Temporal Databases: Beyond Finite Extensions (Position Paper)

Marianne Baudinet
Université Libre de Bruxelles*

Jan Chomicki
Kansas State University†

Pierre Wolper
Université de Liège‡

January 1993

Abstract

We argue that temporal databases should not be restricted to relations with finite extensions. Many temporal events are periodic and have no natural bounds. Moreover, such events have a more compact representation when allowed to be unbounded. We present two formalisms for representing and querying possibly infinite periodic data and discuss some of their properties, including expressiveness and query evaluation complexity. Finally, we turn to implementation issues and argue that significant extensions to existing database systems are necessary in order to implement the frameworks we describe.

1 Position

A large body of work has so far been devoted to the problem of adding time in databases and particularly in relational databases [BADW82, McK86, SS88, Sno90, Soo91]. Many different models of temporal databases have been proposed, often raising and addressing different aspects of the problem ([SA85, JS92] provide nice comparative surveys of these). Even though these approaches seem to diverge on various points, they implicitly rely on one major assumption: a relation, be it time-dependent, always has a finite extension. This means that even if the temporal database model relies on an unbounded time domain, arbitrary limits must be imposed on the time range of the database relations. For instance, if some fact is true every day at 8 a.m., its extension includes as many time points as there are days on which the fact is true. Thus, this representation is only usable

*Address: Informatique; 50 Avenue F.D. Roosevelt, C.P. 165; 1050 Brussels; Belgium. Email: mb@cs.ulb.ac.be

†Address: Department of Computing and Information Sciences, Nichols Hall, Kansas State University, Manhattan, KS 66502. Email: chomicki@cis.ksu.edu

‡Address: Institut Montefiore, B28; 4000 Liège Sart-Tilman; Belgium. Email: pw@montefiore.ulg.ac.be

if the number of days is finite (or arbitrarily restricted to be finite) and, in practice, if it is not too large. Furthermore, such a representation does not allow any exploitation of the temporal pattern of data, for instance in the evaluation of queries or in the presentation of query answers. These reasons lead us into believing that limiting temporal relations to finite extensions is a counterproductive restriction. Rather, we demonstrate in this paper that using implicit representations from which the (possibly infinite) extensions can be computed is a very promising alternative.

We consider two different implicit representations of temporal extensions: one based on repetition and constraints, and one based on deductive rules. In order to express that some information is true every day at 8 a.m., one can write that it is true at every time point of the form $t_0 + 24n$ where t_0 corresponds to the first occurrence of 8 a.m. at which the information holds and n is a variable ranging over the integers. Furthermore, if necessary, one can limit the range of the variable n by introducing constraints. For instance, one can specify that the temporal extension being considered consists of all time points x such that x is of the form $t_0 + 24n$ and such that $0 \leq x \leq 673$. Section 3 is devoted to this representation, which was introduced in [KSW90] and further studied in [BNW91]. Related ideas about the use of constraints in databases were presented in [KKR90] and [Rev90].

In order to express deductively that some information is true every day at 8 a.m., one can say that it is true at some time t_0 corresponding to the first occurrence of 8 a.m. at which the information holds, and then specify that, if the information is true at some time t, it is true at time $t + 24$ hours. This form of representation has been studied since [CI88] and [Bau89b], and is presented in Section 4.

The main advantage of implicit representations is expressiveness. They make the representation of infinite extensions possible and often allow for a more compact representation of finite extensions. Expressiveness is thus one important comparison and evaluation criterion when discussing the formalisms we have just outlined. In Section 5, we introduce the relevant notions of expressiveness and use them to compare the two formalisms we present. But expressiveness is not the only issue. Representing information is only useful if it can be queried with reasonable efficiency. Therefore, a second comparison and evaluation criterion when discussing formalisms for infinite temporal extensions is the complexity of evaluating queries. We also discuss this issue in Section 5. Finally, in Section 6, we discuss implementation issues and argue that the formalisms we present require significant extensions to existing database systems. We view building temporal databases that can handle infinite extensions as one of the challenges of temporal database research for the forthcoming years.

2 Preliminaries

In this paper, the time domain that we use is discrete and unbounded. We do not wish to get into philosophical arguments as to whether time is actually bounded in the past by the Big Bang and in the future by the "Big Crunch" [JS92]. That is not our point. Rather, we consider it a useful abstraction to adopt a model of time that is unbounded. Since we also consider time to be discrete, we view time as isomorphic to the integers.

We are concerned with the representation of valid time [JCG+92]. We consider relations that have any number of non-temporal data attributes and one or more temporal attributes ranging over the temporal domain, that is, the set of integers. A *temporal database* is a finite collection of such relations. The extension of the non-temporal attributes is assumed to be represented explicitly. Our concern is the representation of the possibly infinite extensions of the temporal attributes.

3 Representing Infinite Periodic Data with Constraints

3.1 Definition of the Framework

The *generalized databases* of [KSW90] generalize the usual relational databases by allowing the relations to have, in addition to the usual data attributes, one or more temporal attributes whose values are not simply time tags, but are in fact (possibly infinite) periodic sets of time tags (of integer values). These periodic sets, called *linear repeating points*, are finitely represented by their periodicity and a value in the set. Furthermore, they may be constrained individually or related pairwise by linear inequalities. Let us define the framework of generalized databases more precisely.

Definition 3.1 A *linear repeating point* (*lrp*) is a set

$$\{x(n) \in \mathcal{Z} \quad | \quad x(n) = an + b, \text{ with } a, b \text{ in } \mathcal{Z}, \\ \text{and } n \text{ ranging from } -\infty \text{ up to } +\infty \text{ in } \mathcal{Z}\},$$

where $\mathcal{Z}$ denotes the set of integers. Such a set is simply denoted by $an + b$. The constant a is the periodicity and the constant b is a value in the set (for $n = 0$).

For instance, the lrp $5n + 3$ denotes the infinite periodic set of integers $\{\ldots, -7, -2, 3, 8, 13, \ldots\}$. Notice that when its periodicity is 0 ($a = 0$), an lrp simply denotes a singleton.

The constraints that may apply to the temporal attributes are conjunctions of atomic constraints, that is, of linear equalities or inequalities between at most two temporal attributes in

which the coefficients of temporal attributes are 1. More specifically, if T_i and T_j are temporal attributes, atomic constraints are of the form

$$T_i \leq T_j + a, \quad T_i = T_j + a, \quad T_i \leq (\geq) a, \quad \text{or} \quad T_i = a.$$

These constraints allow one to "couple-up" the time values actually retained out of two periodic sets. Let us illustrate this.

Example 3.1 The tuple of lrp's $(2n_1 + 3,\ 2n_2 + 5)$ constrained by the equality $T_2 = T_1 + 2$ (T_1 and T_2 respectively denote the first and the second temporal attribute) represents the infinite set of tuples

$$\{\ldots, (-1, 1), (1, 3), (3, 5), \ldots\}.$$

It is thus a subset of the cartesian product $(2n_1 + 3) \times (2n_2 + 5)$.

Definition 3.2 Let $\mathcal{T}$ denote the collection of lrp's and $\mathcal{D}$ a collection of nontemporal data values. A *generalized tuple* of temporal arity m and data arity ℓ is an element of $\mathcal{T}^m \times \mathcal{D}^\ell$, together with constraints on the temporal components. In other words, a *general tuple* of temporal arity m and data arity ℓ is a ground tuple of the form

$$(a_1 n_1 + b_1,\ \ldots,\ a_m n_m + b_m,\ d_1, \ldots, d_\ell) \text{ with } \mathit{constraints}(T_1, \ldots, T_m)$$

where

- each $a_i n_i + b_i$ $(1 \leq i \leq m)$ is an lrp (we assume that the variables n_i are all distinct),
- each d_k $(1 \leq k \leq \ell)$ is a data constant from $\mathcal{D}$,
- $\mathit{constraints}(T_1, \ldots, T_m)$ denotes a finite set of constraints over the temporal attributes $T_1, \ldots, T_m$.

Such a generalized tuple is in fact a finite representation of a possibly infinite set of ground tuples, namely the set

$$\{(t_1, \ldots, t_m, d_1, \ldots, d_\ell) \quad | \quad t_1 \in (a_1 n_1 + b_1), \ldots, t_m \in (a_m n_m + b_m), \\ \text{and } \mathit{constraints}(t_1, \ldots, t_m) \text{ is satisfied } \}.$$

The constraints $\mathit{constraints}(t_1, \ldots, t_m)$ in a generalized tuple thus define a subset of the cartesian product of the lrp's $(a_1 n_1 + b_1) \times \ldots \times (a_m n_m + b_m)$, namely the subset of time-point tuples that are actually in the database.

Example 3.2 The generalized tuple of the relation

$train\_schedule(departure\_time, arrival\_time, origin, destination)$,

train_schedule				
$40n_1 + 5$	$40n_2 + 65$	*liège*	*brussels*	with $T_1 \geq 0 \wedge T_2 = T_1 + 60$,

represents the fact that there is a train from Liège to Brussels at time 5 and every 40 minutes thereafter, each train arriving 60 minutes after having left (we assume a time granularity of one minute here, and take time 0 to be some fixed time, say 0 o'clock some Monday morning).

The above tuple thus represents the following infinite set of time-point tuples.

$$\{(t_1, t_2, liège, brussels) \quad | \quad t_1 \in (40n_1 + 5),\ t_2 \in (40n_2 + 65),\ t_1 \geq 0 \quad \text{and} \quad t_2 = t_1 + 60\}$$

A relation in the generalized database framework is, as usual, a finite set of tuples of a given schema.

Definition 3.3 A *generalized relation* is a finite set of generalized tuples of a given schema.

Example 3.3 The generalized relation *action* below describes the timing of some activities of two robots. Its first two attributes are temporal attributes and its third and fourth attributes are usual data attributes.

action				
0	1	*robot1*	*task1*	
$10n_1 + 2$	$10n_2 + 6$	*robot1*	*task2*	$T_1 = T_2 - 4 \wedge T_1 \geq 0$
$8n_1 + 5$	$8n_2 + 7$	*robot2*	*task1*	$T_1 = T_2 - 2 \wedge T_1 \geq 8$
$10n_1$	$10n_2 + 3$	*robot2*	*task2*	$T_1 = T_2 - 3$

It must be noted that generalized relations with an arbitrary number of temporal attributes are allowed. Generalized relations with one temporal attribute correspond to point-based temporal data: they allow for the representation of facts that hold periodically at time points. Generalized relations with two temporal attributes correspond to interval-based temporal data: they allow for the representation of facts that hold periodically over intervals. Generalized relations with more than two temporal attributes can be useful for representing periodic data for which a beginning

interval and an ending interval is known for instance. They are also useful as intermediate steps in computations. For instance, when concatenating temporal intervals, the mid-point at which the intervals are joined appears as a third temporal attribute before being projected out.

3.2 Query Languages for Generalized Databases

Generalized databases can be queried exactly like classical relational databases, using relational algebra and calculus.

3.2.1 Relational Algebra for Generalized Databases

[KSW90] gives a detailed description of how the various relational algebra operations are computed on generalized databases. Of all these operations, the trickiest is the projection. This is due to the presence of the constraints that may relate temporal attributes. In a classical relational database, the projection of a relation on some of its columns simply requires forsaking the other columns. In a generalized database, before forsaking columns that represent temporal attributes, its constraints have to be projected out, that is, the temporal variables denoting the temporal attributes that are projected out must be eliminated from the system of constraints. This means that the projection operation requires using an algorithm that allows one to eliminate variables from an integer constraint system. Would the temporal attributes range over the reals, it would be very simple: the algorithm would compute linear combinations of the constraints. But here, the temporal attributes can only take integer values, and such a solution is not adequate. The solution described in [KSW90] consists in "normalizing" each tuple of the concerned relation in such a way that all its temporal components have the same periodicity. Only thereafter can the projection be carried out with algorithms that are appropriate for the real numbers.

The complexity of computing the relational algebra operations is studied in [KSW90]. The general result is that everything can be done in polynomial time when the relations are normalized.[1] However, one should notice that normalizing a relation may cause a significant blow-up of the space required to store the relation.

3.2.2 A First-Order Query Language for Generalized Databases

Once algorithms for computing the relational algebra operations are known, one can consider the evaluation of first-order queries. The natural first-order language to use with generalized databases

[1] This complexity measure, called the *fixed-schema* complexity assumes that the schema of the relations is fixed and only the number of tuples may vary. When the schema may also vary, all the relational algebra operations still take polynomial time except for the difference which takes exponential time.

is a two-sorted first-order logic. One sort is temporal points (interpreted over the integers), the other is generic.

The language includes one interpreted predicate of temporal arity 2 and data arity 0, namely $\leq$, and any number of uninterpreted predicates. In the spirit of what is done in relational databases, function symbols are not used on the generic sort. On the temporal sort, one interpreted function can be used, namely the successor function $(+1)$. Arbitrary quantification is allowed on both temporal and nontemporal variables.

Example 3.4 The following formula of this two-sorted first-order query language refers to the data of Example 3.3. It expresses the fact that there are robots x and y and time instants t_1 and t_2 such that, if x performs *task2* over a time interval $[t_1, t_2]$ of length at least 5, then y does not perform any task over any subinterval of that time interval.

$$\exists X \exists Y \exists T_1 \exists T_2 \forall T_3 \forall T_4 \forall Z$$
$$[\,(action(T_1, T_2, X, \text{task2}) \wedge T_1 \leq T_3 \leq T_4 \leq T_2 \wedge T_1 + 5 \leq T_2) \supset \neg action(T_3, T_4, Y, Z)\,]$$

It turns out that the time complexity of evaluating such queries on a generalized database is polynomial in the size of the database. This complexity measure is in fact the data-complexity measure as defined by Vardi [Var82].

3.3 Extensions of the Framework

In Example 3.2, we have assumed a time granularity of one minute. However, other kinds of temporal data may require using other basic time units. Indeed, different temporal events have different periodicities, each of which is most easily expressed in some natural time unit (such as e.g. hour, day, week, month, semester, year, etc.). Using the same basic unit of time for all temporal facts of a database could thus lead to very cumbersome representations of the facts: consider e.g. how one would represent, using a minute as the basic time unit, the occurrence of an event every Monday of the second week of every month. [NS92] proposes an extension of the generalized database formalism that supports natural time units and exploits the relationships between these time units in the evaluation of relational algebra operations.

Another extension of the generalized database framework, proposed in [BNW91], introduces a deductive query language (such as those presented in Section 4), which is further studied in [FVP92].

4 Representing Infinite Extensions Using Deductive Rules

Deductive databases extend relational databases by allowing some relations, the *intensional relations*, to be defined by recursive rules in a deductive language, in addition to the usual relations stored in the database, the *extensional relations*. The best known deductive language is the logic programming language *Datalog*, which is based on Horn clauses [GMN84, Ull88, Ull89, GM92]. We refer the reader to [vEK76, Llo87, Ull88] for standard definitions regarding logic programming and *Datalog*.

Example 4.1 A standard illustration of *Datalog* is the *ancestor* relation. Let us assume that the binary relation *parent* is given in the database (it is an extensional relation). Then, the following two rules recursively define the intensional relation *ancestor* in terms of the *parent* relation.

$$ancestor(X, Y) \leftarrow parent(X, Y)$$
$$ancestor(X, Y) \leftarrow parent(X, Z) \ \& \ ancestor(Z, Y)$$

These rules state that, whoever X and Y are, X is an ancestor of Y if either X is a parent of Y or there is a person Z such that X is a parent of Z and Z is an ancestor of Y.

Queries are also expressed in the deductive language.

Example 4.2 For instance,

$$\leftarrow ancestor(adam, U)$$

where U is a variable and *adam* a constant, requests all the persons of whom 'adam' is an ancestor.

The deductive approach has been extended to handle temporal data. The temporal deductive language $Datalog_{1S}$ (*Datalog* with one successor, previously called *temporal deductive databases*) has been proposed and investigated in [CI88, CI89, Cho90b, Cho90a]. $Datalog_{1S}$ extends *Datalog* by allowing every predicate (that is, relational symbol) to have (at most) one temporal parameter in addition to the usual data parameters. Temporal parameters are temporal terms that are constructed using a distinguished constant 0, variables, and the unary function symbol $+1$ (the *successor* function). They can be viewed as being interpreted over the natural numbers.

Example 4.3 The following rules schedule the meetings of a professor with his students. The intensional predicate *meets* has one temporal and one non-temporal attribute. It is defined in

terms of two extensional predicates *meets_first*, which also has one temporal and one non-temporal attribute, and *follows*, which has two non-temporal attributes.

$$meets(T, X) \leftarrow meets\_first(T, X)$$
$$meets(T + 1, Y) \leftarrow follows(X, Y) \,\&\, meets(T, X)$$

Let us consider an extensional database containing the following facts.

$$meets\_first(0, emma)$$
$$follows(emma, kathy)$$
$$follows(kathy, emma)$$

The above deductive rules can be used to derive the following infinitely many facts from the database.

$$meets(0, emma)$$
$$meets(1, kathy)$$
$$meets(2, emma)$$
$$meets(3, kathy)$$
$$\ldots$$

Notice that these facts form a periodic set when viewed as a function of the integers, but that their period depends on the contents of the database.

Example 4.4 Consider the following deductive rules governing the schedule of backups in a distributed system.

$$backup(T + 24, X) \leftarrow backup(T, X)$$
$$backup(T, Y) \leftarrow dependent(X, Y) \,\&\, backup(T, X)$$

The first rule states that a backup on a machine should be taken *every 24 hours*. The second rule requires that the backups should be taken simultaneously on all *dependent* machines (e.g., sharing files). Notice that, whatever database these rules are applied to, the derived facts will again be periodic, but this time, the period will not depend on the contents of the database.

Note: A temporal-logic based extension of *Datalog* proposed in [AM89], namely *Templog*, has also been investigated as a deductive language for representing and querying temporal data [Bau89b, Bau89a, Bau92]. *Templog* and $Datalog_{1S}$ have been shown to be equivalent to one another for representing and querying temporal data [Bau89a, Cho90a]. In *Templog*, predicates can vary with time (that is, they can represent a different relation at every time instant), but the time point they refer to is defined implicitly by temporal operators rather than by an explicit temporal argument, as in $Datalog_{1S}$.

5 Comparison and Evaluation Criteria for Infinite Temporal Databases

There are two main natural criteria for evaluating a scheme for representing infinite temporal information. The first is what it can represent (expressiveness), the second is how hard or easy it is to extract information from the representation (complexity). We first deal with expressiveness.

5.1 Expressiveness Issues

There are several notions and issues in expressiveness. To focus our discussion, let us consider three typical statements about expressiveness.

1. Intervals are more expressive than points.
2. One can represent the fact that X happens the first Thursday of each month.
3. Deductive languages are more expressive than first-order languages.

We will now argue that these three statements refer to different notions of expressiveness and show how these notions apply to the frameworks presented in Sections 3 and 4.

5.1.1 Interval-Based Approach Versus Point-Based Approach

It is common in the AI literature to see intervals studied as basic objects distinct from points. This might seem counter to the intuition that an interval is defined by two points. Unless one deals with very unusual models of time, this intuition is perfectly correct. What distinguishes point and interval models are the type of temporal predicates that are used. In a point approach, the predicates are unary ($p(t)$). In an interval approach, predicates are binary ($p(t_1, t_2)$); thus one might very well have that $p(t_1, t_2)$ is true without having that $p(t_3, t_4)$ is true for $t_1 < t_3 < t_4 < t_2$. On the other hand, in a point approach, the only meaning one can give to a predicate p being true on an interval $[t_1, t_2]$ is $(\forall t : t_1 \leq t \leq t_2)p(t)$. This of course implies that p is also true in any interval $[t_3, t_4]$ such that $t_1 < t_3 < t_4 < t_2$.

Of the two formalisms we have described, the one based on repeating points allows predicates of arbitrary temporal arity. The one based on deductive rules only allows unary predicates. Thus, from this point of view repeating points are preferable. However, practical cases where genuine intervals are necessary are not that frequent.

5.1.2 Data Expressiveness

The second statement above refers to what can be represented in a temporal database. If one thinks in terms of traditional databases, this might seem like a nonissue. Indeed, traditional databases are limited to relations with finite extensions and, of course, all finite extensions are representable. The two formalisms we have presented, however, describe relations with infinite extensions. Now, it is clear that for cardinality reasons, it is impossible to have a finite representation of all infinite extensions. For instance, relations with exactly one temporal attribute ranging over the integers have extensions that are (finite or infinite) sets of integers. There are uncountably many such sets and hence all cannot be finitely represented.

Thus, one needs to answer the question of which fragments of the class of infinite extensions the formalisms we have proposed actually capture. This notion is referred to as *data expressiveness* in [BNW91], where it is shown that both our formalisms capture the class of periodic sets.

5.1.3 Query Expressiveness

The third statement we mentioned above refers to the queries that can be asked of a temporal database. Again, for the usual set-theoretic reasons, no language can represent all queries. Thus the issue of what queries a language can represent is meaningful. Note that this notion, referred to as *query expressiveness* in [BNW91], is quite distinct from data expressiveness. For example, the data expressiveness of a unary temporal relation is the *sets* of integers it can represent, whereas the query expressiveness of a yes/no query language over such a relation is the *sets of sets* of integers the language can define.

The query expressiveness of the two formalisms we have discussed are incomparable, though they are both adequate for most applications. One can find a precise discussion of these issues in [BNW91] where some characterizations of the expressiveness of our languages are given.

5.2 Query Evaluation and Complexity issues

In a database system, one wants query evaluation to be fast. This is not a property that will be destroyed by our more expressive but less explicit representations of temporal information. There are two ways to look at the problem: theoretical complexity results, and what one could expect for common queries in an implemented system.

A summary of theoretical complexity results can be found in [BCW92]. Basically, these results say that, in the general case, complexity is high (EXPTIME or PSPACE), though there are many cases where it is PTIME. This last class of results does not mean more than 'efficiency is not

impossible'. There are no experiences with implemented systems, but we are quite optimistic about the possibility of obtaining reasonable performance.

Finally, note that one major advantage of the formalisms we have presented is that they allow the presentation of query answers in the same way as the content of the database. This can be an asset both from a complexity and an ease of use point of view. Indeed, it can be much more efficient to handle representations of large sets than the sets themselves, and it is certainly more useful to know that something occurs every Monday than at the equivalent list of dates.

6 Implementations of Infinite Periodic Sets

In this section, we discuss whether infinite periodic sets can be implemented using existing database technology. We consider the two approaches to the representation of infinite periodic sets presented in Sections 3 and 4.

6.1 Constraint-Based Approach

The constraint-based approach uses traditional relational query languages (calculus and algebra) but demands a considerably more general notion of *attribute domain*, as well as specialized algorithms for evaluating queries.

A new domain of *linear repeating points* (lrp's) is necessary. In this domain the following operations need to be supported: intersection, subtraction, period multiplication, and equality test [KSW90]. This domain may be implemented using the *abstract data type* facility provided by some experimental database systems like Postgres [SK91] or Starburst [LLPS91].

Another domain that is required is that of *linear arithmetic constraints* with some simple syntactic transformations of constraints including merging, splitting, and negation [KSW90]. Such constraints are associated with every tuple, so the simplest way to implement them would be to add an extra attribute to every temporal relation, whose values are linear arithmetic constraints. Again, an abstract data type facility is essential to support such a construct.

Both linear repeating points and arithmetic constraints are *expressions* containing logical variables. Unfortunately, no extensible relational database system we are familiar with (including Postgres and Starburst) provides support for symbolic data of this kind. In fact linear repeating points and arithmetic constraints would probably have to be stored as strings, and operations would have to be provided to convert back and forth between strings and an internal form on which the syntactic transformations could more readily be performed.

The implementations of standard operations of relational algebra have to be specialized in the constraint-based approach. They would have to take into account the special character of temporal attributes (containing lrp's) and constraints. Although the demand for providing a *specialization facility* for relational algebra operations seems to transcend the area of temporal databases (it may also be useful in, e.g., spatial databases), we know of no extensible relational database system providing it.

6.2 Deductive Approaches

Both *Datalog$_{1S}$* and *Templog* are logic programming languages. Moreover, *Datalog$_{1S}$* being a syntactic subset of Prolog, it is in a limited way available to any user of a Prolog system. Nonetheless, no Prolog system we are familiar with provides support for the finite representation of infinite query answers [CI89, CI93], which is essential when dealing with infinite periodic sets. Moreover, the termination of standard top-down or bottom-up query evaluation in the context of *Datalog$_{1S}$* cannot be guaranteed [Cho90a]. A specialized version of bottom-up evaluation can be guaranteed to terminate for finite-answer queries [Cho90a]. However, no existing logic programming or deductive database system provides the means to specialize its evaluation procedure.

7 Conclusions

As can be seen from the above discussion, the problems of providing a comprehensive framework for the implementation of infinite periodic sets are largely independent of the existence of a common relational-style infrastructure for temporal databases à la TQuel [Sno87] (or a temporal extension of SQL). In this implementation task, the availability of powerful extensible, relational or deductive, database systems is crucial. We conjecture that the implementation of other advanced temporal concepts, like, e.g., indefinite temporal data, will share the above property and will only be facilitated by the advances in the area of *general* extensible database systems.

References

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

[Bau89a] M. Baudinet. *Logic Programming Semantics: Techniques and Applications*. PhD thesis, Stanford University, February 1989.

[Bau89b] M. Baudinet. Temporal Logic Programming is Complete and Expressive. In *ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1989.

[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.

[BNW91] M. Baudinet, M. Niézette, and P. Wolper. On the Representation of Infinite Temporal Data and Queries. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1991.

[BCW92] Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Temporal deductive databases. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases*, Database Systems and Application Series. Benjamin/Cummings, 1992. Forthcoming.

[BADW82] A. Bolour, T.L. Anderson, L.J. Dekeyser, and H.K.T. Wong. The role of time in information processing: A survey. *ACM SIGArt Newsletter*, 80:28–48, April 1982.

[Cho90a] J. Chomicki. *Functional Deductive Databases: Query Processing in the Presence of Limited Function Symbols*. PhD thesis, Rutgers University, New Brunswick, New Jersey, January 1990. Also Laboratory for Computer Science Research Technical Report LCSR-TR-142.

[Cho90b] J. Chomicki. Polynomial-Time Computable Queries in Temporal Deductive Databases. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Nashville, Tennessee, April 1990.

[CI88] J. Chomicki and T. Imieliński. Temporal Deductive Databases and Infinite Objects. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Austin, Texas, March 1988.

[CI89] J. Chomicki and T. Imieliński. Relational Specifications of Infinite Query Answers. In *ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, May 1989.

[CI93] J. Chomicki and T. Imieliński. Finite Representation of Infinite Query Answers. *ACM Transactions on Database Systems*, 1993. To appear.

[FVP92] Laurent Fribourg and Marcos Veloso Peixoto. Bottom-up evaluation of Datalog programs with arithmetic constraints. Technical Report LIENS-92-13, Laboratoire d'Informatique de l'Ecole Normale Supérieure, Paris, June 1992.

[GMN84] H. Gallaire, J. Minker, and J. Nicolas. Logic and Databases: a Deductive Approach. *ACM Computing Surveys*, 16(2):153–185, June 1984.

[GM92] J. Grant and J. Minker. The Impact of Logic Programming on Databases. *Communications of the ACM*, 35(3):66–81, March 1992.

[JCG+92] C.S. Jensen, J. Clifford, S.K. Gadia, A. Segev, and R.T. Snodgrass. A glossary of temporal database concepts. *ACM SIGMOD Record*, 21(3), September 1992.

[JS92] Christian S. Jensen and Richard Snodgrass. The TempIS project – Proposal for a data model for the temporal structured query language. TempIS Technical Report No. 37, Department of Computer Science, University of Arizona, Tucson, AZ, July 1992.

[KSW90] F. Kabanza, J.-M. Stévenne, and P. Wolper. Handling Infinite Temporal Data. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 392–403, Nashville, Tennessee, April 1990.

[KKR90] P.C. Kanellakis, G.M. Kuper, and P.Z. Revesz. Constraint Query Languages. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 299–313, Nashville, Tennessee, April 1990.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.

[LLPS91] G.M. Lohman, B. Lindsay, H. Pirahesh, and K.B. Schiefer. Extensions to Starburst: Objects, Types, Functions and Rules. *Communications of the ACM*, 34(10):94–109, October 1991.

[McK86] E. McKenzie. Bibliography: Temporal databases. *ACM SIGMOD Record*, 15(4):40–52, December 1986.

[NS92] Marc Niézette and Jean-Marc Stévenne. An efficient symbolic representation of periodic time. In *First International Conference on Information and Knowledge Management*, Baltimore, Maryland, November 1992.

[Rev90] P.Z. Revesz. A Closed Form for Datalog Queries with Integer Order. In *International Conference on Database Theory*, pages 187–201. Springer-Verlag, LNCS 470, 1990.

[SA85] Richard Snodgrass and Ilsoo Ahn. A taxonomy of time in databases. In *ACM-SIGMOD International Conference on Management of Data*, pages 236–246, Austin, Texas, May 1985.

[Sno87] Richard Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, 1987.

[Sno90] Richard Snodgrass. Temporal databases: Status and research directions. *ACM SIGMOD Record*, 19(4):83–89, December 1990.

[Soo91] Michael D. Soo. Bibliography on temporal databases. *ACM SIGMOD Record*, 20(1):14–23, March 1991.

[SS88] R. Stam and R. Snodgrass. A bibliography on temporal databases. *IEEE Database Engineering*, 7(4):231–139, December 1988.

[SK91] M. Stonebraker and G. Kemnitz. The POSTGRES Next-Generation Database Management System. *Communications of the ACM*, 34(10):78–92, October 1991.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

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

[Var82] M.Y. Vardi. The Complexity of Relational Query Languages. In *ACM SIGACT Symposium on Theory of Computing*, pages 137–146, 1982.

[vEK76] M.H. van Emden and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, 1976.