Snodgrass. Temporal Object-Oriented Databases: A Critical Comparison. In W. Kim, editor, *Modern Database Systems*, pages 386–408. Addison-Wesley, 1994.

[106] R. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, 19(9), 1986.

[107] R. Snodgrass et al. A TSQL2 Tutorial. *SIGMOD Record*, 23(3):27–34, September 1994.

[108] R. Snodgrass et al. TSQL2 Language Specification. *SIGMOD Record*, 23(1):65–86, March 1994.

[109] R.T. Snodgrass. Temporal Databases. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 22–64. Springer-Verlag, LNCS 639, 1992.

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

[111] A.U. Tansel. A Generalized Relational Framework for Modeling Temporal Data. In Tansel et al. [110], pages 183–201.

[112] C.I. Theodoulidis and P. Loucopoulos. The Time Dimension in Conceptual Modelling. *Information Systems*, 16(3):273–300, 1991.

[113] W. Thomas. A Combinatorial Approach to the Theory of ω-automata. *Information and Control*, 48(3):261–283, 1981.

[114] D. Toman, J. Chomicki, and D.S. Rogers. Datalog with Integer Periodicity Constraints. In *International Logic Programming Symposium*, Ithaca, New York, November 1994.

[115] A. Tuzhilin and J. Clifford. A Temporal Relational Algebra as a Basis for Temporal Relational Completeness. In *International Conference on Very Large Data Bases*, 1990.

[116] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[117] P. van Beek. Reasoning about Qualitative Temporal Information. *Artificial Intelligence*, 58:297–326, 1992.

[118] J.F.A.K. van Benthem. *The Logic of Time.* D.Reidel, 2nd edition, 1991.

[119] R. van der Meyden. The Complexity of Querying Indefinite Data about Linearly Ordered Domains. In *ACM Symposium on Principles of Database Systems*, 1992.

[120] R. van der Meyden, 1994. Personal communication.

[121] M.H. van Emden and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, 1976.

[122] A. Van Gelder and R.W. Topor. Safety and Translation of Relational Calculus Queries. *ACM Transactions on Database Systems*, 16(2):235–278, June 1991.

[123] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science.* Elsevier/MIT Press, 1990.

[124] M. Y. Vardi. A Temporal Fixpoint Calculus. In *ACM Symposium on Principles of Programming Languages*, 1988.

[125] M.Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing*, pages 137–146, 1982.

[126] M.Y. Vardi. Fundamentals of Dependency Theory. In E. Börger, editor, *Trends in Theoretical Computer Science*, pages 171–224. Computer Science Press, 1988.

[127] Y. Venema. Expressiveness and Completeness of an Interval Tense Logic. *Notre Dame Journal of Formal Logic*, 31(4):529–547, 1990.

[128] M. Vilain and H. Kautz. Constraint Propagation Algorithms for Temporal Reasoning. In *National Conference on Artificial Intelligence*, 1986.

[129] X.S. Wang, S. Jajodia, and V.S. Subrahmanian. Temporal Modules: An Approach Toward Federated Temporal Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 227–236, 1993.

[130] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56:72–99, 1983.

Country	*Capital*	*Years of Independence*
Czech Kingdom	Prague	$\{1198, \ldots, 1620\}$
Czechoslovakia	Prague	$\{1918, \ldots, 1938\} \cup \{1945, \ldots, 1992\}$
Czech Republic	Prague	$\{1993, \ldots\}$
Slovakia	Bratislava	$\{1939, \ldots, 1945\} \cup \{1993, \ldots\}$
Poland	Gniezno	$\{1025, \ldots, 1039\}$
Poland	Cracow	$\{1040, \ldots, 1595\}$
Poland	Warsaw	$\{1596, \ldots, 1794\} \cup \{1918, \ldots, 1938\} \cup \{1945, \ldots\}$

Table 1: Timestamp view

Year	*Independent Countries*
...	...
1917	{}
1918	{(Czechoslovakia,Prague),(Poland,Warsaw)}
...	...
1939	{(Slovakia,Bratislava)}
1945	{(Czechoslovakia,Prague),(Poland,Warsaw)}
...	...
1993	{(Czech Republic,Prague),(Slovakia,Bratislava),(Poland,Warsaw) }
...	...

Table 2: Snapshot view

Country	*Capital*	*Year of Independence*
...	...	...
Czechoslovakia	Prague	1918
Poland	Warsaw	1918
...	...	...
Slovakia	Bratislava	1939
...	...	...
Czechoslovakia	Prague	1945
Poland	Warsaw	1945
...	...	...
Czech Republic	Prague	1993
Slovakia	Bratislava	1993
Poland	Warsaw	1993
...	...	...

Table 3: Model-theoretic view

Country	*Capital*	*Years of Independence*
Czech Kingdom	Prague	$1198 \leq t < 1621$
Czechoslovakia	Prague	$1918 \leq t < 1939 \vee 1945 \leq t \leq 1992$
Czech Republic	Prague	$1992 < t$
Slovakia	Bratislava	$1939 \leq t < 1945 \vee 1992 < t$
Poland	Gniezno	$1025 \leq t < 1040$
Poland	Cracow	$1040 \leq t < 1596$
Poland	Warsaw	$1596 \leq t < 1795 \vee 1918 \leq t < 1939 \vee 1945 \leq t$

Table 4: Timestamp view with timestamp formulas

Id	*Op*	*Time*	*Country*	*Capital*
1	Insert	1198	Czech Kingdom	Prague
2	Delete	1621	Czech Kingdom	Prague
3	Insert	1918	Czechoslovakia	Prague
4	Delete	1939	Czechoslovakia	Prague
5	Insert	1939	Slovakia	Bratislava
6	Delete	1945	Slovakia	Bratislava
7	Insert	1945	Czechoslovakia	Prague
8	Delete	1993	Czechoslovakia	Prague
9	Insert	1993	Czech Republic	Prague
10	Insert	1993	Slovakia	Bratislava
...				

Table 5: Backlog representation

Country	*Capital*	*Years of Independence*
Czech Kingdom	Prague	$1198 \leq t < 1621$
Czechoslovakia	Prague	$1918 \leq t < 1939 \vee 1945 \leq t \leq c$
Czech Republic	Prague	$c < t$
Slovakia	Bratislava	$1939 \leq t < 1945 \vee c < t$
Poland	Gniezno	$1025 \leq t < 1040$
Poland	Cracow	$1040 \leq t < 1596$
Poland	Warsaw	$1596 \leq t < 1795 \vee 1918 \leq t < 1939 \vee 1945 \leq t$

Global condition
$c < 1993$

Table 6: Incomplete Information