Logics for Databases and Information Systems

Edited by

Jan Chomicki, Gunter Saake
Monmouth University / University of Magdeburg

Kluwer Academic Publishers
Boston/Dordrecht/London

Contents

1 TEMPORAL LOGIC IN INFORMATION SYSTEMS

Jan Chomicki
and David Toman

Abstract: Temporal logic is obtained by adding temporal connectives to a logic language. Explicit references to time are hidden inside the temporal connectives. Different variants of temporal logic use different sets of such connectives. In this chapter, we survey the fundamental varieties of temporal logic and describe their applications in information systems.

Several features of temporal logic make it especially attractive as a query and integrity constraint language for temporal databases. First, because the references to time are hidden, queries and integrity constraints are formulated in an abstract, representation-independent way. Second, temporal logic is amenable to efficient implementation. Temporal logic queries can be translated to an algebraic language. Temporal logic constraints can be efficiently enforced using auxiliary stored information. More general languages, with explicit references to time, do not share these properties.

Recent research has proposed various implementation techniques to make temporal logic practically useful in database applications. Also, the relationships between different varieties of temporal logic and between temporal logic and other temporal languages have been clarified. We report on these developments and outline some of the remaining open research problems.

1.1 INTRODUCTION

Time is ubiquitous in information systems. Almost every enterprise faces the problem of its data becoming out of date. However, such data is often valuable, so it should be archived and some means to access it should be provided. Also, some data may be inherently historical, e.g., medical, cadastral, or judicial records. Temporal databases provide a uniform and systematic way of dealing with historical data. Many languages have been proposed for temporal databases, among others *temporal logic*. Temporal logic combines abstract, formal semantics with the amenability to efficient implementation. This chapter shows how temporal logic can be used in temporal database applications. Rather than presenting new results, we report on recent developments and survey the field in a systematic way using a unified formal framework [GHR94; Cho94]. The handbook [GHR94] is a comprehensive reference on mathematical foundations of temporal logic.

In this chapter we study how temporal logic is used as a *query* and *integrity constraint* language. Consequently, model-theoretic notions, particularly *formula satisfaction*, are of primary interest. Axiomatic systems and proof methods for temporal logic [GHR94] have found so far relatively few applications in the context of information systems. Moreover, one needs to bear in mind that for the standard linearly-ordered time domains temporal logic is not recursively axiomatizable [GHR94], so recursive axiomatizations are by necessity incomplete.

Databases are inherently first-order structures and thus in this chapter we are primarily interested in *first-order* temporal logic. This is in sharp contrast with another major application area of temporal logic, *program verification*, where the formalisms studied are usually propositional [MP92; Pnu86].

We introduce here a number of fundamental concepts and distinctions that are used throughout the chapter. First, there is a choice of *temporal ontology*, which can be based either on time *points* (instants) or *intervals* (periods). In most database applications the point-based view is more natural and thus we concentrate on it in this chapter. However, in section 1.5 we briefly discuss interval-based temporal logic. (Intervals are predominant in AI applications.) Second, time can be single-dimensional or multi-dimensional. Multiple time dimensions can occur if, for example, multiple kinds of time (e.g., transaction time and valid time [SA86]) are required in an application. In addition we show that multiple temporal dimensions are *necessary* to evaluate general first-order temporal queries. Except for section 1.5, we adopt the single-dimensional view. Finally, there is a choice of *linear* vs. *nonlinear* time, i.e., whether time should be viewed as a single line or rather as a tree [Eme90], or even an acyclic graph [Wol89]. Although nonlinear time is potentially applicable to some database problems like version control or workflows, there has been very little work in this area. Therefore, in this chapter we concentrate on temporal domains that are linearly ordered sets.

Definition 1 (Temporal Domain) *A single-dimensional linearly ordered temporal domain is a structure* $\mathrm{T}_P = (T, <)$*, where* T *is a set of time instants and* $<$ *is a linear order on* T*.*

Often, a temporal domain has also a distinguished element 0, typically standing for the *beginning of time*. For full generality, we allow also negative time. The standard temporal domains are: natural numbers $\mathbf{N} = (N, 0, <)$, integers $\mathbf{Z} = (Z, 0, <)$, rationals $\mathbf{Q} = (Q, 0, <)$, and reals $\mathbf{R} = (R, 0, <)$.

The rest of the chapter is organized as follows. Section 1.2 shows several ways to introduce time into the standard relational model and defines the fundamental notions of *temporal databases*. It also shows how such databases naturally arise as *histories* of ordinary relational databases. Section 1.3 introduces temporal logic as a *query language* for temporal databases and shows its place among other temporal query languages. It also introduces techniques needed for efficient *query evaluation* over compact representations of temporal databases and in the end discusses more practical temporal query languages and their relationship to temporal logic. Section 1.4 shows how temporal logic can be used as a language for specifying temporal *integrity constraints* and discusses the issues involved in the maintenance of such constraints. Sections 1.5 and 1.6 focus on the *limitations* of a single-dimensional first-order temporal logic and on different ways of overcoming these limitations. Section 1.5 introduces *multidimensional* temporal connectives and Section 1.6 discusses *non-first-order* extensions of temporal logic. Section 1.7 contains brief conclusions.

1.2 TEMPORAL DATABASES

Before we discuss temporal logic and other temporal query languages we need to introduce the underlying data model—*temporal databases*. We focus on temporal databases defined as natural extensions of the standard relational model. A standard relational database is a *first-order structure* built from a *data domain* D, usually equipped with a built-in equality (diagonal) relation. This domain is extended to a *relational database* by adding to it a finite instance $(\mathbf{r}_1, \ldots, \mathbf{r}_k)$ of a user-defined relational database schema $\rho = (r_1, \ldots, r_k)$ over D. A natural first-order query language over such databases—the relational calculus—coincides with *first-order logic* over the vocabulary $(=, r_1, \ldots, r_k)$ of the above extended structure. An answer to a query in relational calculus is the set of valuations (tuples) that make the query true in the given relational database. Domain independent relational calculus queries (those that depend only on the instance of ρ and not on the underlying domain of values D) can be equivalently expressed in *relational algebra* [Cod72]. In this way the relational model provides a natural declarative paradigm for representing and querying information stored in a relational database, as well as the possibility of efficient implementation of queries through relational algebra. The choice of the relational model is very natural for studying first-order temporal logic because the semantics of that language is commonly defined with respect to a particular temporal extension of first-order relational structures.

1.2.1 Abstract Temporal Databases

It is useful to introduce a distinction between the representation-independent meaning of a temporal database and its concrete, finite representation. The former is termed an *abstract* temporal database and the latter a *concrete* one. Concrete databases are discussed in Section 1.3.3. Some query languages, including temporal logic, have their semantics defined in terms of abstract temporal databases—they will be termed *abstract* as well. Other languages whose semantics is defined in terms of *concrete databases* will be appropriately called *concrete*. Abstract query languages are discussed in Section 1.3.1, concrete ones in Section 1.3.4.

One obtains an abstract temporal database by linking a standard relational database with a temporal domain. There are several alternative ways of doing that [Cho94]:

The timestamp model is defined by augmenting all tuples in relations by an additional temporal attribute.

The snapshot model is defined as a mapping of the temporal domain to the class of standard relational databases. This gives a Kripke structure with the temporal domain serving as the accessibility relation.

The parametric model considers the values stored in individual fields of tuples in the database to be functions of time.

We do not consider the parametric model in this chapter as it is not directly relevant to temporal logic.

Definition 2 (Abstract Temporal Database) *Let* $\rho = (r_1, \ldots, r_k)$ *be a database schema,* D *a data domain and* T_P *a temporal domain.*

A relation symbol R_i *is a* temporal extension *of a relation symbol* r_i *if it contains all attributes of* r_i *and a single additional attribute* t *of sort* T_P *(w.l.o.g. we assume it is the first attribute). The sort of the attributes of* R_i *is* $\mathrm{T}_P \times \mathrm{D}^{\mathrm{arity}(r_i)}$.

A timestamp temporal database *is a first-order structure* $\mathrm{D} \cup \mathrm{T}_P \cup \{\mathbf{R}_1 \ldots, \mathbf{R}_k\}$*, where* $\mathbf{R}_i$ *are* temporal relations— *instances of the temporal extensions* R_i*. In addition we require that the set* $\{\mathbf{a} : (t, \mathbf{a}) \in \mathbf{R}_i\}$ *be finite for every* $t \in \mathrm{T}_P$ *and* $0 < i \leq k$.

A snapshot temporal database *over* D, T_P*, and* ρ *is a map* $\mathrm{DB} : \mathrm{T}_P \to \mathcal{DB}(\mathrm{D}, \rho)$*, where* $\mathcal{DB}(\mathrm{D}, \rho)$ *is the class of finite relational databases over* D *and* ρ.

It is easy to see that snapshot and timestamp abstract temporal databases are merely different views of the same data and thus can represent the same class of temporal databases.

Works		
Year	Name	Company
1990	John	IBM
1991	John	IBM
1992	John	IBM
1993	John	Microsoft
...	John	Microsoft
1984	Mary	DEC
1985	Mary	DEC
1990	Mary	IBM
...	Mary	IBM
1990	Steve	HP
...	Steve	HP

Education				
Year	Name	Degree	Major	School
1980	John	BS	CS	MIT
1986	John	MS	CS	Stanford
1990	John	PhD	CS	Stanford
1984	Mary	BS	CS	UofT
1990	Mary	PhD	CS	UofT
1990	Steve	MS	BA	Harvard

Figure 1.1 An instance of a temporal database from Example 3.

Example 3 *A head-hunting company* Brains 'R Us *is introducing a new kind of service for its corporate customers: All the job seekers' resumes are put in a temporal database that the customers can subsequently query. We discuss here only a fragment of this database, consisting of two relations* Works(Year,Name,Company) *and* Education(Year,Name,Degree,Major,School).

These two relations represent the user view of the database. They will likely be stored in a different, more space-efficient format that uses time intervals. An example instance is shown in Figure 1.1.

Here are several historical queries that can be expressed in temporal logic:

- *find all people who have worked for only one company.*
- *find all people whose terminal degree is from MIT.*
- *list all UofT PhD's in computer science that have been continuously employed by IBM since their graduation.*
- *find all job-hoppers—people who never spent more than two years in one place.*

1.2.2 Relational Database Histories

Relational databases are updatable and it is natural to consider sequences of database states resulting from the updates.

Definition 4 (History) *A* history *over a database schema ρ and a data domain D is a sequence $H = (H_0, \ldots, H_n, H_{n+1}, \ldots)$ of database instances (called* states*) such that*

1. *all the states $H_0, \ldots, H_n, H_{n+1}, \ldots$ share the same schema ρ and the same data domain* D*,*
2. *H_0 is the initial instance of the database,*
3. *H_i results from applying an update to H_{i-1}, $i \geq 1$.*

A finite *history is defined like a general history except that the sequence of states is finite.*

There is a clear correspondence between histories over D and ρ and snapshot temporal databases over D, **N** (natural numbers), and ρ (see Definition 2). Consequently, any query language $\mathcal{L}$ for abstract temporal databases can also be used to query database histories. However, there is a difference in the restrictions placed on updates: while there are no a priori limitations placed on snapshot temporal database updates (they can involve any snapshot), histories are *append-only* (the past cannot be modified).

A finite history is a finite sequence of finite states and thus can be represented in finite space. However, in most applications it is impractical to store all the past database states and some encoding is used. For integrity checking it is often the case that the encoding is lossy (to obtain significant space savings) and completely determined by the integrity constraints to be maintained. For an example of such an encoding see section 1.4.3.

1.3 TEMPORAL QUERIES

In this section we establish the place of *First-order Temporal Logic* among various query languages for temporal databases. Rather than concentrating solely on temporal logic, we show the relationships among various temporal extensions of the relational model. In this way we contrast the features and shortcomings of temporal logic with other temporal query languages.

First, we discuss two major approaches to introducing time into relational query languages. Both of them are developed in the context of *abstract* temporal databases and thus lead to abstract query languages. The first approach uses explicit variables (attributes) and quantifiers over the temporal domain; the second adds modal temporal connectives and hidden temporal contexts. We report on the relative expressive power of these extensions.

In the second part we concentrate on *concrete* temporal databases: space-efficient encodings of abstract temporal databases necessary from the practical point of view. First, we explore in detail the most common encoding of time based on intervals and the associated concrete query languages. We introduce semantics-preserving translations of abstract temporal query languages to their concrete counterparts. We also introduce a generalization of such encodings using *constraints*. We conclude the section with a discussion of SQL-derived temporal query languages.

1.3.1 Abstract Temporal Query Languages

The two different ways of linking time with a relational database (Definition 2) lead to two different temporal extensions of the relational calculus (first-order logic). The snapshot model gives rise to special *temporal connectives*. On the other hand, the timestamp model requires the introduction of explicit attributes and quantifiers for handling time. The first approach is especially appealing because it *encapsulates* all the interaction with the temporal domain inside the temporal connectives. In this way the manipulation of the temporal dimension is completely hidden from the user, as it is performed on implicit temporal attributes.

Historically, many different variants of temporal logic based on different sets of connectives have been developed [GHR94]. Some of the connectives, like $\Diamond$ (*"sometime in the future"*), $\Box$ (*"always in the future"*), or $\mathbf{until}$ are well-known and have been universally accepted. But in general any appropriate first-order formula in the language of the temporal domain (or, as we will see in Section 1.6, even a second-order one) can be used to define a temporal connective.

Definition 5 (First-order Temporal Connectives) *Let $k \geq 0$ and*

$$O ::= t_i < t_j \mid O \wedge O \mid \neg O \mid \exists t_i.O \mid X_i$$

the first-order language of T_P extended with propositional variables X_i, $0 < i \leq k$. We define a (k-ary) temporal connective *to be an O-formula with exactly one free variable t_0 and k free propositional variables $X_1, \ldots, X_k$. We assume that t_i is the only temporal variable free in the formula to be substituted for X_i.*

We define Ω to be a finite set of definitions of temporal connectives. Each definition is a pair consisting of a name $\omega(X_1, \ldots, X_k)$ and a (definitional) O-formula ω^* for the connective.

We call the variables t_i the *temporal contexts*: t_0 defines the outer temporal context of the connective that is made available to the surrounding formula; the variables $t_1, \ldots, t_k$ define the temporal contexts for the subformulas substituted for the propositional variables $X_1, \ldots, X_k$.

The above definition allows only *first-order* temporal connectives. This is sufficient to define the common temporal connectives $\mathbf{since}$, $\mathbf{until}$, and their derivatives.

Example 6 *The common binary temporal connectives are defined as follows:*

$$X_1 \ \mathbf{until}\ X_2 \ \stackrel{\triangle}{=}\ \exists t_2.t_0 < t_2 \wedge X_2 \wedge \forall t_1(t_0 < t_1 < t_2 \rightarrow X_1)$$
$$X_1 \ \mathbf{since}\ X_2 \ \stackrel{\triangle}{=}\ \exists t_2.t_0 > t_2 \wedge X_2 \wedge \forall t_1(t_0 > t_1 > t_2 \rightarrow X_1)$$

The commonly used unary temporal connectives, $\Diamond$ ("*sometime in the future*"), $\Box$ ("*always in the future*"), $\blacklozenge$ ("*sometime in the past*"), and $\blacksquare$ ("*always in the past*") are defined in terms of $\mathbf{since}$ and $\mathbf{until}$ as follows:

$$\Diamond X_1 \stackrel{\triangle}{=} \text{true } \mathbf{until}\ X_1 \qquad \Box X_1 \stackrel{\triangle}{=} \neg\Diamond\neg X_1$$
$$\blacklozenge X_1 \stackrel{\triangle}{=} \text{true } \mathbf{since}\ X_1 \qquad \blacksquare X_1 \stackrel{\triangle}{=} \neg\Diamond\neg X_1.$$

For a discrete linear order we also define the $\bigcirc$ (next) and $\bullet$ (previous) operators as

$$\bigcirc X_1 \stackrel{\triangle}{=} \exists t_1.t_1 = t_0 + 1 \wedge X_1 \qquad \bullet X_1 \stackrel{\triangle}{=} \exists t_1.t_1 + 1 = t_0 \wedge X_1.$$

Clearly, all the above connectives are definable in the first-order language of T_P (the successor $+1$ and the equality $=$ on the domain T_P are first-order definable in the theory of discrete linear order).

The connectives $\mathbf{since}$, $\blacklozenge$, $\blacksquare$, and $\bullet$ are called *past temporal connectives* (as they refer to the past) and $\mathbf{until}$, $\Diamond$, $\Box$, and $\bigcirc$ *future temporal connectives*.

Example 7 *Real-time temporal connectives, in which specific bounds on time are imposed, can also be defined in the same framework. For example, the connective* $\Diamond_{[k_1,k_2]}$ *is defined as:*

$$\Diamond_{[k_1,k_2]}X_1 \ \stackrel{\triangle}{=}\ \exists t_1.t_0 + k_1 \le t_1 \le t_0 + k_2 \wedge X_1.$$

Propositional temporal logic with real-time temporal connectives has been extensively used for the specification and verification of real-time systems [AH92; Koy89]. The first-order version of this logic has been applied to the specification of real-time integrity constraints [Cho95], and real-time logic programs [Brz93; Brz95]. Similarly to the $\bigcirc$ and $\bullet$ connectives the real time connectives (with integral bounds) are first-order definable in the theory of discrete linear order.

We discuss the use of a more expressive language, e.g., monadic second-order logic over the signature of T_P, to define a richer class of temporal connectives in Section 1.6.

First-order Temporal Logic. The modal query language—first-order temporal logic—is defined to be the original single-sorted first-order logic (relational calculus) extended with a finite set of temporal connectives.

Definition 8 (First-order Temporal Logic: syntax) *Let Ω be a* finite *set of (names of) temporal connectives.* First-order Temporal Logic (FOTL) L^{Ω} *over a schema ρ is defined as:*

$$F ::= r(x_{i_1}, \dots, x_{i_k}) \mid x_i = x_j \mid F \wedge F \mid \neg F \mid \omega(F_1, \dots, F_k) \mid \exists x.F$$

where $r \in \rho$ and $\omega \in \Omega$.

A standard linear-time temporal logic can be obtained from this definition using the temporal connectives from Example 6:

Example 9 *The standard FOTL language $L^{\{\mathbf{since},\mathbf{until}\}}$ is defined as*

$$F ::= r(x_{i_1}, \dots, x_{i_k}) \mid x_i = x_j \mid F \wedge F \mid \neg F \mid F_1 \ \mathbf{since}\ F_2 \mid F_1 \ \mathbf{until}\ F_2 \mid \exists x.F$$

where $\mathbf{since}$ *and* $\mathbf{until}$ *are the names for the connectives defined in Example 6.*

Example 10 *We show here how various temporal connectives are used to formulate the queries from Example 3. The query* "find all people who have worked for only one company" *is formulated as*

$$\blacklozenge\Diamond(\exists c.\mathrm{Works}(x,c) \wedge \neg\exists c'.(\Diamond\blacklozenge\mathrm{Works}(x,c') \wedge c' \neq c).$$

The query "find all people whose terminal degree is from MIT and the year of their graduation" *is formulated as*

$$\exists d.\exists m.\mathrm{Education}(x,d,m,MIT) \wedge \neg\Diamond\exists d'.\exists m'.\exists s'.\mathrm{Education}(x,d',m',s')$$

To list all UofT PhD's in CS who have been continuously employed by IBM since their graduation including the years of employment we use the query

$$\mathrm{Works}(x,IBM)\ \mathbf{since}\ \mathrm{Education}(x,PhD,CS,UofT).$$

Finally, the query "find all job-hoppers—people who never spent more than two years in one place" *is expressed as:*

$$\blacksquare\Box(\neg\exists c.\mathrm{Works}(x,c) \wedge \bigcirc\mathrm{Works}(x,c) \wedge \bigcirc\bigcirc\mathrm{Works}(x,c)).$$

The standard way of giving semantics to such a language is as follows.

Definition 11 (FOTL: semantics) *Let* DB *be a snapshot temporal database over* D*,* T_P*, and ρ, φ a formula of L^{Ω}, $t \in \mathrm{T}_P$, and θ a valuation. We define a relation* $\mathrm{DB},\theta,t \models \varphi$ *by induction on the structure of φ:*

$$
\begin{array}{ll}
\mathrm{DB},\theta,t \models r_j(x_{i_1},\dots,x_{i_k}) & \text{if } r_j \in \rho, (\theta(x_{i_1}),\dots,\theta(x_{i_k})) \in \mathbf{r}_j^{\mathrm{DB}(t)} \\
\mathrm{DB},\theta,t \models x_i = x_j & \text{if } \theta(x_i) = \theta(x_j) \\
\mathrm{DB},\theta,t \models \varphi \wedge \psi & \text{if } \mathrm{DB},\theta,t \models \varphi \text{ and } \mathrm{DB},\theta,t \models \psi \\
\mathrm{DB},\theta,t \models \neg\varphi & \text{if not } \mathrm{DB},\theta,t \models \varphi \\
\mathrm{DB},\theta,t \models \exists x_i.\varphi & \text{if there is } a \in \mathrm{D} \text{ such that } \mathrm{DB},\theta[x_i \mapsto a],t \models \varphi \\
\mathrm{DB},\theta,t \models \omega(F_1,\dots,F_k) & \text{if } \mathrm{T}_P,[t_0 \mapsto t] \models \omega^* \text{ where} \\
 & \quad T_P,\delta \models X_i \text{ is interpreted as } \mathrm{DB},\theta,\delta(t_i) \models F_i
\end{array}
$$

where $\mathbf{r}_i^{\mathrm{DB}(t)}$ *is the interpretation of the relation symbol* r_i *in* DB *at time* t.

We assume the rigid interpretation of constants (it does not change over time). The answer to a query φ over DB *is the set of tuples* $\varphi(\mathrm{DB}) := \{(t, \theta_{|FV(\varphi)}) : \mathrm{DB}, \theta, t \models \varphi\}$ *where* $\theta_{|FV(\varphi)}$ *is the restriction of* θ *to the free variables of* φ.

Example 12 *The above definition applied to the standard language* $L^{\{\mathbf{since},\mathbf{until}\}}$ *gives the usual semantics of the* $\mathbf{since}$ *and* $\mathbf{until}$ *connectives:*

$$D, \theta, i \models \varphi \ \mathbf{until}\ \psi \text{ if } \exists j.j > i \wedge D, \theta, j \models \psi \wedge \forall k.j > k > i \rightarrow D, \theta, k \models \varphi$$

Two-sorted first-order logic. The second natural extension of the relational calculus to a temporal query language is based on explicit variables and quantification over the temporal domain T_P. It is just the two-sorted version of first-order logic (2-FOL) over D and T_P, with the limitation that the relations can have only one temporal argument [BTK91].

Definition 13 (2-FOL: syntax) *The two-sorted first-order language* L^P *over a database schema* ρ *is defined by:*

$$M ::= R(t_i, x_{i_1}, \ldots, x_{i_k}) \mid t_i < t_j \mid x_i = x_j \mid M \wedge M \mid \neg M \mid \exists x_i.M \mid \exists t_i.M$$

where R *is the temporal extension of* r *for* $r \in \rho$.

Similarly to FOTL we can use 2-FOL to formulate temporal queries:

Example 14 *The query* "find all people who have worked for only one company" *is formulated in 2-FOL as*

$$\exists c, t.\mathrm{Works}(t, x, c) \wedge \neg\exists c', t'.(\mathrm{Works}(t', x, c') \wedge c' \neq c).$$

The semantics for this language is defined in the standard way, similarly to the semantics of relational calculus [AHV95].

Definition 15 (2-FOL: semantics) *Let* DB *be a timestamp temporal database over* D*,* T_P*, and* ρ*,* φ *a formula in* L^P*, and* θ *a two-sorted valuation. We define the satisfaction relation* $\mathrm{DB}, \theta \models \varphi$ *as follows:*

$$\begin{array}{ll}
\mathrm{DB}, \theta \models R_j(t_i, x_{i_1}, \ldots, x_{i_k}) & \text{if } R_j \in \rho, (\theta(t_i), \theta(x_{i_1}), \ldots, \theta(x_{i_k})) \in \mathbf{R}_j^{\mathrm{DB}} \\
\mathrm{DB}, \theta \models t_i < t_j & \text{if } \theta(t_i) < \theta(t_j) \\
\mathrm{DB}, \theta \models x_i = x_j & \text{if } \theta(x_i) = \theta(x_j) \\
\mathrm{DB}, \theta \models \varphi \wedge \psi & \text{if } \mathrm{DB}, \theta \models \varphi \text{ and } \mathrm{DB}, \theta \models \psi \\
\mathrm{DB}, \theta \models \neg\varphi & \text{if not } \mathrm{DB}, \theta \models \varphi \\
\mathrm{DB}, \theta \models \exists t_i.\varphi & \text{if there is } s \in \mathrm{T}_P \text{ such that } \mathrm{DB}, \theta[t_i \mapsto s] \models \varphi \\
\mathrm{DB}, \theta \models \exists x_i.\varphi & \text{if there is } a \in \mathrm{D} \text{ such that } \mathrm{DB}, \theta[x_i \mapsto a] \models \varphi
\end{array}$$

where $\mathbf{R}_j^{\mathrm{DB}}$ *is the interpretation of the relation symbol* R_j *in the database* DB.

A L^P *query is a* L^P *formula with exactly one free temporal variable.*

The answer to a L^P query φ over DB *is the set* $\varphi(\mathrm{DB}) := \{\theta_{|FV(\varphi)} : \mathrm{DB}, \theta \models \varphi\}$ *where* $\theta_{|FV(\varphi)}$ *is the restriction of the valuation* θ *to free variables of* φ.

The restriction to a single temporal attribute in the signature of queries guarantees closure over the universe of single-dimensional temporal relations.

1.3.2 Expressive Power

In the remainder of this section we compare the expressive power of FOTL and 2-FOL. First we define a mapping $\mathrm{Embed} : L^\Omega \to L^P$ to show that the L^Ω formulas can be expressed in the L^P language:

Definition 16 (Translation) *Let* Embed *be a mapping of* L^Ω *formulas to* L^P *formulas defined as follows:*

$$\begin{array}{lcl}
\mathrm{Embed}(r_i(x_1,\ldots,x_{v_i})) & = & R_i(t_0,x_1,\ldots,x_{v_i}) \\
\mathrm{Embed}(x_i = x_j) & = & x_i = x_j \\
\mathrm{Embed}(F_1 \wedge F_2) & = & \mathrm{Embed}(F_1) \wedge \mathrm{Embed}(F_2) \\
\mathrm{Embed}(\neg F) & = & \neg\,\mathrm{Embed}(F) \\
\mathrm{Embed}(\exists x.F) & = & \exists x.\,\mathrm{Embed}(F) \\
\mathrm{Embed}(\omega(F_1,\ldots,F_k)) & = & \omega^*(\mathrm{Embed}(F_1)[t_0/t_1],\ldots,\mathrm{Embed}(F_k)[t_0/t_k])
\end{array}$$

where $\omega(X_1,\ldots,X_k)$ *is the name of* ω^* *in* Ω *and* $F[t_0/t_i]$ *is a substitution of* t_i *for* t_0 *in* F.

We know that we can freely move between snapshot and timestamp representations (see Definition 2). Definition 16 allows us to translate queries in L^Ω to queries in L^P while preserving their semantics.

Theorem 17 *Let* DB_1 *be a snapshot temporal database and* DB_2 *an equivalent timestamp database. Then* $\mathrm{DB}_1, \theta, s \models \varphi \iff \mathrm{DB}_2, \theta[t_0 \mapsto s] \models \mathrm{Embed}(\varphi)$ *for all* $\varphi \in L^\Omega$.

Therefore Definition 16 can also be used to define the semantics of L^Ω queries over timestamp temporal databases.

Another consequence of Definition 16 and Theorem 17 is that L^P is at least as expressive as L^Ω (denoted by $L^\Omega \sqsubseteq L^P$). What is the relationship in the other direction?

Expressive Incompleteness of First-order Temporal Logic. While both snapshot and timestamp temporal models can be used to represent temporal databases equivalently, we show that the corresponding query languages *cannot* express the same queries. This is a major difference from the propositional case where linear-time temporal logic has the same expressive power as the monadic first-order logic over linear orders. The latter result was established by Kamp for complete linear orders, extended by Stavi for all linear orders, and later reproved several times using various proof techniques [Kam68; Sta79; GHR94; IK89]. On the other hand Kamp also proved the following separation result in a first-order setting $L^{\{\mathbf{since},\mathbf{until}\}} \sqsubset L^{\{\mathbf{since},\mathbf{until},\mathbf{now}\}} \sqsubseteq L^P$ for dense linearly ordered time ($\sqsubset$ denotes the "strictly weaker than" relationship of languages). Thus on general structures $L^{\{\mathbf{since},\mathbf{until}\}}$ is strictly weaker than L^P [Kam71]. However, the proof of this fact uses structures that can not be modeled as abstract temporal databases because they are infinite in both the data and temporal dimensions. Also, the proof technique does not consider arbitrary temporal connectives. Moreover, it is not clear if this proof technique can be adapted to discrete linear orders [Kam71].

Thus there was a hope that the gap in expressive power could be bridged by introducing additional connectives and/or by restricting the structures to abstract temporal databases that are finite at every moment of time. However, two recent independent results show that this is not the case, and that for all practical purposes $L^\Omega \sqsubset L^P$:

Theorem 18 (AHVdB96) $L^{\{\mathbf{since},\mathbf{until}\}} \sqsubset L^P$ *over the class of finite timestamp temporal databases.*

Theorem 19 (TN96) *Let Ω be an arbitrary finite set of first-order temporal connectives. Then $L^\Omega \sqsubset L^P$ for the class of timestamp temporal databases over a dense linear order.*

In both cases the language of temporal logic L^Ω is shown not to be able to express the query "are there two distinct time instants at which a unary relation R contains exactly the same values?" On the other hand, this query can be easily expressed in L^P using the formula

$$\exists t_1, t_2 . t_1 < t_2 \wedge \forall x . R(t_1, x) \iff R(t_2, x).$$

This result has several *very unpleasant* consequences: A single-dimensional first-order complete temporal query language can not be *subquery closed*. This means that in general we can not define all queries to be combinations of simpler single-dimensional queries.

This fact also prevents us from decomposing large queries into views (virtual relations defined by queries). An even more serious problem is that there is no relational algebra defined over the universe of single-dimensional temporal relations that is able to express all first-order temporal queries.

Temporal Relational Algebra. Similarly to relational algebra, a *Temporal Relational Algebra* is a (finite) set of (first-order definable) operators defined on the universe of single-dimensional temporal relations.

Example 20 (TC90) *A temporal relational algebra (TRA) is a set of algebraic operators $\pi_V, \sigma_F, \bowtie, \cup, -, \mathcal{S}, \mathcal{U}$ over the universe of single dimensional temporal relations defined by:*

$$\begin{array}{ll}
\pi_V(\mathbf{R}) & = \{t, \theta_{|V} : \mathrm{DB}, \theta, t \models R\} \\
\sigma_F(\mathbf{R}) & = \{t, \theta_{|FV(R)} : \mathrm{DB}, \theta, t \models R \wedge F\} \\
\mathbf{R} \bowtie \mathbf{S} & = \{t, \theta_{|FV(R) \cup FV(S)} : \mathrm{DB}, \theta, t \models R \wedge S\} \\
\mathbf{R} \cup \mathbf{S} & = \{t, \theta_{|FV(R) \cup FV(S)} : \mathrm{DB}, \theta, t \models R \vee S\} \\
\mathbf{R} - \mathbf{S} & = \{t, \theta_{|FV(R) \cup FV(S)} : \mathrm{DB}, \theta, t \models R \wedge \neg S\} \\
\mathcal{S}(\mathbf{R}, \mathbf{S}) & = \{t, \theta_{|FV(R) \cup FV(S)} : \mathrm{DB}, \theta, t \models R \textbf{ since } S\} \\
\mathcal{U}(\mathbf{R}, \mathbf{S}) & = \{t, \theta_{|FV(R) \cup FV(S)} : \mathrm{DB}, \theta, t \models R \textbf{ until } S\}
\end{array}$$

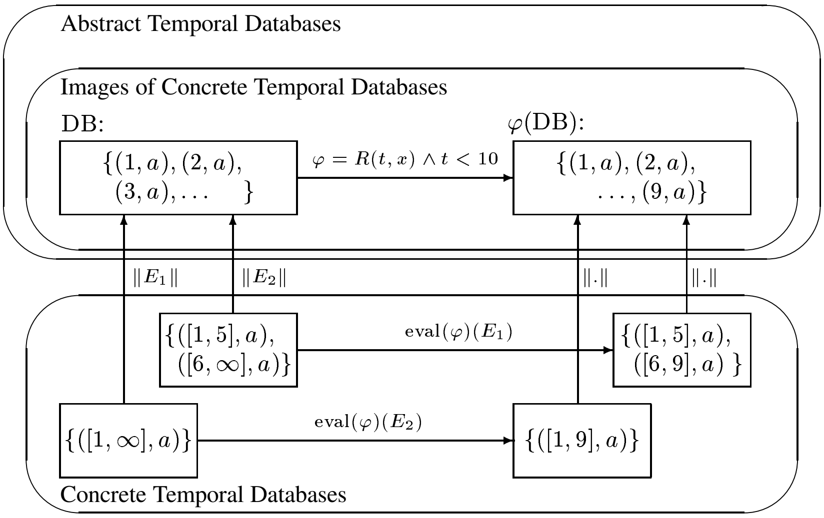
where $\mathbf{R}$ *and* $\mathbf{S}$ *are the interpretations of the symbols* R *and* S *in the database* DB.

The above definition allows us to translate (domain-independent) formulas in $L^{\{\mathbf{since},\mathbf{until}\}}$ to TRA. However, this is also the reason why TRA with arbitrary finite set of first-order definable operators can not express all first-order queries (an immediate consequence of Theorems 18 and 19). This fact causes major problems when implementing query processors for temporal query languages as the common (and efficient) implementations inherently depend on the equivalence of relational algebra and calculus to be able to execute all queries [AHV95; Ull89].

1.3.3 Space-efficient Encoding of Temporal Databases

While abstract temporal databases provide a natural semantic domain for interpreting temporal queries, they are not immediately suitable for the implementation, as they are possibly infinite (e.g., when the database contains a fact holding for infinitely many time instants). Even for finite abstract temporal databases a direct representation may be extremely space inefficient: tuples are often associated with a large number of time instants (e.g., a validity interval). In addition, changes in the *granularity* of time may affect the size of the stored relations.

Our goal in this section is to develop a compact *encoding* for (a subclass of) abstract temporal databases that makes it possible to efficiently store such databases in finite space. The most common approach to such an encoding is to use *intervals* as codes for convex 1-dimensional sets of time instants. The choice of this representation is based on the following (empirical) observation: Sets of time instants describing validity of a particular fact in the real world can be often described by an interval or a finite union of intervals. We briefly discuss other encodings at the end of this section. For simplicity from now on we assume a discrete integer-like structure of time. However, dense time can also be accommodated by introducing open and half-open intervals. All the results in this section carry over to the latter setting.



E_1 and E_2 are two concrete temporal databases that represent the same abstract temporal database DB.

Figure 1.2 Abstract and Concrete Timestamp Temporal Databases

Definition 21 (Interval-based Domain $\mathrm{T_I}$) *Let $\mathrm{T}_P = (T, <)$ be a discrete linearly ordered point-based temporal domain. We define the set*

$$I(T) = \{(a, b) : a \leq b, a \in T \cup \{-\infty\}, b \in T \cup \{\infty\}\}$$

where $<$ is the order over T_P extended with $\{(-\infty, a), (a, \infty), (-\infty, \infty) : a \in T\}$ (similarly for $\leq$). We denote the elements of $I(T)$ by $[a, b]$ (the usual notation for intervals). We also define four relations on the elements of $I(T)$:

$$\begin{array}{ll} ([a, b] <_{--} [a', b']) \iff a < a' & ([a, b] <_{+-} [a', b']) \iff b < a' \\ ([a, b] <_{-+} [a', b']) \iff a < b' & ([a, b] <_{++} [a', b']) \iff b < b' \end{array}$$

for $[a, b], [a', b'] \in I(T)$. The structure $\mathrm{T}_I = (I(T), <_{--}, <_{+-}, <_{-+}, <_{++})$ is the Interval-based Temporal Domain (corresponding to T_P).

A concrete (timestamp) temporal database is defined analogously to the abstract (timestamp) temporal database. The only difference is that the temporal attributes range over intervals (T_I) rather than over the individual time instants (T_P).

Definition 22 (Concrete Temporal Database) *A* concrete temporal database *is a finite first-order structure* $\mathrm{D} \cup \mathrm{T}_I \cup \{\mathbf{R}_1 \ldots, \mathbf{R}_k\}$*, where* $\mathbf{R}_i$ *are the* concrete temporal relations *which are finite instances of* R_i *over* D *and* T_I*.*

Clearly the values of the interval attributes can be encoded as pairs of their endpoints which are elements of $T \cup \{-\infty, \infty\}$. However, it is important to understand that both T_P and T_I model *the same* structure of time instants, a single-dimensional linearly ordered time line. This requirement is the crucial difference between the use of intervals in temporal databases and in various interval-based logics (cf. Section 1.5). The meaning of concrete temporal databases is defined by a mapping to the class of abstract temporal databases.

Definition 23 (Semantic Mapping $\|.\|$) *Let* DB_1 *be an abstract temporal database and* DB_2 *a concrete temporal database over the same schema* ρ*. We say that* DB_2 encodes DB_1 *if*

$$\mathbf{R}_i^{\mathrm{DB}_1}(t, \mathbf{x}) \iff \exists I \in \mathrm{T}_I . \mathbf{R}_i^{\mathrm{DB}_2}(I, \mathbf{x}) \wedge t \in I$$

for all r_i *in* ρ*,* $t \in \mathrm{T}_P$*, and* $\mathbf{x} \in D^{\mathrm{arity}(r_i)}$*, where* $\mathbf{R}_i^{\mathrm{DB}}$ *is the interpretation of the relation symbol* R_i *in the database* DB*. This correspondence defines a map* $\|.\|$ *from the class of concrete temporal databases to the class of abstract temporal databases as an extension of the mapping of the relations in* DB_2 *to the relations in* DB_1*.*

Note that $\|.\|$ is neither injective nor onto. Therefore there is no unique *canonical* concrete temporal database that encodes a given abstract temporal database (cf. Figure 1.2). If only a single temporal dimension is allowed however, we can define a *canonical* form for concrete temporal relations using *coalescing*: A single-dimensional temporal relation is *coalesced* if every fact is associated only with maximal non-overlapping intervals. A concrete temporal database is *coalesced* if all the user-defined relations are coalesced. Unfortunately, such a canonical normal form does *not* generalize to higher dimensions and Theorems 18 and 19 show that we can not restrict our attention to the single-dimensional case.

1.3.4 Concrete Temporal Query Languages

The simplest query language over concrete temporal databases is the two-sorted first-order logic where variables and quantifiers of the temporal sort range over the domain T_I rather than T_P.

Definition 24 (Interval-based Language $\mathbf{L}^{\mathbf{I}}$) *Let* ρ *be a database schema and*

$$L ::= R_i(I, \mathbf{x}) \mid L \wedge L \mid \neg L \mid \exists x.L \mid \exists I.L \mid x_1 = x_2 \mid I_1^* < I_2^*$$

where R_i *is the temporal extension of* $r_i \in \rho$ *and* $I^* \in \{I^+, I^-\}$*.*

The language L^I uses $I_1^* < I_2^*$ instead of the symbols $<_{--}, <_{+-}, <_{-+}, <_{++}$ from the *actual* structure of T_I. However, it is easy to see that, e.g., $I^- < J^-$ can be expressed as $I <_{--} J$, etc., and the new notation is thus merely syntactic sugar. We could also equivalently use Allen's algebra operators [All83]. The resulting language is equivalent to L^I.

We assume the usual Tarskian semantics for formulas in L^I. Therefore L^I is fairly easy to implement using standard relational techniques. However, it is crucial to understand that this semantics of L^I is *not* point-based—the elements of T_I correspond to points in the two-dimensional plane (cf. Section 1.5). Thus L^I can not be immediately used as a query language over interval-based encodings of *point-based* abstract temporal databases because, among other things, it can easily express representation-dependent queries. Consider the following example:

Example 25 *Let* $\mathrm{DB}_1, \mathrm{DB}_2$ *be two concrete temporal databases over the schema* $(r(x))$ *defined by* $\mathbf{R}^{\mathrm{DB}_1} = \{([0,2], a), ([1,3], a)\}$ and $\mathbf{R}^{\mathrm{DB}_2} = \{([0,3], a)\}$. *Then the formula* $\exists I, J.\exists x(R(I,x) \wedge R(J,x) \wedge I \neq J)$ *is true in* DB_1 *but false in* DB_2.

This observation leads to the following definition:

Definition 26 ($\|.\|$-generic Queries) *Let* $\|.\|$ *be the semantics mapping and* $\varphi \in L^I$. *Then we say that* φ is $\|.\|$-generic *if* $\|\mathrm{DB}_1\| = \|\mathrm{DB}_2\|$ *implies* $\|\varphi(\mathrm{DB}_1)\| = \|\varphi(\mathrm{DB}_2)\|$ *for all concrete temporal databases* $\mathrm{DB}_1, \mathrm{DB}_2$.

Most interval-based query languages (e.g., TQuel or SQL/Temporal; cf. Section 1.3.6) are directly based on the language L^I (or some of its variants). This choice inherently leads to the possibility of expressing non $\|.\|$-generic queries.

1.3.5 Evaluation of Abstract Query Languages using Compilation

A desirable solution is to use one of the abstract query languages for querying concrete temporal databases. However, the semantics of these languages is defined over the class of abstract temporal databases (and we can not simply apply the queries to the images of concrete temporal databases under $\|.\|$, as this would completely defy the purpose of using the concrete encodings and we would have to face the possibility of handling infinite relations). Thus we need to evaluate abstract queries directly over the concrete encodings. This goal is achieved using *compilation techniques* that transform abstract queries to formulas in L^I while *preserving meaning* under $\|.\|$:

Theorem 27 (Tom96) *There is a (recursive) mapping* $F : L^P \to L^I$ *such that* $\varphi(\|\mathrm{DB}\|) = \|F(\varphi)(\mathrm{DB})\|$ *for all* $\varphi \in L^P$ *and all concrete temporal databases* DB.

Moreover we can show that when using the interval-based encoding L^P can express all $\|.\|$-generic queries in L^I:

Theorem 28 (Tom96) *For every* $\|.\|$*-generic* $\varphi \in L^I$ *there is* $\psi \in L^P$ *such that* $\|\varphi(\mathrm{DB})\| = \psi(\|\mathrm{DB}\|)$ *for all concrete temporal databases* DB.

The mapping from Theorem 27 can be also used for L^Ω by composing it with the Embed map from Definition 16. However, we may ask if there is a more direct way from L^Ω to L^I? The following theorem gives a direct mapping of L^Ω to ATSQL (which is essentially a SQL version of L^I; cf. Section 1.3.6):

Theorem 29 (BCST96) *There is a (recursive) mapping* $G : L^\Omega \to L^I$ *such that* $\varphi(\|\mathrm{DB}\|) = \|G(\varphi)(\mathrm{DB})\|$ *for* $\varphi \in L^\Omega$ *and* DB *an arbitrary* coalesced *concrete temporal database.*

This mapping is considerably simpler than the indirect way through L^P. However, we pay the price for simplicity by having to maintain coalesced temporal relations, including all intermediate results during the bottom-up evaluation of the query. Note that the use of coalescing is possible due to the inherent single-dimensionality of L^Ω.

The mappings defined in Theorems 27 and 29 bring up an interesting point: what are the images of the temporal connectives themselves? It turns out that the results of such translations can be considered to be the equivalents of the original connectives that operate on concrete temporal relations:

Example 30 *Let* $\mathbf{UNTIL} = \lambda r.\lambda s.F \circ \mathrm{Embed}(r \ \mathbf{until} \ s)$ *and* ϕ *and* ψ *two queries in* L^Ω*. Then* $(\phi \ \mathbf{until} \ \psi)(\|\mathrm{DB}\|) = \|(F \circ \mathrm{Embed}(\phi) \ \mathbf{UNTIL} \ F \circ \mathrm{Embed}(\psi))(\mathrm{DB})\|$ *for all concrete temporal databases* DB.

A similar trick can be used to define the remaining temporal connectives. For coalesced databases we can use G in place of $F \circ \mathrm{Embed}$. This definition can be used to define an algebra over concrete relations that preserves the $\|.\|$ mapping and is thus suitable for implementing L^Ω. This algebra can serve as the *concrete* counterpart of the temporal relational algebra introduced in Example 20 [TC90].

Constraint Encoding. A careful analysis of Definition 21 reveals that the intervals are essentially quantifier-free formulas in $Th(<_{Lin})$ with exactly one free variable. This idea can be generalized to more general classes of constraints [KKR95]: Let (T, σ) be a point-based temporal domain with the signature σ. Then we can define the set of formulas $C_\sigma = \{\phi(t) : \phi \in L_\sigma \wedge FV(\phi) = \{t\}\}$ where L_σ is the set of finite conjunctions of atomic formulas in the language of σ. The set C_σ can serve as the basis of the temporal domain for a class of concrete temporal databases, similarly to intervals in Definition 21. An example of an alternative encoding is the use of *periodic constraints* [KSW95], or *linear arithmetic constraints* [KKR95]. It is important to note that the use of a different constraint theory as the basis of the encoding results in a *different* class of concrete temporal databases, often incomparable with the class of concrete temporal databases based on intervals.

1.3.6 SQL and Derived Temporal Query Languages

Up to this point we have only discussed temporal query languages based on logic. In this section, we focus on the proposals for temporal extensions of more practical query languages, especially SQL [ISO92]. When designing such an extension we need to overcome several obstacles:

1. The semantics of SQL and other practical languages is commonly based on a bag (duplicate) semantics rather than on a set (Tarskian) semantics. Therefore we need to design our extension to be consistent with the semantics of the language we started with. This also means that we need to deal with various non-first-order features of the original language, e.g., with aggregation (the ability to count the number of tuples in a relation or to compute the sum of values in an attribute of the relation over all tuples).

2. We need to design the extension in a way that consistently supports the chosen model of time. This point is often not emphasized enough and many of the proposals drift from the intended model of time in order to accommodate extra features. However, such design decisions lead to substantial problems in the long run, especially when a precise semantics of the extension has to be spelled out (this is one of the reasons why only informal semantics exist for many of these languages).

3. To obtain a feasible solution we need to use a compact encoding of temporal databases introduced in Section 1.3.3. Therefore we need an *efficient* query evaluation procedure the chosen class of *concrete databases*.

We would like to point out that vast majority of practical temporal query languages assume a *point-based model of time* (i.e., the truth of facts is associated with single time instants rather than with sets of time instants) [Cho94]. Unfortunately (and also in most cases) the *syntax* is based on the syntax of L^I or some of its variants, e.g., languages that use Allen's interval algebra operators [All83]. This discrepancy leads to a tension between the syntactic constructs used in the language and the intended semantics of queries. While we focus mostly on temporal extensions of SQL, our observations are general enough to apply to temporal extensions of other query languages, e.g., TQuel [Sno87].

Example 31 *We demonstrate the differences between the approaches using the following query: List all persons who have been unemployed between jobs. This query can be easily formulated in temporal logic as follows:*

$$\blacklozenge \exists y.\mathrm{Works}(x,y) \wedge \neg\exists y.\mathrm{Works}(x,y) \wedge \Diamond\exists y.\mathrm{Works}(x,y).$$

The query could also be equivalently expressed using future (or past) temporal connectives only, see Example 36.

The temporal extensions of SQL can be divided into two major groups depending on the *syntactic constructs* added to support temporal queries: the languages based on L^I and those based on L^P. It is also possible to base such an extension on L^Ω but this possibility has been little explored.

Languages based on $\mathbf{L^I}$. This group contains the majority of current proposals, in particular the SQL/Temporal proposal to the ANSI/ISO SQL standardization group [SBJS96] and ATSQL [SJB95], the *applied* version of TSQL2 [Sno93] . Both these languages are directly based on L^I with Allen's algebra operators dressed in SQL syntax and using bag (duplicate) semantics.

Let us try to formulate the query from Example 31 in such a language, e.g., TSQL2 or its successor, SQL/Temporal. The solution which most people come up with is the query below (we use an intuitive and simplified syntax to make our point; for full details on the syntax of SQL/Temporal see [SJB95; SBJS96]).

Example 32 *The query from Example 31 in SQL/Temporal:*

```
select r1.Name
from   Works r1, Works r2
where  r1.Name = r2.Name
  and  r1.Year before r2.Year
```

Note that the `Year` *attributes range over intervals and the* `before` *relationship denotes the before relationship between two intervals. For a similar example in TQuel see [Cho94].*

Strangely enough, this query accesses the relation Works only twice while the original query in Example 31 needs to access the relation three times. This is often considered to be a feature of the L^I-based proposals and is attributed to the use of interval-based temporal attributes. It is also very appealing due to the savings in the query evaluation cost. However, a closer scrutiny reveals that the above SQL/Temporal query is incorrect. Indeed, it also returns names of all people who held at least three consecutive jobs even if there was no gap between the jobs. This result is consistent with the *two-dimensional* interval-based semantics of L^I. Similarly we can show many innocent-looking queries to be non-generic (in sense of Definition 26) and therefore necessarily incorrect with respect to their intended meaning. On the other hand the access to the interval endpoints (the *nonsequenced semantics* [SBJS96]) is essential to write non-trivial temporal queries in SQL/Temporal.

There are two principal approaches that try to avoid the incorrect and unexpected behavior by modifying the semantics of the above languages.

Coalescing. The first (and historically oldest) approach is based on *coalescing*: the assumption that timestamps are represented by maximal non-overlapping intervals (see Section 1.3.3). This is also a commonly made assumption when queries like the one in Example 32 are formulated. Coalescing attempts to produce a *normal form* of temporal relations over which the semantics of queries could be (uniquely) defined. The formal justification of this approach lies in realization that the intended semantics of the language is point-based and therefore we can evaluate queries over any of the $||.||$-equivalent temporal databases (one of which is the coalesced one). For a detailed discussion of coalescing in temporal databases see [BSS96].

The most prominent representatives of this approach are TQuel [Sno87; Sno93], and TSQL2 [Sno95; SAA$^+$94]. However:

- Coalescing does not solve the problem with the above query: if a person works for three different companies she is still in the answer to the query. The query is correct if the Works relation is coalesced *after* projecting out the attribute $\mathrm{Company}$. This is not done in the (informal) semantics of TQuel or TSQL2. It also means that the performance gain attributed to the use of interval-valued attributes does not exist as we need to re-coalesce temporal relations on the fly.

- While coalescing preserves $||.||$-equivalence in the set-based semantics it is incompatible with the use of duplicate semantics as it inherently removes duplication. This is the main reason why the newer proposals, e.g., SQL/Temporal or ATSQL, do not use coalescing in order to preserve compatibility with SQL's duplicate semantics.

However, the most serious problem with coalescing-based approaches is exposed by Theorems 18 and 19: The theorems show that we cannot evaluate all first-order queries using only one temporal dimension. This result is fatal to the coalescing-based approaches as there is no unique coalescing for temporal dimension higher than one.

Folding and Unfolding. The second approach is based on two additional operations: *fold* and *unfold* [Lor93]. These two operations allow us to explicitly convert a concrete temporal relation with interval-based timestamps to a temporal relation with point-based timestamps. The query from Example 31 can be correctly formulated using fold/unfold, as it now becomes possible to refer to unfolded temporal relations (with duplicates). However, the use of these operations is prohibitively expensive.

Example 33 *Consider a temporal relation R containing a single tuple $(a, [-2^n, 2^n])$ for some $n > 0$. Clearly, this relation can be stored in $2n + |a|$ bits. However, unfolding this relation gives us $\{(a, i) : -2^n \leq i \leq 2^n\}$. This relation needs space $2^n \cdot |a|$ which is* exponential *in the size of the original relation R.*

Such a cost clearly disqualifies the above approach as a basis for a practical temporal query language. In addition, unfolding of a concrete relation that represents an infinite abstract relation, e.g., the relation $R = \{([0, \infty])\}$, is not possible.

Languages based on $\mathbf{L^P}$. While query languages based on L^P were often considered to be inherently inefficient, recent results (especially Theorem 27 [Tom96]) allow us to define a *point-based* extension of SQL that can be efficiently evaluated over concrete interval-based temporal databases. The proposed language, SQL/TP, is a clean temporal extension of SQL [Tom97]:

- The syntax and semantics of SQL/TP are defined as a natural extension of SQL with an additional data type based on the point-based temporal domain T_P.
- On the other hand, the use of Theorem 27 avoids the problems in Example 33: the result of the F map is an ordinary query in L^I (or SQL). Therefore it can be efficiently evaluated over concrete temporal databases based on interval encoding of timestamps (like any other SQL query).

The SQL/TP proposal also includes a definition of meaningful duplicate semantics and aggregation operations that are compatible with standard SQL [Tom97]. The query from Example 31 can be formulated in SQL/TP in the expected way:

```
select r1.Name
from   Works r1, Works r2
where  r1.Name = r2.Name
  and  r1.Year < r2.Year
  and  not exists (
    select *
    from   Works r3
    where  r3.Name = r1.Name
      and  r1.Year < r3.Year
      and  r3.Year < r2.Year )
```

It is easy to see that the above formulation is is very similar to the declarative formulation of the query in the language L^P or in temporal logic.

Languages based on $\mathbf{L}^\Omega$. Another possible temporal extension of SQL can be based on the language L^Ω for some finite set of temporal connectives Ω. The temporal connectives can be introduced in the language similarly to set operations, e.g., the `union` operation.

Example 34 (SQL/$\{\mathbf{since}, \mathbf{until}\}$) *The extended language is defined as follows. Every SQL query is also a SQL/$\{\mathbf{since}, \mathbf{until}\}$ query. In addition if* `Q1` *and* `Q2` *are two queries (fullselects) then*

```
Q1 since Q2          Q1 until Q2
```

are also SQL/$\{\mathbf{since}, \mathbf{until}\}$ queries. The semantics of this language is based on a natural extension of Definition 11.

We can use Theorem 29 to evaluate queries in this language efficiently over coalesced interval-encoded concrete temporal databases [BCST96]. Note that in this case all temporal relations have only one temporal attribute and therefore we can use coalescing.

Alternatively we can compose the mappings defined in Definition 16 with Theorem 27 to obtain a query evaluation procedure for L^{Ω}. This time we do not have to enforce coalescing of concrete temporal relations, as Theorem 27 allows to evaluate queries over *arbitrary* concrete temporal databases (based on interval encoding).

1.4 TEMPORAL INTEGRITY CONSTRAINTS

In this section we show how temporal logic can be used as a language for specifying *temporal integrity constraints* in relational databases. We have shown in Section 1.2 how the notion of a temporal database occurs naturally in the context of relational database histories. Here, we discuss different notions of temporal constraint satisfaction and various approaches to the problem of temporal integrity maintenance. Then we summarize the existing results about the computational complexity of checking temporal integrity constraints and sketch an efficient evaluation method for a class of such constraints.

1.4.1 Notions of Constraint Satisfaction

Definition 35 (Constraint Satisfaction) *Given an abstract temporal query language $\mathcal{L}$, a closed formula $C \in \mathcal{L}$, and a history $H = (H_0, \ldots, H_n, H_{n+1}, \ldots)$, we say that C is satisfied by H if $H, i \models C$ for every $i \geq 0$.*

This definition assumes some standard notion of temporal formula satisfaction, e.g., Definition 11.

Example 36 *Consider the following constraint C_0:* "a student cannot graduate with honors if he retook any course". *This constraint can be expressed as a temporal logic formula with future connectives:*

$$\neg\exists x.\exists y.\mathit{takes}(x,y) \wedge \Diamond(\mathit{takes}(x,y) \wedge \Diamond \mathit{honors}(x))$$

or as a formula with past connectives:

$$\neg\exists x.\exists y.\mathit{honors}(x) \wedge \blacklozenge(\mathit{takes}(x,y) \wedge \blacklozenge \mathit{takes}(x,y))$$

Temporal integrity constraints are imposed on the *current history* of a database, i.e., the sequence of states up to the current one. Such a history is of course finite. However, the satisfaction of a temporal integrity constraint is defined with respect to an infinite history representing a possible future evolution of the database. The notion of *potential constraint satisfaction* (called also *potential validity* [LS87]) reconciles those two views.

Definition 37 (Potential Constraint Satisfaction) *Given a closed formula $C \in \mathcal{L}$, C is* potentially satisfied *at instant t if the current finite history $(H_0, H_1, \ldots, H_t)$ can be extended to an infinite history $H = (H_0, H_1, \ldots, H_t, \ldots)$ that satisfies C.*

In other words, a constraint is potentially satisfied after an update if the history ending in the state resulting from the update has an (infinite) extension to a model of the constraint. Notice that from the above definition it follows that if a constraint is potentially satisfied at a given instant, it was also potentially satisfied at all the earlier instants.

In the database context, considering infinite sequences of states means that the database is infinitely updatable. This seems like a desirable characteristic, even though in real applications databases have only a finite lifetime. Similarly, in concurrent systems one often studies infinite behaviors, although every practical program runs only for a finite time. In both cases infinity provides a convenient mathematical abstraction.

1.4.2 Temporal Integrity Maintenance

We discuss now three different scenarios for temporal integrity maintenance: constraint checking, temporal triggers, and transaction validation. In this context, we also review the well-known distinction between safety and liveness properties [Pnu86].

Constraint Checking [Cho95; CT95; GL96; HS91; LS87]. The assumption here is that updates are arbitrary mappings producing finite states. An update is *committed* if in the resulting history all the constraints are potentially satisfied and *aborted* otherwise.

Temporal Triggers [SW95]. Again, the basic assumption is that updates are arbitrary.

Definition 38 (Trigger Firing) *Let $(H_0, H_1, \ldots, H_t)$ be a finite history and T a Condition-Action trigger of the form "***if** C **then** A*"[WC96]. Then T* fires *at instant t for a (ground) substitution θ to the free variables of C if $\neg C\theta$ (the result of applying the substitution θ to $\neg C$) is not potentially satisfied at t. The action executed is $A\theta$.*

Intuitively, a trigger fires after an update if no extension of the history ending in the state resulting from the update can make the trigger condition false. So we can see that the notion of trigger firing is dual to potential constraint satisfaction. That corresponds to the intuition that integrity checking triggers should fire when the integrity is violated. However, it is up to the trigger designer to guarantee that the integrity be restored by appropriately programming the action part of the trigger. There are currently no tools, formal or otherwise, that could help him in this task. The work on automatically generating triggers from constraint specifications [CW90] and declarative specification of constraint maintenance [BCP94] can perhaps be generalized to temporal integrity constraints and triggers.

Transaction Validation [dCCF82; Kun85; Lip90]. Here the assumption is that the database will only be updated by a fixed set of transactions. The transactions are analyzed in advance. De Castilho et al. [dCCF82] proposed to check whether the transactions are always guaranteed to preserve the integrity constraints using theorem proving techniques. Kung [Kun85] presented a model-checking [Eme90] approach to testing the consistency of transaction specifications with integrity constraints. Completeness, decidability, and computational complexity issues were not addressed in either work. It would be interesting to see whether recent advances in model checking and theorem proving techniques can be applied in this area. In a related work, Lipeck [Lip90] showed how to refine transaction specifications, so that temporal integrity constraints are never violated.

Safety vs. Liveness. Does every temporal logic formula make sense as a temporal integrity constraint? Here the distinction between *safety* and *liveness* formulas [AS85; Pnu86] is very helpful. Intuitively, safety formulas say that "nothing bad ever happens" and liveness formulas – that "something good will happen". An example of a safety formula is $\Box\, \exists x.p(x)$, of a liveness formula $\Diamond\, \exists x.p(x)$.

Definition 39 (Property) *Any set of histories over a single schema can be considered a* property. *A property $\mathcal{P}$ is* defined *by a (closed) formula A if $\mathcal{P}$ is the set of histories that satisfy A.*

Definition 40 (Safety, AS85) *A property $\mathcal{P}$ is a* safety *property if for all histories $H = (H_0, H_1, H_2, \ldots)$ the following holds: if H does not belong to P, then some prefix $(H_0, H_1, \ldots, H_t)$ of H cannot be extended to any history in $\mathcal{P}$. A formula defining a safety property is called a* safety formula*.*

Definition 41 (Liveness, AS85) *A property $\mathcal{P}$ is a* liveness *property if any finite history $H = (H_0, H_1, H_2, \ldots, H_t)$ can be extended to an element of $\mathcal{P}$. A formula defining a liveness property is called a* liveness formula*.*

A violation of a safety formula can thus be always detected as a violation of its potential satisfaction in some finite history. Consequently, safety formulas are particularly suitable to the constraint checking scenario. On the other hand, liveness formulas are always potentially satisfiable. (A detailed discussion of safety vs. liveness in the database context can be found in [CN95], and in the context of proving properties of concurrent programs in [MP92].)

Similar considerations apply to temporal triggers, although because of the duality *negations* of safety formulas are of interest there. In the transaction validation scenario, both safety and liveness formulas are meaningful as constraints, because restricting updates to a fixed set of transactions prevents liveness formulas from being potentially satisfied in a trivial way. In the first two scenarios, future states can be arbitrary, while in the third they are partially determined by the given transactions. It may be interesting to investigate whether the methodology of proving safety and liveness properties developed by Manna and Pnueli [MP92] can be applied in the transaction validation scenario.

1.4.3 Temporal Constraint Checking

We summarize here the results about restricted classes of temporal integrity constraints expressed in temporal logic. Lipeck and Saake [LS87] proposed the class of *biquantified formulas* (without using this specific term). Biquantified formulas allow only future temporal operators and restricted quantification in the following sense: the quantifiers can be either *external* (not in the scope of any temporal connective) or *internal* (no temporal connective in their scope). Moreover, all external quantifiers are universal. Chomicki [Cho95] proposed the class of *past* temporal formulas, in which the only temporal connectives are ● and $\mathbf{since}$ and their derivatives, without any restrictions on quantification. The main theoretical results are as follows.

Theorem 42 (CN95) *For biquantified safety formulas with no internal quantifiers (called* universal*), potential constraint satisfaction is decidable (in exponential time). For biquantified safety formulas with a single internal quantifier, potential constraint satisfaction is undecidable*

Theorem 43 (Cho95) *For past formulas potential constraint satisfaction is undecidable.*

Notice that the first formulation of the constraint C_0 (Example 36) is a universal biquantified formula, while the second is a past formula. As far as we know the above results are still the only known characterizations of the computational complexity of checking temporal integrity constraints formulated in temporal logic.

For past formulas, Chomicki [Cho95] proposed a practical method for checking temporal logic constraints. As potential constraint satisfaction is undecidable for this class of constraints, a different notion of constraint satisfaction is used, namely the adaptation of Definition 35 to finite histories. (Such an adaptation is possible because past formulas refer only to the current history which is finite.) This notion is in general strictly weaker than potential satisfaction but for most practical constraints the two notions coincide. In Chomicki's method, every state H_i is augmented with new *auxiliary* relations to form an *extended state* H_i'. Only the last extended state is stored and used for checking constraints. For a fixed set of constraints, the number of auxiliary relations is fixed and their size is polynomial in the cardinality of the active domain of the current history. The method provides a lossy encoding of the current history: The given constraints can be accurately checked but arbitrary temporal queries cannot be precisely answered.

The method works as follows. For every constraint C, the extended database schema contains, in addition to database relations, an *auxiliary* relation r_α for every temporal subformula α of C. For each free variable of α, there is a different attribute of r_α.

Example 44 *Consider the second formulation of the constraint C_0 from Example 36. The auxiliary relations for this constraint are defined as follows:*

$$r_{\alpha_1}(x,y) \Leftrightarrow \blacklozenge takes(x,y)$$
$$r_{\alpha_2}(x,y) \Leftrightarrow \blacklozenge(takes(x,y) \wedge r_{\alpha_1}(x,y))$$

Given r_{α_1} and r_{α_2}, the constraint C_0 can be evaluated as the following query:

$$\neg\exists x.\exists y.(honors(x) \wedge r_{\alpha_2}(x,y)).$$

An auxiliary relation r_α at time i should contain exactly those domain values that make α true at i. Thus, auxiliary relations are defined *inductively*. First, their instances at time 0 are defined, and then it is shown how to obtain the instances at time $i+1$ from those at time i. The inductive definitions are automatically obtained from the syntax of the constraints.

Example 45 *Consider again the constraint C_0 from Example 36. We obtain the following definition of the auxiliary relation r_{α_1}:*

$$r^0_{\alpha_1}(x) \stackrel{\triangle}{=} False$$
$$r^{i+1}_{\alpha_1}(x,y) \stackrel{\triangle}{=} r^i_{\alpha_1}(x,y) \vee takes^i(x,y).$$

Similarly:

$$r^0_{\alpha_2}(x) \stackrel{\triangle}{=} False$$
$$r^{i+1}_{\alpha_2}(x,y) \stackrel{\triangle}{=} r^i_{\alpha_2}(x,y) \vee takes^i(x,y) \wedge r^i_{\alpha_1}(x,y).$$

The relation symbols with the superscript $i+1$ denote relations in the extended state after the update, and those with i denote relations in the extended state before the update. Moreover, the inductive definitions *do not depend on the value of i itself*, thus can be used as definitions of relational views that reference relations in the states before and after the update. These views should be materialized in every state, except for the first one, in the same way. In the first state, the views are initialized with empty relations. An implementation of the above method has been completed [CT95].

While in principle any temporal query language can be used to express temporal integrity constraints, in practice only temporal logic has been studied in this context. Why is it the case? The following property seems to be crucial: Each subformula of a temporal logic formula is associated with a single temporal context. This property is responsible for the limited expressiveness of temporal logic (see section 1.3) but also makes possible associating auxiliary (nontemporal) relations with subformulas and efficiently updating those relations according to the flow of time. Two-sorted first-order logic does not share this property and thus it seems unlikely that it will prove useful as a practical temporal constraint language.

There remain many open problems in the area of temporal database integrity. For example, one should consider more expressive constraint languages that use non-first-order constructs (see Section 1.6) or temporal aggregation. It seems that Chomicki's method [Cho95] can be adapted to handle at least some of those extensions. Another topic is the semantics and implementation of evolving sets of constraints. For a more detailed discussion of open problems in this area see [Cho95].

1.5 MULTIDIMENSIONAL TIME

In Section 1.3 we considered only single-dimensional temporal databases: temporal relations were allowed only a single temporal attribute. However, we have also seen that a single temporal dimension is not sufficient for the evaluation of first-order queries, as Theorems 18 and 19 show that higher temporal dimensions are necessary to represent the intermediate results during query evaluation. In this section we consider lifting the restriction to a single temporal dimension. There are two cases to consider:

- temporal models with fixed number of dimensions (> 1), and
- temporal models with a varying number of temporal dimensions without an upper bound.

The main result in this section is that that from the expressive power point of view, these two approaches are not equivalent (and this should not be a surprise anymore).

1.5.1 Why Multiple Temporal Dimensions?

To motivate the introduction of multiple temporal dimensions in the context of temporal databases, consider the following examples:

- *Bitemporal* databases: with each tuple in a relation two kinds of time are stored. The valid time (when a particular tuple is true) and the transaction time (when the particular tuple is inserted/deleted in the database) [JSS94].
- *Spatial* databases: multiple dimensions over an interpreted domain can be used for representing *spatial data* where multiple dimensions serve as coordinates of points in a k-dimensional Euclidean space.

Most of the data modeling techniques require only fixed-dimensional data. However, the true need for arbitrarily large dimensionality of data models originates in the requirement of having a first-order complete query language (see Theorem 48).

1.5.2 Abstract Query Languages for Multi-dimensional Time

The representation of multiple temporal dimensions in abstract temporal databases is quite straightforward: We merely index relational databases by the elements of an appropriate self-product of the temporal domain (in the case of snapshot temporal databases), or add the appropriate number of temporal attributes (in the case of timestamp temporal databases).

To define multidimensional temporal query languages we essentially follow the development of Section 1.3.

It is easy to see that the language L^P is inherently multi-dimensional: we simply abandon the restriction on the number of free temporal variables in queries. To define the multidimensional counterpart of L^Ω we first define the *multidimensional temporal connectives*.

Definition 46 (Multidimensional Temporal Connective) *Let $m > 0$ and $k \geq 0$. A* k-ary m-dimensional temporal connective *is a formula in the first-order language of the temporal domain T with exactly m free variables $t_0^1, \ldots, t_0^m$ and k free relation variables $X_1, \ldots, X_k$ (we assume that $t_i^1, \ldots, t_i^m$ are the only temporal variables free in the formula substituted for X_i).*

Similarly to Definition 5 we define Ω to be a finite set of definitions of temporal connectives, each definition consisting of a name $\omega(X_1, \ldots, X_k)$ and a definitional formula ω^*.

The language $L^{\Omega(m)}$ is a (single-sorted) first-order logic extended with a finite set $\Omega(m)$ of m-dimensional temporal connectives. The semantics of $L^{\Omega(m)}$ queries is defined using the satisfaction relation

$$\mathrm{DB}, \theta, t_1, \ldots, t_m \models \varphi$$

similarly to Definition 11: the only difference is that now we use m evaluation points $t_1, \ldots, t_m$ instead of a single evaluation point t. This definition can be used to define most of the common multi-dimensional temporal logics, e.g.,

- the temporal logic with the *now* operator [Kam71],
- the Vlach and Åqvist system [Åqv79], and
- most of the interval logics [All84; vB83].

Again, Definition 46 allows only logics with first-order definable temporal connectives. It also clarifies the difference between two distinct uses of intervals in temporal databases:

1. intervals as encodings of convex 1-dimensional sets, or
2. intervals as a representation of 2-dimensional points.

These two approaches assume completely different *meaning* to be assigned to the same construct—a pair of time instants—in different contexts. Consider the following two examples:

Example 47 First consider the following fragment of a concrete temporal database:

```
king("Charles IV",  "Czech Kingdom", [1347, 1378])
king("Casimir III", "Poland",        [1333, 1370])
```

In this case the intervals serve as encodings of their internal points: Charles IV was indeed the King of the Czech Kingdom *every year* (every time instant) between 1347 and 1378. In this setting, the set operations on intervals correspond to their boolean counterparts: to find out at what time both Charles IV and Casimir III were kings we can simply take the intersection of $[1347, 1378]$ and $[1333, 1370]$.

On the other hand, consider another fragment of a temporal database:

```
electricity("Jones A.", 40, 05/15/96, 06/15/96)
electricity("Smith J.", 35, 05/01/96, 06/01/96)
```

A tuple in the above relation stores the information about the electricity charges incurred by a customer in a given period of time. It is easy to see that here the intervals *do not* represent the sets of their internal points, but rather individual points in a 2-dimensional space. Thus applying set-based operations on these intervals does not have a clear and intuitive meaning.

Note that in Section 1.3 we used solely the first paradigm. The second paradigm often corresponds to languages $L^{\Omega(2)}$ [All84; vB83].

To compare the expressive power of temporal logics with respect to the dimension of the temporal connectives we use the following observation. The $L^{\Omega(m)}$ language can be used over a n-dimensional temporal database for $n < m$ by modifying the definition of the satisfaction relation as follows:

$$\mathrm{DB}, \theta, s_1, \ldots, s_m \models R(t_1, \ldots, t_n, \mathbf{x}) \iff (s_1, \ldots, s_n, \theta(\mathbf{x})) \in \mathbf{R}$$

Similarly we can assume that all temporal formulas from $L^{\Omega(n)}$ can be used as subformulas in $L^{\Omega(m)}$. Thus $L^{\Omega(m)} \sqsubseteq L^{\Omega(m+1)}$ over m-dimensional temporal databases. It is also easy to see that a natural extension of the Embed map to m dimensions,

Embed_m, gives us $L^{\Omega(m)} \sqsubseteq L^P$. The following theorem shows that all of the inclusions are proper:

Theorem 48 (TN96) $L^{\Omega(m)} \sqsubset L^{\Omega(m+1)}$ *for* $m > 0$ *and an arbitrary finite set of* m*-dimensional temporal connectives* $\Omega(m)$.

As a consequence $L^{\Omega(m)} \sqsubset L^P$ for all $m > 0$. Thus L^P is the only first-order complete temporal query language (among the languages discussed in this chapter). On the other hand, for any fixed query $\varphi \in L^P$ we can find an $m > 0$ such that there is an equivalent query in $L^{\Omega(m)}$. Thus, e.g., the query that was used to separate FOTL from 2-FOL in Section 1.3 can be expressed in $L^{\Omega(2)}$.

1.5.3 Encoding of Multi-dimensional Temporal Databases

Similarly to the single-dimensional case, storing the abstract multi-dimensional temporal databases directly may induce enormous space requirements. Thus we need to use encodings for multiple temporal dimensions. However, the introduction of multiple dimensions brings new challenges. The choice of encoding for sets of points in the multidimensional space is often much more involved than taking products of the encoding designed for the single-dimensional case. Assume that we attempt to represent the sets of points by hyper-rectangles—the multi-dimensional counterparts of intervals. It is easy to see that we can write first-order queries that do not preserve closure over this encoding:

Example 49 *Consider the query* $\varphi(t_1, t_2) = R(t_1) \land R(t_2) \land t_1 < t_2$. *This query evaluated over the database* $R = \{([1, 10])\}$ *returns a triangle-like region where, for all the points in the region, the first coordinate is less then the second coordinate.*

There are several ways of dealing with this issue:

- We can choose a *multi-dimensional temporal logic* where all the introduced connectives preserve closure over the chosen encoding.
- We can introduce closure restriction for formulas in L^P [CGK96; Tom97]. Such a restriction is designed to guarantee *attribute independence* of the free variables in the query and subsequently closure over an encoding obtained by taking an appropriate number of cartesian (self-)products of the single-dimensional encoding.
- We can use a more general encoding using constraints in some suitable constraint language [KKR95].

Another problem with using a multi-dimensional view of time is that it is much harder to define *normal forms* for temporal relations: in the single-dimensional case the coalesced relations provide a unique normal form (for the interval based encoding). However in two or more dimensions, such a normal form does not exist anymore (even when we only use hyper-rectangles).

1.6 BEYOND FIRST-ORDER TEMPORAL LOGIC

We survey here a number of temporal query languages whose expressive power goes beyond that of temporal logic. We have already seen one such formalism in section 1.3, namely two-sorted first-order logic L^P. Most of those languages have only recently been proposed and thus their relative expressive power is not completely known and implementation techniques (in particular compilation to concrete query languages) have yet to be developed. In all likelihood such an implementation will require the development of more powerful concrete query languages, as the present languages like TQuel or TSQL2 are not sufficiently expressive to serve as the targets of the compilation.

Second-order Temporal Connectives. The definition of temporal connectives (Definition 5) can be extended with *monadic* second-order quantification over the temporal domain (over subsets of the domain). This approach provides extra expressive power. For example, the unary connective "any time at an even distance from now" can be defined as (X_1 is a placeholder for the formula to which the connective applies):

$$\begin{array}{rcl} \equiv_2^0 X_1 & \stackrel{\triangle}{=} & \exists t_1.X_1 \wedge \exists S.t_0 \in S \wedge t_1 \in S \wedge closed(S) \\ & & \wedge \forall S'.(t_0 \in S' \wedge closed(S') \Rightarrow S \subseteq S') \end{array}$$

where

$$closed(S) \stackrel{\triangle}{=} \forall t.(t \in S \Leftrightarrow t+2 \in S).$$

For the temporal domain $(\mathbf{N}, <)$, the above extension is identical in expressive power to ETL, temporal logic with temporal connectives defined using regular expressions, studied by Wolper [Wol83] (the propositional case) and Abiteboul et al. [AHVdB96] (the first-order case). The latter paper also shows that the expressive power of ETL is incomparable to that of L^P. For other temporal domains, the expressive power of temporal logic with monadic second-order connectives has not yet been studied.

Fixpoints. For a general discussion of fixpoint query languages, see [AHV95]. A number of temporal fixpoint query languages have recently been recently proposed by Abiteboul et al. [AHVdB95]:

- TS-FIXPOINT: the extension of L^P with inflationary fixpoints,
- T-FIXPOINT: the extension of temporal logic with inflationary fixpoints and some additional constructs: moves back and forth in time, and local and non-inflationary variables (for details, see [AHVdB95]).

Abiteboul et al. [AHVdB95] also proposed the corresponding non-inflationary versions of those languages, and showed that TS-FIXPOINT is at least as expressive as T-FIXPOINT and that the relationship in the other direction depends on some unresolved questions in complexity theory. On the other hand, T-FIXPOINT is more expressive than L^P. These languages appear to be mainly of theoretical interest. Fixpoint temporal logic μTL [Var88] has been extensively used in program verification, although only in the propositional case. The first-order version of μTL remains to be studied. In particular, its relationship to T-FIXPOINT and TS-FIXPOINT needs to be elucidated.

Temporal Logic Programming. Another way to escape the limitations of temporal logic is to keep its syntax but use different semantics for its Horn subset. This is analogous to the move from first-order logic to logic programming. Indeed, proposals by Abadi and Manna [AM89], Baudinet [Bau92; Bau95], and Brzoska [Brz91; Brz93; Brz95] have been made to extend the language of Horn clauses with temporal connectives in such a way that there is still some notion of least model and resolution-based operational semantics, see Chapter **??**. Not surprisingly, those languages can be usually translated to the standard logic programming languages. For instance, the temporal connectives in Templog [AM89; Bau92; Bau95] can be simulated in Prolog using an additional predicate argument that can contain the successor function symbol [BCW93; CI88]. In this way, an exact correspondence is obtained between function-free Templog and *Datalog*$_{1S}$, an extension of Datalog with the successor function symbol in one predicate argument. More sophisticated temporal connectives involving numeric bounds on time [Brz91; Brz93; Brz95] can be simulated using arithmetic constraints in the Constraint Logic Programming paradigm of Jaffar and Lassez [JL87]. One can also study the extensions of the above Horn clause languages with various kinds of negation [AB94]. Recently, *Datalog*$_{1S}$ with negation has been used to define the operational semantics of active database systems: see Chapter **??**. Temporal logic programming languages are directly amenable to efficient implementation using the existing logic programming technology.

As far as the expressive power is concerned, it is not difficult to see that *Datalog*$_{1S}$ is subsumed by T-FIXPOINT and incomparable to ETL. *Datalog*$_{1S}$ with stratified negation strictly subsumes ETL but its relationship to T-FIXPOINT is unclear.

None of the above deductive or fixpoint languages operates on concrete temporal databases, as discussed in Section 1.3.3. Datalog over constraint encodings has been studied in [KKR95; TCR94].

We conclude this discussion by showing a real-life query expressible in the temporal logic programming languages and temporal fixpoint query languages mentioned above but not in any variant of temporal logic, including those with monadic second-order definable connectives. Intuitively, this query involves "recursion through time", and thus is also not definable in any first-order language, including L^P.

Example 50 *Find all the computers at risk—potentially infected by a virus—where "being at risk" is defined in the following way: a computer is* at risk *at a given time if it has been earlier infected by a virus or communicating with a computer already at risk. The formulation in Datalog$_{1S}$ is as follows:*

$$\begin{aligned}
&atRisk(T+1, X) \leftarrow infected(T, X).\\
&atRisk(T+1, X) \leftarrow atRisk(T+1, X).\\
&atRisk(T+1, X) \leftarrow communicate(T, X, Y), atRisk(T, Y).
\end{aligned}$$

1.7 CONCLUSIONS

Recent research has produced a better understanding of the mathematical foundations of temporal query languages and temporal integrity constraints. Also, practical implementation techniques for a number of semantically clean languages have been proposed. Temporal logic constitutes one of the focal points of this research.

Throughout this chapter, we have outlined some of the remaining open issues that deal with temporal language design, analysis, and implementation. Much work remains still to be done before temporal logic and related languages can fulfill their promise and become as ubiquitous in databases and information systems as time is.

References

[AB94] K. R. Apt and R. N. Bol. Logic Programming and Negation: A Survey. *Journal of Logic Programming*, 19-20:9–71, 1994.

[AH92] R. Alur and T. A. Henzinger. Logics and Models of Real-Time: A Survey. In *Real-Time: Theory in Practice*, pages 74–106. Springer-Verlag, LNCS 600, 1992.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AHVdB95] S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal Connectives versus Explicit Timestamps in Temporal Query Languages (preliminary report). In *International Workshop of Temporal Databases*, Zürich, Switzerland, September 1995. Springer-Verlag.

[AHVdB96] S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM Symposium on Principles of Database Systems*, pages 49–57, Montréal, Canada, June 1996.

[All83] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[All84] J. F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154, 1984.

[AM89] M. Abadi and Z. Manna. Temporal Logic Programming. *Journal of Symbolic Computation*, 8(3):277–295, September 1989.

[Åqv79] L. Åqvist. A Conjectured Axiomatization of Two-Dimensional Reichenbachian Tense Logic. *J. Philosophical Logic*, 8:1–45, 1979.

[AS85] B. Alpern and F. B. Schneider. Defining Liveness. *Information Processing Letters*, 21:181–185, 1985.

[Bau92] 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.

[Bau95] M. Baudinet. On the Expressiveness of Temporal Logic Programming. *Information and Computation*, 117(2):157–180, 1995.

[BCP94] E. Baralis, S. Ceri, and S. Paraboschi. Declarative Specification of Constraint Maintenance. In *International Conference on Entity-Relationship Approach*, pages 205–222. Springer-Verlag, LNCS 881, 1994.

[BCST96] M. Böhlen, J. Chomicki, R. T. Snodgrass, and D. Toman. Querying TSQL2 Databases with Temporal Logic. In *International Conference on Extending Database Technology*, pages 325–341, Avignon, France, 1996. Springer Verlag, LNCS 1057.

[BCW93] M. Baudinet, J. Chomicki, and P. Wolper. Temporal Deductive Databases. In Tansel et al. [TCG$^+$93], chapter 13, pages 294–320.

[Brz91] Ch. Brzoska. Temporal Logic Programming and its Relation to Constraint Logic Programming. In Vijay Saraswat and Kazunori Ueda, editors, *International Logic Programming Symposium*, pages 661–677. MIT Press, 1991.

[Brz93] Ch. Brzoska. Temporal Logic Programming with Bounded Universal Modality Goals. In David S. Warren, editor, *International Conference on Logic Programming*, pages 239–256. MIT Press, 1993.

[Brz95] Ch. Brzoska. Temporal Logic Programming in Dense Time. In John W. Lloyd, editor, *International Logic Programming Symposium*, pages 303–317. MIT Press, 1995.

[BSS96] M. Böhlen, R. T. Snodgrass, and M. D. Soo. Coalescing in Temporal Databases. In *International Conference on Very Large Data Bases*, pages 180–191, 1996.

[BTK91] F. Bacchus, J. Tenenberg, and J. A. Koomen. A Non-Reified Temporal Logic. *Artificial Intelligence*, 52(1):87–108, 1991.

[CGK96] J. Chomicki, D. Goldin, and G. Kuper. Variable Independence and Aggregation Closure. In *ACM Symposium on Principles of Database Systems*, pages 40–48, Montréal, Canada, June 1996.

[Cho94] J. Chomicki. Temporal Query Languages: A Survey. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic, First International Conference*, pages 506–534. Springer-Verlag, LNAI 827, 1994.

[Cho95] J. Chomicki. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186, June 1995.

[CI88] J. Chomicki and T. Imieliński. Temporal Deductive Databases and Infinite Objects. In *ACM Symposium on Principles of Database Systems*, pages 61–73, Austin, Texas, March 1988.

[CN95] J. Chomicki and D. Niwinski. On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535, December 1995.

[Cod72] E. F. Codd. Relational Completeness of Data Base Sub-Languages. In R. Rustin, editor, *Data Base Systems*, pages 33–64. Prentice-Hall, 1972.

[CT95] J. Chomicki and D. Toman. Implementing Temporal Integrity Constraints Using an Active DBMS. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):566–582, August 1995.

[CW90] S. Ceri and J. Widom. Deriving Production Rules for Constraint Maintenance. In Dennis McLeod, Ron Sacks-Davis, and Hans-Joerg Schek, editors, *International Conference on Very Large Data Bases*, pages 566–577, 1990.

[dCCF82] J. M. V. de Castilho, M. A. Casanova, and A. L. Furtado. A Temporal Framework for Database Specifications. In *International Conference on Very Large Data Bases*, pages 280–291, 1982.

[Eme90] E. A. Emerson. Temporal and Modal Logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier/MIT Press, 1990.

[GHR94] D. M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press, 1994.

[GL96] M. Gertz and U. Lipeck. Deriving Optimized Integrity Monitoring Triggers from Dynamic Integrity Constraints. *Data and Knowledge Engineering*, 20(2):163–193, 1996.

[HS91] K. Hülsmann and G. Saake. Theoretical Foundations of Handling Large Substitution Sets in Temporal Integrity Monitoring. *Acta Informatica*, 28(4), 1991.

[IK89] N. Immerman and D. Kozen. Definability with Bounded Number of Bound Variables. *Information and Computation*, 83(2):121–139, November 1989.

[ISO92] ISO. Database Language SQL. ISO/IEC 9075:1992, International Organization for Standardization, 1992.

[JL87] J. Jaffar and J-L. Lassez. Constraint Logic Programming. In *ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[JSS94] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying Temporal Data Models via a Conceptual Model. *Information Systems*, 19(7):513–547, 1994.

[Kam68] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

[Kam71] J. A. W. Kamp. Formal Properties of 'now'. *Theoria*, 37:227–273, 1971.

[KKR95] P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint Query Languages. *Journal of Computer and System Sciences*, 51(1):26–52, August 1995.

[Koy89] R. Koymans. *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. PhD thesis, Technische Universiteit Eindhoven, 1989.

[KSW95] F. Kabanza, J-M. Stevenne, and P. Wolper. Handling Infinite Temporal Data. *Journal of Computer and System Sciences*, 51(1):3–17, August 1995.

[Kun85] C. H. Kung. On Verification of Database Temporal Constraints. In *ACM SIGMOD International Conference on Management of Data*, pages 169–179, Austin, Texas, 1985.

[Lip90] U. Lipeck. Transformation of Dynamic Integrity Constraints into Transaction Specifications. *Theoretical Computer Science*, 76(1):115–142, 1990.

[Lor93] N. A. Lorentzos. The Interval-Extended Relational Model and Its Application to Valid-time Databases. In Tansel et al. [TCG+93], pages 67–91.

[LS87] U. W. Lipeck and G. Saake. Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems*, 12(3):255–269, 1987.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[Pnu86] A. Pnueli. Applications of Temporal Logic to the Specification and Verification of Reactive Systems: a Survey of Current Trends. In *Current Trends in Concurrency*, pages 510–584. Springer-Verlag, LNCS 224, 1986.

[SA86] R. T. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, 19(9), 1986.

[SAA+94] R. T. Snodgrass, I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kafer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. A. Sripada. TSQL2 language specification. *SIGMOD Record*, 23(1):65–86, March 1994.

[SBJS96] R. T. Snodgrass, M. H. Böhlen, C. S. Jensen, and A. Steiner. Adding Valid Time to SQL/Temporal. ISO/IEC JTC1/SC21/WG3 DBL MAD-146r2 21/11/96, (change proposal), International Organization for Standardization, 1996.

[SJB95] R. T. Snodgrass, C. S. Jensen, and M. H. Böhlen. Evaluating and Enhancing the Completeness of TSQL2. Technical Report TR 95-5, Computer Science Department, University of Arizona, 1995.

[Sno87] R. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Sno93] R. T. Snodgrass. An overview of TQuel. In Tansel et al. [TCG$^+$93], pages 141–182.

[Sno95] R. T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.

[Sta79] J. Stavi. Functional Completeness over Rationals. Unpublished manuscript, Bar-Ilan University, Israel, 1979.

[SW95] A. P. Sistla and O. Wolfson. Temporal Triggers in Active Databases. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):471–486, 1995.

[TC90] A. Tuzhilin and J. Clifford. A Temporal Relational Algebra as a Basis for Temporal Relational Completeness. In Dennis McLeod, Ron Sacks-Davis, and Hans-Joerg Schek, editors, *International Conference on Very Large Data Bases*, pages 13–23, 1990.

[TCG$^+$93] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.

[TCR94] D. Toman, J. Chomicki, and D. S. Rogers. Datalog with Integer Periodicity Constraints. In Maurice Bruynooghe, editor, *International Logic Programming Symposium*, pages 189–203. MIT Press, 1994. Full version to appear in Journal of Logic Programming.

[TN96] D. Toman and D. Niwinski. First-Order Queries over Temporal Databases Inexpressible in Temporal Logic. In *International Conference on Extending Database Technology*, pages 307–324, Avignon, France, 1996. Springer-Verlag, LNCS 1057.

[Tom96] D. Toman. Point vs. Interval-based Query Languages for Temporal Databases. In *ACM Symposium on Principles of Database Systems*, pages 58–67, Montréal, Canada, June 1996.

[Tom97] D. Toman. Point-based Temporal Extensions of SQL. In *International Conference on Deductive and Object-Oriented Databases*, 1997.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1989.

[Var88] M. Y. Vardi. A Temporal Fixpoint Calculus. In *ACM Symposium on Principles of Programming Languages*, pages 250–259, 1988.

[vB83] J. F. A. K. van Benthem. *The Logic of Time*. D. Reidel, 1983.

[WC96] J. Widom and S. Ceri, editors. *Active Database Systems*. Morgan Kaufmann, 1996.

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56:72–99, 1983.

[Wol89] P. Wolper. On the Relation of Programs and Computations to Models of Temporal Logic. In B. Banieqbal, B. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, pages 75–123. Springer-Verlag, LNCS 398, 1989.

Index