To appear in: Journal of Computer and System Sciences

Handling Infinite Temporal Data*†

F. Kabanza J-M. Stevenne P. Wolper
Université de Liège‡

January 25, 1995

Abstract

In this paper, we present a powerful framework for describing, storing, and reasoning about infinite temporal information. This framework is an extension of classical relational databases. It represents infinite temporal information by generalized tuples defined by linear repeating points and constraints on these points. We characterize the expressiveness of these generalized relations in terms of predicates definable in Presburger arithmetic. Next, we prove that relations formed from generalized tuples are closed under the operations of relational algebra and provide complexity results for the evaluation of first-order queries.

1 Introduction

Temporal information and temporal reasoning is currently of interest in several areas of computer science: artificial intelligence, concurrent program verification, and databases. However, the paradigms used for temporal reasoning in these three areas vary widely. In artificial intelligence, the focus is on expressing temporal information in a natural way. This has led researchers to concentrate on temporal frameworks based on intervals. Results in this area include work on the theory of intervals [All83, VK86, Lad88] and on interval based temporal logics [HS86, Ven88]. Unfortunately, even though the theory of intervals is itself decidable, interval based temporal logics are highly undecidable and thus probably impractical as reasoning tools.

In the field of concurrent programming, temporal logic has been recognized as a useful tool for dealing qualitatively with liveness properties. For instance, temporal logic easily expresses that something happens eventually or infinitely often. The logics used here are point based and are interpreted over infinite sequences. One interesting development in this area is "model-checking". In this approach, verifying a program is viewed as

*A preliminary version of this paper appears in the Proceedings of the Ninth Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Nashville, April 2-4, 1990.

†The following text presents research results of the Belgian National incentive-program for fundamental research in artificial intelligence initiated by the Belgian state - Prime Minister's Office - Science Policy Programming. The scientific responsibility is supported by its authors.

‡address: Institut Montéfiore, B28; 4000 Liège Sart-Tilman; Belgium.
Email: {kabanza,stevenne,wolper}@montefiore.ulg.ac.be

evaluating the truth of a temporal formula on a temporal structure representing the program [CES86, LP85, PZ86, VW86]. Note that model-checking is essentially a form of query evaluation on a special type of database.

In databases, the emphasis has been on representing and retrieving time information in the context of information systems [Tan86, Ari86, CW83]. With few exceptions [CI88], the temporal information represented is finite and can be incorporated into relational databases. Issues studied here are efficiency, expressiveness and convenience.

In this paper, we present a powerful framework for describing, storing, and reasoning about temporal information. This framework borrows concerns and concepts from all three areas outlined above. From artificial intelligence, we take the concern with intervals and expressive logics. Indeed, even if the theory of intervals and the theory of pairs of points coincide in most settings, interval *logics* are more expressive than point based logics [Ven88]. Moreover, we believe that it is often more convenient to describe temporal information in terms of intervals rather than points. (see Example 2.4 in Section 2 illustrating the need of an interval framework in a practical context like train schedules.)

From concurrent program verification, we borrow the concern with infinite and repeating temporal information. It is easy to argue that most people presently alive do not care about what will happen after the year 2090 and thus that finite temporal data is sufficient in applications more mundane than concurrent program verification. This line of reasoning ignores two facts. First, it is often difficult to set an arbitrary limit on the time period that will be considered. Second, even if such a limit exists, using methods that can handle infinite time can lead to a more compact and tractable representation. For instance, it is preferable to state that something happens every year forever than to state that it happens in 1989, 1990,1991,...,2090.

For interval temporal logics, the standard problems (satisfiability, validity,...) are highly intractable[1] [HS86]. This means that using such a logic for practical temporal reasoning is at best problematic. Alternative methods of reasoning are thus necessary. One possibility is reasoning by "evaluation" as is done in model-checking for concurrent programs. We adopt this approach and, since this type of reasoning is the standard paradigm of databases, we describe our temporal reasoning framework as a form of generalized database.

We view time as isomorphic to the integers and consider temporal predicates as defined on intervals, i.e. pairs of integers.[2] Ideally, we would want to be able to represent any interval predicate whether its extension is finite or infinite. This is of course impossible with finite representations since there are uncountably many interval predicates over the integers. We thus introduce a representation of a particular class of interval predicates. It is based on a generalized form of relation that includes two temporal attributes defined by what we name *linear repeating points* (points of the form $c + kn$) and by constraints on such points. This gives us a natural and convenient representation of many

[1] There are decidable interval temporal logics [AN88], but these are really point based logics in disguise, i.e. they are no more expressive than point based logics.

[2] Our decision to view intervals as pairs of points is motivated by the fact that it makes the description of the class of models we are interested in more natural. As was shown in [Lad88], suitable choices make the theory of pairs of points identical to the theory of intervals considered as basic objects defined in [All83].

infinite temporal predicates. For instance, all periodically repeating predicates (consider the activity of robots in a factory or the flight times at an airport,...) are immediately representable.

The rest of this paper is organized as follows. In the next section, we define our notion of temporal database and prove some expressiveness results. These results relate predicates definable in Presburger arithmetic to temporal predicates definable by generalized relations. Section 3 describes how each relational algebra operation can be performed on generalized databases and analyses the complexity of these operations. We distinguish between two types of complexity measures: *fixed-schema complexity* and *general complexity*. In fixed-schema complexity, we assume that the schema of the database is fixed and we measure the complexity of operations as a function of the number of tuples in the database. In general complexity, we assume that both the number of tuples and the size of the schema can vary. We prove that under fixed-schema complexity, all operations can be computed in PTIME. Furthermore, we prove that under general complexity, all operations except complementation (which requires exponential time) can be computed in PTIME and show that nonemptiness of generalized relations is decidable in PTIME whereas nonemptiness of complement is NP-complete. Finally, we define a first-order query language for databases of generalized relations and prove that evaluating queries can be done in PTIME under the data-complexity measure.

2 Infinite Temporal Data Bases

2.1 Definitions

The temporal databases we consider are collections of generalized relations, themselves collections of generalized tuples. A generalized relation includes temporal and nontemporal attributes. A generalized relation corresponding to an interval predicate will have two temporal arguments. However, for generality's sake and to preserve closure under various operations, we handle arbitrary numbers of temporal arguments[3]. A generalized tuple will assign data values to the nontemporal attributes and linear repeating points together with constraints to the temporal attributes. We now give precise definitions.

Definition 2.1 *A* **linear repeating point (lrp)** *is a set $\{x(n)\}$ of one or an infinite number of points of $\mathcal{Z}$ (the set of integers). The elements $x(n)$ of the lrp are defined by an expression of the form $x(n) = c + kn$, where the constants k and c are in $\mathcal{Z}$ and the variable n takes all values in $\mathcal{Z}$ from $-\infty$ to $+\infty$.*

Example 2.1 The *lrp* $3 + 5n$ represents the set $\{\ldots, -7, -2, 3, 8, 13, 18, 23, \ldots\}$ ∎

Constraints on the temporal attributes are conjunctions of atomic constraints. We distinguish between general and restricted constraints. *General constraints* are built from

[3]Generalized relations with more than two temporal arguments will usually only appear as intermediate steps in computations. For instance, as we will see, concatenating intervals is done by first identifying the common bound of the intervals, hence yielding a relation with three temporal arguments, and then projecting out that middle point.

Perform

0	1	Robot1	Task1	
$2+2n_1$	$4+2n_2$	Robot1	Task2	$X_1 = X_2 - 2 \ \wedge \ X_1 \geq -1$
$6+10n_1$	$7+10n_2$	Robot2	Task1	$X_1 = X_2 - 1 \ \wedge \ X_1 \geq 10$
$10n_1$	$3+10n_2$	Robot2	Task2	$X_1 = X_2 - 3$

Table 1: Example of generalized relation.

atomic constraints that are arbitrary linear equalities or inequalities between at most two temporal attributes. *Restricted constraints* are built from equalities and inequalities in which the coefficients of temporal attributes are 1. More specifically, if X_i and X_j are temporal attributes, restricted atomic constraints are of the following form:

$$X_i \leq X_j + a, \ X_i = X_j + a, \ X_i \leq (\geq) a, \ \text{or} \ X_i = a.$$

We can now define generalized tuples.

Definition 2.2 *Let $\mathcal{T}$ be the set of lrps*[4] *and $\mathcal{D}$ a set of nontemporal data values. A* **generalized tuple** *of temporal arity k and data arity l is an element of $\mathcal{T}^k \times \mathcal{D}^l$, together with constraints on the temporal components.*

Note that a generalized tuple can be viewed as defining a potentially infinite set of tuples, one for each possible combination of values of the temporal attributes.

Example 2.2

- The generalized tuple (2 temporal arguments) $[1, 1+2n] \ \wedge \ X_2 \geq 0$ represents the infinite set of tuples $\{[1,1],[1,3],[1,5],\ldots\}$.
- The generalized tuple $[3+2n_1, 5+2n_2] \wedge X_1 = X_2 - 2$ represents the infinite set of tuples $\{\ldots,[1,3],[3,5],[5,7],[7,9],\ldots\}$.

∎

Definition 2.3 *A* **generalized relation** *is a finite set of generalized tuples of the same schema.*

Note that limiting constraints to conjunctions is not really a restriction since a generalized tuple defined by disjunctive constraints can be split into several conjunctive generalized tuples.

Example 2.3 Table 1 is an example of a generalized relation representing the activities of robots. ∎

[4] we assume that the variable of each lrp is distinct

The following example motivates our use of relations with two temporal attributes that represent an interval.

Example 2.4 Suppose we want to represent the schedule of the trains going from Liège to Brussels. For example, there is a train leaving Liège at 7:02 and arriving in Brussels at 8:20 and another train leaving at 7:46 arriving at 8:50. This can be represented by the following relation which has temporal arity 2.

Train

`7:02`	`8:20`
`7:46`	`8:50`

With this representation the fact that during a certain period (7:46 to 8:20) there are two trains traveling on the same route does not cause any ambiguity. Now, if we were restricted to a point based temporal logic, we would have to use two predicates of temporal arity 1, for example "Leaving" and "Arriving".

Leaving
`7:02`
`7:46`

Arriving
`8:20`
`8:50`

But then, we would lost the relation between the starting and ending points of the intervals. For example, one could conclude that there is a train leaving at 7:46 and arriving at 8:20. A solution would be to add a non temporal predicates to the relations "Leaving" and "Arriving" that acts as a train identifier.

Leaving

`7:02`	Slow
`7:46`	Express

Arriving

`8:20`	Slow
`8:50`	Express

However, if we wanted to represent the fact that every hour, h, there is a train leaving Liège at $h:02$ and arriving in Brussels at $h+1:20$ and another train leaving at $h:46$ arriving at $h+1:50$ using repeating points we would obtain:

Leaving

$02 + \texttt{one\_hour} * n$	Slow
$46 + \texttt{one\_hour} * n$	Express

Arriving

$80 + \texttt{one\_hour} * n$	Slow
$110 + \texttt{one\_hour} * n$	Express

But this solution is incorrect since one can for instance conclude that there is a train leaving Liège at $h+1:46$ and arriving at Brussels at$h:50$! With two temporal attributes (intervals), the solution is simple:

Train

$02 + \texttt{one\_hour} * n_1$	$80 + \texttt{one\_hour} * n_2$	$X_1 = X_2 - 78$
$46 + \texttt{one\_hour} * n_1$	$110 + \texttt{one\_hour} * n_2$	$X_1 = X_2 - 64$

∎

2.2 Expressiveness

A natural question to ask about our generalized relations is: how expressive are they? The only way we can answer this question is to relate the expressiveness of generalized relations to that of other formalisms. To study expressiveness, we will concentrate on purely temporal predicates (no nontemporal arguments). We use the following definitions of expressibility by generalized relations.

Definition 2.3.1 *A k-ary predicate on the integers is* **lrp definable** *if its extension can be defined by a generalized relation with k temporal arguments using general constraints.*

Definition 2.3.2 *A k-ary predicate on the integers is* **weak lrp definable** *if its extension can be defined by a generalized relation with k temporal arguments using restricted constraints.*

The reference formalism to which we compare the expressiveness of generalized relations is Presburger arithmetic. It turns out that what can be defined in Presburger arithmetic almost coincides with what can be defined by generalized relations. This provides some theoretical justification to the intuitive fact that generalized relations are sufficiently expressive for most applications.

Definition 2.3.3 *A k ary predicate on the integers is* **Presburger definable** *if it is definable by a Presburger arithmetic formula with k free variables.*

The results are the following.

Theorem 2.1 *A unary predicate on the integers is weak lrp definable iff it is Presburger definable.*

Proof: Presburger arithmetic formulas are boolean combinations of the following basic Presburger formulas [Pre29, End72]:

$$\begin{cases} k_1 v & = & c, \\ k_1 v & < & c, \\ k_1 v & > & c, \\ k_1 v & \equiv_{k_2} & c,^{5} \end{cases}$$

where v is a variable (on Z) and where k_1, k_2 and c are constants (in Z). We will show (Section 3) that generalized relations are closed under union, intersection and complementation. Hence, to prove that unary Presburger predicates are representable by generalized relations with restricted constraints, it is sufficient to show that each of the basic formulas is expressible.

1. $k_1 v = c$. If $c' = c/k_1 \in Z$ then the corresponding relation is the tuple:

$$[c'],$$

   otherwise the relation is empty.

[5] $\equiv_{k_2}$ is equality modulo k_2

2. $k_1 v < c$ is equivalent to $k_1 v \leq c-1$. Let $c' = \lfloor (c-1)/k_1 \rfloor$. Then the corresponding relation is the tuple:
$$[n_1] \quad X_1 \leq c'.$$

3. $k_1 v > c$ is equivalent to $k_1 v \geq c+1$. Let $c' = \lceil (c+1)/k_1 \rceil$. Then the corresponding relation is the tuple:
$$[n_1] \quad X_1 \geq c'.$$

4. The formula $k_1 v \equiv_{k_2} c$ can be rewritten $k_1 v = k_2 n + c$ where n is a variable ranging on Z. The latter formula actually expresses the intersection of the two lrps: $k_1 v$ and $k_2 n + c$. If this intersection is empty there is no solution for v and the corresponding relation is empty. Otherwise, the intersection has the form (see Section 3.2): $kn + c'$ where $k = lcm(k_1, k_2)$. We can again rewrite our original formula as $k_1 v = kn + c'$, hence $v = (kn + c')/k_1 = k'n + c''$ with $k' = k/k_1$ and $c'' = c'/k_1$.[6] Finally, the relation corresponding to v is the tuple:
$$[k'n + c''].$$

∎

Theorem 2.2 *A binary predicate on the integers is lrp definable iff it is Presburger definable.*

Proof: To prove that binary Presburger predicates are representable by generalized relations, it is sufficient to show that each of the following basic Presburger formulas:

$$\begin{cases} k_1 v_1 & = & k_2 v_2 + c, \\ k_1 v_1 & < & k_2 v_2 + c, \\ k_1 v_1 & > & k_2 v_2 + c, \\ k_1 v_1 & \equiv_{k_3} & k_2 v_2 + c, \end{cases}$$

where v_1 and v_2 are variables (on Z) and where k_1, k_2, k_3 and c are constants (in Z) is expressible.

1. The relations corresponding to $k_1 v_1 = k_2 v_2 + c$, $k_1 v_1 < k_2 v_2 + c$ and $k_1 v_1 > k_2 v_2 + c$ are respectively:

X_1	X_2	
n_1	n_2	$k_1 X_1 = k_2 X_2 + c$,

X_1	X_2	
n_1	n_2	$k_1 X_1 \leq k_2 X_2 + c - 1$, and

X_1	X_2	
n_1	n_2	$k_1 X_1 \geq k_2 X_2 + c + 1$.

[6] Note that c'' is an integer (take $n = 0$, c' must intersect the first lrp, hence it is a multiple of k_1).

2. For $k_1 v_1 \equiv_{k_3} k_2 v_2 + c$, let let $x_1 = k_1 v_1$ and $x_2 = k_2 v_2$. The formula then becomes: $x_1 \equiv_{k_3} x_2 + c$. This formula represents the set of pairs (x_{1_i}, x_{2_i}) in which each x_{1_i} is "((a multiple of k_3) plus c)" distant from x_{2_i}. The relation representing all these pairs is:

X_1	X_2
$k_3 n_1 + c$	$k_3 n_2$
$k_3 n_1 + c + 1$	$k_3 n_2 + 1$
$k_3 n_1 + c + 2$	$k_3 n_2 + 2$
$\vdots$	$\vdots$
$k_3 n_1 + c + (k_3 - 1)$	$k_3 n_2 + (k_3 - 1)$.

Now, we must also take into account that x_1 and x_2 are respectively multiple of k_1 and k_2. This can be done by intersecting the above relation with the relation:

X_1	X_2
$k_1 n_1$	$k_2 n_2$.

This intersection (see Subsection 3.2, page 9) yields a new relation in which all tuples have the form:

$$[k_1(k' n_1 + c'_i), k_2(k'' n_2 + c''_i)].$$

Finally, since $X_1 = k_1 v_1$ and $X_2 = k_2 v_2$, the relation corresponding to (v_1, v_2) is the relation formed from the latter by dividing the first and the second column by k_1 and k_2 respectively:

$$\bigcup_i \{[(k' n_1 + c'_i), (k'' n_2 + c''_i)]\}.$$

∎

Note that since the constraints we use involve at most two temporal attributes, n-ary Presburger definable predicates are not always lrp definable for $n > 2$. Thus, our formalism can only represent a fraction of what is expressible in Presburger arithmetic, but this fraction is the one that is significant for point-based and interval-based temporal properties.

Preburger arithmetic is a simple, powerful and natural formalism for describing sets of integers. The fact that our formalism has the same expressive power as Presburger arithmetic for unary and binary predicates can be viewed as an indication that the class of sets we can represent is a robust natural one. This is about as much as can be said about expressiveness from a theoretical point of view. More detailed conclusions would have to be drawn from the pragmatics of representing temporal data.

3 Operations on generalized relations

From now on, we will assume that we are dealing with databases in which all tuples are defined by restricted constraints. As we will see, this assumption is necessary for the computation of the projection of a relation. In what follows, we only describe how the temporal attributes of relations are handled in operations. The data component of the tuples is handled as in a traditional relational database.

3.1 Union

The union of two generalized relations is constructed by simply merging these relations. In practice, one would also attempt to eliminate the redundancies that might appear between the tuples of the merged relation. We do not consider this problem.

3.2 Intersection

3.2.1 Intersection of two linear repeating points

Let $c_1 + n_1 k_1$ and $c_2 + n_2 k_2$ be two lprs and let k be *the least common multiple(lcm) of* $\{|k_1|, |k_2|\}$. If there is some $j \in [0, (k/k_1) - 1]$ such that

$$\frac{c_1 - c_2}{k_2} + \frac{jk_1}{k_2} = i \in Z$$

then the intersection is nonempty and

$$\begin{array}{l} c_1 + n_1 k_1 \, \cap \, c_2 + n_2 k_2 = c + nk \text{ with} \\ c = c_2 + ik_2 \end{array}$$

otherwise,

$$c_1 + n_1 k_1 \, \cap \, c_2 + n_2 k_2 = \emptyset$$

Note that the problem of finding a j such that

$$\frac{c_1 - c_2}{k_2} + \frac{jk_1}{k_2} = i \in Z$$

is equivalent to the problem of finding j such that

$$(k_1 j + (c_1 - c_2)) \bmod k_2 = 0. \tag{1}$$

This can be solved as follows. Let g be the greatest common divisor of k_1 and k_2. If $d = (c_1 - c_2)/g$ is not an integer then (1) has no solution. Otherwise j satisfies

$$\begin{array}{l} (k'_1 j + d) \bmod k'_2 = 0 \text{ with} \\ k'_1 = k_1/g \text{ and } k'_2 = k_2/g. \end{array}$$

Hence

$$j = (-d \times (k'^{-1}_1 \bmod k'_2)) \bmod k'_2$$

which is easily computed since[7] $k'^{-1}_1 \bmod k'_2$ can be obtained by an extension of Euclid's algorithm for computing the greatest common divisor requiring an $O(\ln \max(k'_1, k'_2))$ time computation [Den82].

[7] $k'^{-1}_1 \bmod k'_2$ is the inverse of k_1 modulo k_2, i.e. an x such that $xk_1 \bmod k_2 = 1$.

3.2.2 Intersection of two generalized relations

Consider two generalized relations:

$$r_1 = \left\{ \bigcup_{i=1}^{l} t_{1i} \right\} \text{ and } r_2 = \left\{ \bigcup_{j=1}^{m} t_{2j} \right\}$$

By elementary set theory, we have that:

$$r_1 \cap r_2 = \bigcup_{i,j} (t_{1i} \cap t_{2j})$$

Therefore, we only need to show how to perform the intersection of two generalized tuples. The intersection $t_1 \cap t_2$ is the generalized tuple whose components are the intersection of the corresponding components of t_1 and t_2 and whose constraints are the union of those of t_1 and t_2.

Example 3.1 To compute

$$\begin{array}{ll} & [2n_1 + 1\,, 3n_2 - 4] \quad X_1 \leq X_2 \,\wedge\, 3 \leq X_1 \\ \cap & [5n_3 \qquad\;\, , 5n_4 + 2] \quad X_1 = X_2 - 2. \end{array}$$

we first compute

$$\begin{array}{ll} 2n_1 + 1 \cap 5n_3 & = 10n + 5 \text{ and} \\ 3n_2 - 4 \cap 5n_4 + 2 & = 15n' + 2. \end{array}$$

Then we add the constraints of both relations to get the final result:

$$[10n + 5, 15n' + 2] \quad X_1 \leq X_2 \,\wedge\, 3 \leq X_1 \,\wedge\, X_1 = X_2 - 2.$$

∎

3.3 Subtraction

3.3.1 Subtraction of two linear repeating points

We compute the subtraction of two repeating points A and B in the case where A includes B. If this is not the case, B can be replaced by $A \cap B$ since $A - B = A - (A \cap B)$.

Now, suppose A is $c_1 + k_1 n_1$ and B is $c_2 + k_2 n_2$. Because B is included in A, we must have that $k_2 = lcm(|k_1|, |k_2|)$. The subtraction is then the set of repeating points of the form

$$k_2 n + c_2 + k_1 j \quad \text{for} \quad j \in \left[1, \frac{k_2 - k_1}{k_1}\right]$$

3.3.2 Subtraction of two generalized relations

Consider two relations r_1 and r_2 as in section 3.2.2. Their subtraction can be expressed as:

$$r_1 - r_2 = \bigcup_{i=1}^{l} (\ldots((t_{1i} - t_{21}) - t_{22}) - \ldots) - t_{2m}).$$

Hence, we only need to give a procedure that performs the subtraction of two generalized tuples.

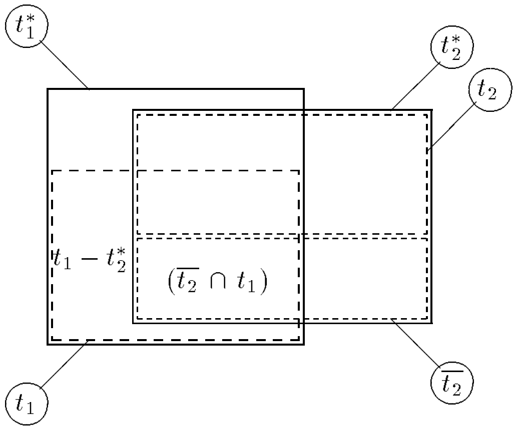

Figure 1: Set view of subtraction.

3.3.3 Subtraction of generalized tuples

Definition 3.1 *The free extension t^* of a tuple t is t without its constraints.*

Subtraction of two free extensions

First, let $Sub(l_1, l_2)$ be the set of lrps representing the subtraction of the lrp l_2 from the lrp l_1. Let $t_1^* = [l_{11}, \ldots, l_{1n}]$ and $t_2^* = [l_{21}, \ldots, l_{2n}]$ be two free extensions. Let $t_3^* = [l_{31}, \ldots, l_{3n}]$ be their intersection. Then, $t_1^* - t_2^* = t_1^* - t_3^*$.

Each tuple in t_1^* which has at least one component, say X_i, in $Sub(l_{1i}, l_{3i})$ is in $t_1^* - t_3^*$. Hence, the subtraction $t_1^* - t_3^*$ is the set of tuples:

$$t_1^* - t_3^* = \bigcup_{i=1}^{n} \bigcup_{j} \{[l_{11}, \ldots, l_j \in Sub(l_{1i}, l_{3i}), \ldots, l_{1n}]\}\,.$$

Subtraction of two generalized tuples

Figure 1 shows that $t_1 - t_2 = (t_1 - t_2^*) \cup (\overline{t_2} \cap t_1)$ where $\overline{t_2} = t_2^* - t_2$. The subtraction is achieved accordingly:

- First compute $r_3 = t_1 - t_2^*$ by adding t_1's constraints to $r_3^* = t_1^* - t_2^*$.
- Then compute $\overline{t_2} \cap t_1$:
  - If there are no constraints associated to t_2 ($t_2 = t_2^*$) then $\overline{t_2} = \emptyset$
  - Otherwise: Add t_2's negated constraints to t_2^* to obtain $\overline{t_2}$ (disjunctive constraints, introduced by the negation, are eliminated by splitting the tuple into several tuples with conjunctive constraints).
  - Let $r_4 = \overline{t_2} \cap t_1$.
- Finally, $t_1 - t_2$ is $r_3 \cup r_4$.

3.4 Projection

Obviously, the projection of a relation can be computed separately for each tuple. In a classical database, to perform the projection of one relation on some of its columns it is sufficient to forsake the other columns. Here, this can not be handled so simply. Before forsaking a temporal column, its constraints have to be projected. This means that we have to eliminate variables from an integer constraint system. For real constraints, this can be done by a simple algorithm that eliminates variables by computing linear combinations of constraints (see for instance [Duf74]).

Unfortunately, as illustrated in Figure 2, such a method is not suitable when the variables are integers. This figure represents the geometric situation described by the following tuple.

$$[4n_1 + 3, 8n_2 + 1] \ \wedge \ X_1 \geq X_2 \ \wedge \ X_1 \leq X_2 + 5 \ \wedge \ X_2 \geq 2$$

It is easy to see that 3, 7, 15, 23... are in the real projection on X_1 even though there are no corresponding points in the tuple.

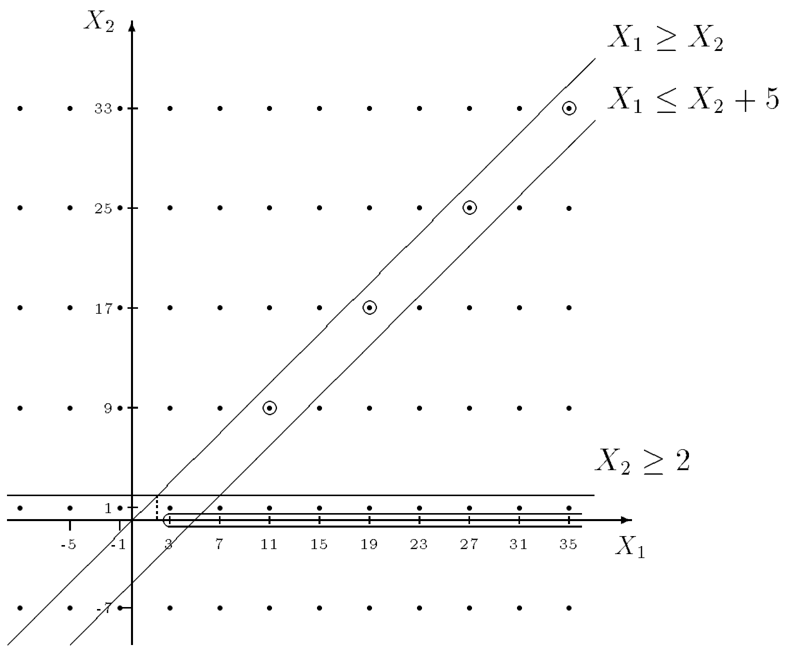

Figure 2: Example of projection problem.

Such problems can be overcome by exploiting the particular form of restricted constraints. The key is to transform tuples in a "*normal form*".

Definition 3.2 *A tuple is said to be in* **normal form** *if there is a unique k such that the value of each attribute X_i is a lrp of the form*

$$\begin{cases} c_i, \ or \\ c_i + kn_i \end{cases}$$

and if all the constraints have one of the forms

$$\begin{cases} kn_i = a, \\ kn_i \leq a, \\ kn_i \geq a, \\ kn_i = kn_j + a, \text{ or} \\ kn_i \leq kn_j + a \end{cases}$$

with $a/k \in \mathcal{Z}$.

The following theorem states that, for a system of constraints in normal form, computing the projection by eliminating variables using linear combinations of constraints is a correct procedure.

Theorem 3.1 *Let T be a tuple in normal form of schema $[X_1, \ldots, X_n]$. Let $\Pi_{[X_2,\ldots,X_n]}T$ be the projection of the tuple T computed by the projection algorithm for constraints on the reals. Let $(n_2, \ldots, n_m)$ be an element of $\Pi_{[X_2,\ldots,X_n]}T$ such that $n_i \in \mathcal{Z}$ for $i = 2, \ldots, m$. Then, there is at least one $n_1 \in \mathcal{Z}$ such that $(n_1, n_2, \ldots, n_m)$ is an element of the tuple T.*

Proof: Let C be the system of constraints associated to the tuple T and suppose there is no such n_1. Because of the correctness of the projection algorithm for constraints on the reals [Duf74], there is a real x such that $(x, n_2, \ldots, n_m)$ verifies C.

The special form of the constraints ensures that the constraint system defines a convex region of space. Thus there exist noninteger n_{Min} and n_{Max} such that $n_{Max} - n_{Min} < 1$ and $\forall x \mid n_{Min} \leq x \leq n_{Max}$: $(x, n_2, \ldots, n_m)$ verifies the system C.

Moreover, n_{Min} and n_{Max} must satisfy equality for at least one constraint. This means that, for example, n_{Min} must verify either $kn_{Min} = a$ or $kn_{Min} = kn_i + a$ with n_i and a/k integers. Let $a' = a/k$, n_{Min} must verify either $n_{Min} = a'$ or $n_{Min} = n_i + a'$ with n_i and a' integers. This is impossible since n_{Min} and n_{Max} are not integers. ∎

Let us now establish that a tuple with restricted constraints can always be normalized. We first prove a lemma.

Lemma 3.1 *An lrp of period k can always be replaced by a set of lrp of period k', with $k' = ck$, $c \in N_0$.*

Proof: Any lrp $a + kn$ is trivially equivalent to the following set of lrps of period $k' = ck$:

$$\begin{cases} & & a & + & k'n \\ k & + & a & + & k'n \\ 2k & + & a & + & k'n \\ & & \vdots & & \\ (c-1)k & + & a & + & k'n \end{cases}$$

∎

Theorem 3.2 *A tuple t can always be replaced by an equivalent set of tuples in normal form.*

Proof: Let

$$t = [c_1 + k_1 n_1, \ldots, c_m + k_m n_m] \wedge C$$

and let k be the *lcm* of $\{k_i \mid i \in [1,m] \wedge k_i \neq 0\}$. The equivalent set of normalized tuples can then be obtained by the following steps.

1. For each *lrp* $c_i + k_i n_i$ with $k_i \neq 0$, let L_i the equivalent set of lrps of period k given by lemma 3.1.

2. The cross product of the set of lrps just computed forms the new equivalent set of free extensions. If we associate the constraints of the original tuple to each new free extensions we obtain an equivalent representation of t:

$$t \equiv \{[l_1 \times \ldots \times l_m] \wedge C \mid l_i \in L_i\}$$

3. In each new tuple, replace the occurrences of X_i in the constraints C by the corresponding lrp: $c'_i + kn'_i$.

4. All tuples with a constraint $kn_i = p$ or $kn_i = kn_j + q$ where p/k and q/k are not integers are eliminated since these constraints cannot be satisfied.

5. Finally, each constraint of the form $kn_i \leq p$, $kn_i \geq q$, and $kn_i \leq kn_j + r$ can respectively be replaced by the equivalent constraint $kn_i \leq \lfloor p/k \rfloor k$, $kn_i \geq \lceil q/k \rceil k$, and $kn_i \leq kn_j + \lfloor r/k \rfloor k$.

Obviously after these five steps the tuple has been replaced by an equivalent set of tuples in normal form. ∎

Example 3.2 Consider the relation containing only the tuple represented by Figure 2:

r		
$4n_1 + 3$	$8n_2 + 1$	$X_1 \geq X_2 \wedge X_1 \leq X_2 + 5 \wedge X_2 \geq 2$

After the two first steps of the normalization, we obtain:

r		
$8n_1 + 3$	$8n_2 + 1$	$X_1 \geq X_2 \wedge X_1 \leq X_2 + 5 \wedge X_2 \geq 2$
$8n_1 + 7$	$8n_2 + 1$	$X_1 \geq X_2 \wedge X_1 \leq X_2 + 5 \wedge X_2 \geq 2$

After the third step we have:

r		
$8n_1 + 3$	$8n_2 + 1$	$8n_1 + 2 \geq 8n_2 \wedge 8n_1 - 3 \leq 8n_2 \wedge 8n_2 \geq 1$
$8n_1 + 7$	$8n_2 + 1$	$8n_1 + 6 \geq 8n_2 \wedge 8n_1 + 1 \leq 8n_2 \wedge 8n_2 \geq 1$

The normalized relation is then

r		
$8n_1 + 3$	$8n_2 + 1$	$8n_1 \geq 8n_2 \wedge 8n_1 \leq 8n_2 \wedge 8n_2 \geq 8$
$8n_1 + 7$	$8n_2 + 1$	$8n_1 \geq 8n_2 \wedge 8n_1 + 8 \leq 8n_2 \wedge 8n_2 \geq 8$

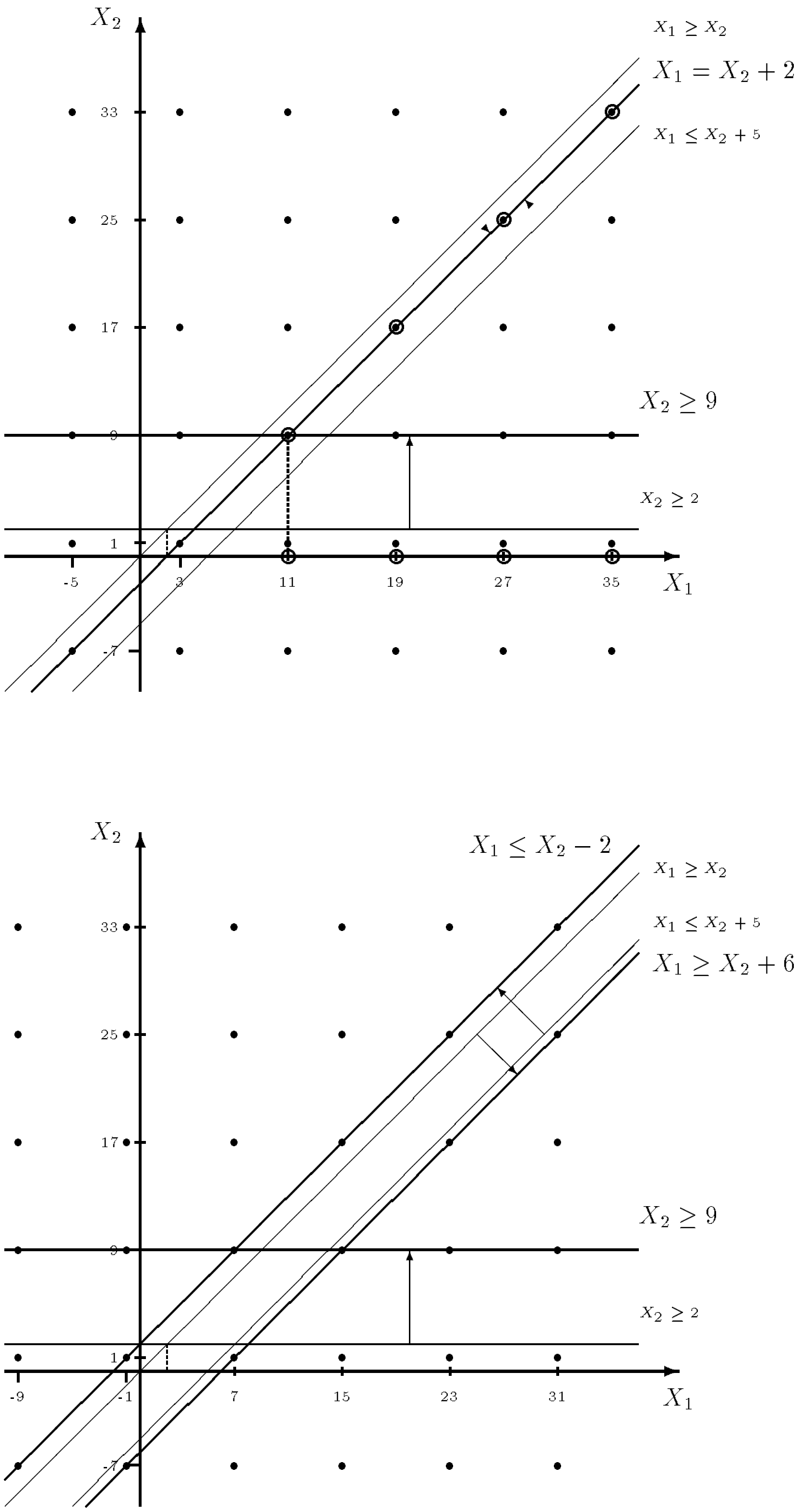

Figure 3: Example of normalization.

or

r		
$8n_1+3$	$8n_2+1$	$X_1 \geq X_2+2 \ \wedge \ X_1 \leq X_2+2 \ \wedge \ X_2 \geq 9$
$8n_1+7$	$8n_2+1$	$X_1 \geq X_2+6 \ \wedge \ X_1 \leq X_2-2 \ \wedge \ X_2 \geq 9$

Figure 3 represents step 5 of the normalization. It consists in shifting the constraint lines in order to force them to go through the repeating points.

Finally we could, for example, eliminate the second column which yields

$\Pi_{X_1} r$	
$8n_1+3$	$8n_1 \geq 8$

or

$\Pi_{X_1} r$	
$8n_1+3$	$X_1 \geq 11$

∎

Since our projection method requires normalization, it might seem excessively costly. It is of course possible to construct cases where it is indeed so, but a number of factors contribute to limiting the impact of normalization. First, normalization is done individually for each tuple, this implies for instance that adding a new tuple to a normalized relation requires only the normalization of that tuple. Furthermore, for computing a projection, it is often sufficient to do a partial normalization. For instance, if attribute i is the one that has to be projected out, then only column i and columns sharing a constraints with column i have to be normalized.

3.5 Selection

If X_i and X_j are not temporal then the selection is computed as in a classical database. Otherwise, simply add the corresponding constraint to each generalized tuple.

3.6 Cross product

The cross product $(r_1 \times r_2)$ is computed by forming a new relation composed of all combinations of tuples from r_1 and r_2 (with their respective constraints).

3.7 Join

The join $r_1 \bowtie r_2$ of two relations is computed by the union of the joins of their tuples:

$$\bigcup_{i,j} t_{1i} \bowtie t_{2j}.$$

To compute the join of two generalized tuples, we first compute the intersection of the tuples restricted to their common attributes. Finally, the "join tuple" is obtained by assembling the intersection of the common columns with the other columns of both tuples and by adding the corresponding constraints.

3.8 The complexity of operations

Let us now examine the complexity of a number of decision problems and operations on generalized relations. We will only do the analysis for normalized relations. Clearly, if the least common multiple of the initial periods is large, normalization can imply a substantial increase in the size of the database. However, this will only be the case if the periods appearing in the database are not closely related. One can expect such an unfavorable situation to the exception rather than the norm.

We first consider the usual operations on relations. We distinguish between two types of complexity measures: *fixed-schema complexity* and *general complexity*. In fixed-schema complexity, we assume that the schema of the database is fixed and we measure the complexity of operations as a function of the number of tuples in the database. In general complexity, we assume that both the number of tuples and the size of the schema can vary.

Under fixed-schema complexity, all operations can be computed in PTIME.

Theorem 3.3 *The projection, selection, union, intersection, subtraction and join of (a) normalized generalized relation(s) can be computed in PTIME under the fixed-schema complexity measure.*

Proof: See appendix A. ∎

For general complexity, all operations except difference can be computed in PTIME. Difference requires exponential time.

Theorem 3.4 *Under the general complexity measure, the projection, selection, union, intersection, and join of (a) normalized generalized relation(s) can be computed in PTIME. The difference of two normalized generalized relations can be computed in EXPTIME.*

Proof: See appendix A. ∎

Next, consider a decision problem, namely that of determining if a generalized relation is nonempty.

Theorem 3.5 *The problem of determining if a normalized generalized relation is non-empty (represents at least one tuple) can be solved in PTIME under the general complexity measure (and hence also the fixed-schema complexity measure).*

Proof: Consider a generalized relation r of schema $[X_1, \ldots, X_m]$. To see if the relation is not empty, it is sufficient to eliminated $(m-1)$ columns by computing $(m-1)$ projections and then to check if there is a unary tuple such that its constraints $(X = a \wedge X \geq b \wedge X \leq c)$ are satisfiable. Since this last problem can obviously be solved in PTIME, and since projection is a PTIME operation(Theorem 3.4), the problem is in PTIME. ∎

Difference (and hence negation) cannot be computed in PTIME under the general complexity measure because the negation of a conjunctive constraint is disjunctive, and recombining the various disjuncts into conjunctions can yield an exponential increase in the number of tuples. The following theorem substantiates the intractability of complementation.

Theorem 3.6 *The nonemptiness of complement problem for generalized relations is NP-complete under the general complexity measure.*

Proof: We have to prove that given a relation r, determining if $\neg r$ is nonempty is a NP-complete problem.

1. *The problem is in NP*

   First notice that if the relation $\neg r$ is non-empty, there is a tuple which is not in the relation r with values bounded by a function of the constants appearing in r (the periods and offsets of the lrps and the constants appearing in constraints). Let a non-deterministic machine guess this tuple. To verify if it is not in r it suffices to compute its intersection with the relation r (this can be done in PTIME) and to verify that the resulting relation is empty (this can also be done in PTIME). Hence the problem is in NP.

2. *The problem is NP-complete*

   We reduce 3-SAT to this problem. Let (U, C) be an instance of 3-SAT, with $U = \{u_1, \ldots, u_m\}$ a set of literals and $C = \{c_1, \ldots, c_l\}$ a set of clauses. To each literal, u_i in U, we associate one column X_i of a generalized relation r. To each clause, c_j in C, we associate a generalized tuple in r. The free extension of this tuple is

   $$[n_1, n_2, \ldots, n_m].$$

   The constraint part of each tuple is defined as follows:

   $$\begin{array}{rll} \forall u_i \in c & \text{add the constraint} & X_i < 0, \\ \forall \neg u_i \in c & \text{add the constraint} & X_i \geq 0. \end{array}$$

   Omitting its free extension, the relation r is of the form:

   $$\begin{array}{ccccc} (X_{11}\,\theta_{11}\,0 & \wedge & X_{12}\,\theta_{12}\,0 & \wedge & X_{13}\,\theta_{13}\,0)\vee \\ (X_{21}\,\theta_{21}\,0 & \wedge & X_{22}\,\theta_{22}\,0 & \wedge & X_{23}\,\theta_{23}\,0)\vee \\ & & \vdots & & \\ (X_{l1}\,\theta_{l1}\,0 & \wedge & X_{l2}\,\theta_{l2}\,0 & \wedge & X_{l3}\,\theta_{l3}\,0) \end{array}$$

   where X_{jk} is the column corresponding to the k^{th} literal of the clause c_j and where θ_{ij} is "$<$" or "$\geq$". The relation $\neg r$ is the relation corresponding to:

   $$\begin{array}{ccccc} (X_{11}\,\eta_{11}\,0 & \vee & X_{12}\,\eta_{12}\,0 & \vee & X_{13}\,\eta_{13}\,0)\wedge \\ (X_{21}\,\eta_{21}\,0 & \vee & X_{22}\,\eta_{22}\,0 & \vee & X_{23}\,\eta_{23}\,0)\wedge \\ & & \vdots & & \\ (X_{l1}\,\eta_{l1}\,0 & \vee & X_{l2}\,\eta_{l2}\,0 & \vee & X_{l3}\,\eta_{l3}\,0) \end{array}$$

   where $\eta_{ij} = \neg\theta_{ij}$. Obviously the relation $\neg r$ corresponds to the conjunction of clauses of the 3-SAT problem. Hence, "finding a truth assignment for the u_i's in U in order to satisfy the conjunction of all the clauses in C" is polynomially reduced to "finding a tuple of positive and/or negative integers which belongs to the relation $\neg r$".

∎

Polynomial complexity seems to imply tractability. Is this really so for operations on generalized relations? Several factors have to be taken into consideration. A first look at the results in Appendix A seems to indicate that we are in a case where polynomial complexity is merely an illusion of tractability. Indeed, the complexity of computing the complement of a relation is a polynomial whose degree is m^2 where m is the number of attributes of the relation. However, m is the number of temporal attributes which in most applications should stay small (1 or 2). Moreover, this frightening complexity comes from the problem of taking the negation of boolean combinations of constraints. This problem is intrinsically an exponential problem (we obtained a polynomial upper bound by restricting ourselves to the fixed schema case), but it is a problem of which many instances can in practice be solved with reasonable resources.

In conclusion, as such, our polynomial complexity results do not indicate tractability. Moreover, even if the degree of the polynomial sometimes appears excessive, they should not be taken as an indication of intractability: they are only worst case upper bounds. The question of whether (and with which restrictions) our framework is usable in practice is still widely open. Our reading of the complexity results we have given is that the situation is not hopeless.

4 The temporal query language

The temporal query language we use is a two-sorted first-order logic. One sort is temporal points (interpreted over the integers), the other is generic. We will refer to the number of temporal arguments of a predicate as its **temporal arity** and to its number of nontemporal arguments of a predicate as its **data arity**.

Our language will include one interpreted predicate of temporal arity 2 and data arity 0 ($\leq$) and any number of uninterpreted predicates. As we intend to deal with intervals, the uninterpreted predicates have a temporal arity of 2 (or possibly 0) and an arbitrary data arity[8]. In the spirit of what is done in relational databases, we do not use function symbols on the generic sort. On the temporal sort, we will use one interpreted function: the successor function. We allow arbitrary quantification on both temporal and nontemporal variables.

Interestingly, this type of two sorted logic has also recently been resuscitated as a reasonable language to use for temporal reasoning in AI [BTK89]. If we consider its restriction to the temporal sort, it is the first-order dyadic theory of linear orders with one successor (i.e. $(\mathcal{P}, Z, 0, S, <)$ where $\mathcal{P}$ is a set of binary predicates). As is shown in [Ven88], this logic is actually more expressive than the modal interval logics of [Sho86] which is itself more expressive than linear temporal logic (LTL) based on points. We thus have a very expressive language.

4.1 Syntax

The syntax of our language is standard first-order. It includes:

[8]Actually predicates of arbitrary temporal arity could also be used.

- temporal and non-temporal variables,
- uninterpreted predicates $p(t_1, \ldots, t_\alpha, x_1, \ldots, x_\beta)$ of temporal arity α and data arity β,
- Boolean connectives,
- existential and universal quantification on both temporal and non-temporal, variables.
- the predicate "$\leq$" of temporal arity 2 and
- the successor function "+1".

Example 4.1 The following formula of our language refers to the data of Example 2.3. It expresses the fact that there is a robot x and and a robot y such that, if x is performing task2 during an interval of length at least 5, then y is not performing any task during any part of that interval.

$$\begin{aligned}
&\exists x \exists y \exists t_1 \exists t_2 \forall t_3 \forall t_4 \forall z \\
&\quad (\text{Perform}(t_1, t_2, x, \text{task2}) \ \wedge \ t_1 \leq t_3 \leq t_4 \leq t_2 \ \wedge \ t_1 + 5 \leq t_2) \supset \\
&\quad \neg \text{Perform}(t_3, t_4, y, z)
\end{aligned}$$

∎

4.2 Semantics

Our language can be given a semantics in the usual way over interpretations $\mathcal{I} = \langle \mathcal{Z}, \mathcal{D}, \Pi \rangle$ consisting of:

- The set $\mathcal{Z}$ of integers as the domain on which temporal points are interpreted,
- The set $\mathcal{D}$ on which non temporal objects are interpreted,
- An interpretation function Π mapping each temporal variable to $\mathcal{Z}$, each non temporal variable to $\mathcal{D}$ and each (k, l)-ary predicate to $2^{\mathcal{Z}^k \times \mathcal{D}^l}$.

The predicate $\leq$ and the successor function are given their usual interpretation over the integers. It is easy to see that our temporal databases define structures of the form above. Hence the evaluation of a query on generalized database has an obvious semantics and can be performed using the relational algebra operations.

4.3 The complexity of query evaluation

We consider the data-complexity [Var82] of evaluating yes/no queries on a database. If the query is fixed, the schema of the database must also be fixed and we can use the results we obtained for fixed-schema complexity. Since all operations are polynomial under that measure, we have the following.

Theorem 4.1 *Determining the truth of yes/no queries on a normalized generalized database can be done in PTIME under the data-complexity measure.*

5 Conclusions and comparison with other work

We view the main contributions of our paper as defining a powerful form of temporal database and showing how it can be used with a simple and expressive query language. We consider that being able to handle temporal predicates with infinite extensions is essential and believe it can be useful in many applications. We also believe that the usefulness of our framework is not limited to traditional database applications. For instance, it provides a tool for temporal reasoning in AI that is in the spirit of "vivid reasoning" [Lev86, EBBK89].

There is little research on manipulating infinite temporal objects in the context of databases. The most notable is probably [CI88]. There, *unary* temporal predicates are defined in a deductive layer in top of the database. By contrast, we incorporate infinite predicates with arbitrary arity directly into the database. This makes operations on temporal predicates easier and does not exclude the eventual use of a deductive layer. Also, in the framework of Chomicki and Imielinski the processing of temporal queries is in general intractable. It is however possible to restrict the form of the temporal rules used in that framework in order to obtain polynomial time query processing [Cho90]. Interestingly, the necessary restrictions lead to rules that define information easily representable by linear repeating points.

Generalized databases where some attributes are defined by variables and constraints on these variables have already appeared in the context of incomplete information [Gra89]. An important difference with our work is that we describe a single database with infinite information rather than a set of possible databases. Moreover, the use of linear repeating points is specific to temporal information. There is nevertheless an unsurprising similarity between some of the complexity bounds obtained in both contexts.

Constraint languages and the idea of combining logic based tools with constraint resolution algorithms have appeared in other contexts. For instance [Sch88] classifies constraint languages and defines a class of constraints similar to our restricted constraints. *Constraint Logic Programming* [JL87] combines logic programming with constraint resolution systems. The goal is to combine the expressive power of logic programming with the efficiency of algebraic treatment. Finally, [KKR90] also introduces the idea of constraints as an intrinsic part of databases. The main difference between their work and ours is that they consider different domains and different classes of constraints. The use of repeating points is specific to our framework and makes it particularly suitable for temporal information.

Another approach to developing efficient temporal reasoning systems is to consider temporal logic restricted to Horn clauses. This yields an analogue of logic programming in the field of temporal reasoning. One example of such a framework is TEMPLOG [AM87, Bau89]. However, TEMPLOG is more similar to the framework developed in [CI88] than to ours. It handles point based predicates rather than interval predicates. Moreover, it is designed to handle relative timing information rather than quantitative timing information.

In the artificial intelligence literature, there is a large body of work on intervals [All83, AK83, All84, AH85]. However, not much is proposed as usable methods for handling interval predicates with infinite extensions. Also, in most case only relative timing information is considered. The work on interval temporal logics is limited to intractability and expressiveness results [HS86, Ven88].

Appendices

A Complexity of relational algebra operations

In what follows, we consider both fixed-schema and general complexity. We denote the number of tuples in the database by N and the number of columns in a table by m. For simplicity (but without loss of generality), we assume that all constraints are inequalities. Since the conjunction of two constraints of the same type can be reduced to the most constraining one[9], there are at most $m(m+1)$ different restricted constraints between m variables. We will assume that each tuple in the database has this maximum number of constraints and hence that the size of the database is $O(m^2N)$.

For each operation, we will first discuss the cost on a normalized database and then briefly examine the situation for nonnormalized databases.

A.1 Normalization

Each lrp of period k_i is replaced by a set of k/k_i lrps, where k is the least common multiple of all the k_i's. Each tuple containing m lrp's of respective periods $k_{x_1}, k_{x_2}, \ldots, k_{x_m}$) will thus be replaced by $(k/k_{x_1} \times k/k_{x_2} \times \cdots \times k/k_{x_m})$ tuples. In the worst case

$$k = \prod_{i=1}^{n} k_i$$

where n is the number of different periods. The maximum number of different periods in a tuple is m and in the database it is mN.

A.2 Union

The fixed-schema complexity is $O(N)$ and the general complexity is $O(m^2N)$ (we do not consider the problem of eliminating redundant information). The resulting number of tuples is $O(N)$. Note that these results also hold for unnormalized databases.

A.3 Intersection

Given two lrps, $c_1 + kn_1$ and $c_2 + kn_2$,

$$c_1 + kn_1 \cap c_2 + kn_2 = \begin{cases} c_1 + kn_1 & \text{if } c_1 \equiv_k c_2 \\ \emptyset & \text{otherwise} \end{cases}$$

Where $\equiv_k$ is equality modulo k.

[9]For example, $X_1 \leq X_2 + 4 \wedge X_1 \leq X_2 - 5 \equiv X_1 \leq X_2 - 5$

Given two tuples, their intersection results from the intersection of their m respective lrps. The intersection of two generalized relations with N tuples will yield at most a relation with N^2 tuples in $O(m^2N^2)$ time. However, only pairs of tuples with the same free extension will have a nonempty intersection. It seems quite reasonable to assume that after normalization the distribution of the different lrps is essentially uniform. This implies that the number of different free extensions is k^m. Thus, for two well distributed relations, the intersection will result in N^2/k^m tuples. Hence, we can say that the fixed-schema complexity of intersection is $O(N^2)$ and more precisely varies from N^2/k^m to N^2 depending on the uniformity of the distribution of the lrps in the relations. The general complexity is $O(m^2N)$. For nonnormalized databases computing each intersection of lrps requires time $O(\ln k)$. In that case, the complexity results above have to be multiplied by a factor $\ln k$.

A.4 Projection

The projection of a normalized relation containing N tuples is a relation containing at most N tuples. Each tuple can be computed in time $O(m(m-1))$. The fixed-schema complexity of projection is thus $O(N)$ and its general complexity is $O(m^2N)$.

Each tuple of a nonnormalized relation must be normalized before a projection can be computed. The cost of this normalization is the multiplicative factor : k^m.

A.5 Cross-Product and Join

Obviously, the fixed-schema complexity of these operations is $O(N^2)$ whereas their general complexity is $O(m^2N^2)$. For nonnormalized databases, the complexity of cross-product remains the same; the complexity of join is multiplied by a factor $\ln k$.

A.6 Negation

Before considering subtraction, let us examine the problem of computing the negation of a relation r, i.e. computing

$$[n_1, \ldots, n_m] - r.$$

Let k be the period of the normalized relation r. Negation can be computed by building a relation containing all free extensions of period k (of which there are k^m) and negating the constraints of the free extensions that appear in r. Let us first establish a bound on the number of tuples obtained by negating the constraints corresponding to one free extension.

Theorem A.1 *Let r be a normalized relation with m columns and N tuples. Suppose that all the tuples in r have the same free extension t^*, then, the relation $t^* - r$ is a relation containing at most $(N+1)^{m(m+1)}$ tuples.*

Proof: The required computation is to take the negation of a disjunctive formula of the form:

$$\bigvee_{i=1}^{N} (a_{i1} \wedge a_{i2} \wedge \ldots \wedge a_{il}) \tag{2}$$

where $l = m(m+1)$.

This negation is the conjunctive formula:

$$\bigwedge_{i=1}^{N} (\neg a_{i1} \vee \neg a_{i2} \vee \ldots \vee \neg a_{il}) \tag{3}$$

Once Formula (3) is recombined into disjunctive normal form, each of its disjuncts is a conjunction that can be reduced to containing only one constraint of each type (keep the strongest). Since there are $m(m+1)$ types of constraints and at most $N+1$ constraint of each type (one per tuple plus TRUE for missing constraints), the result will be a disjunction containing at most $(N+1)^{m(m+1)}$ disjuncts. Hence, the relation $t^* - r$ will contain at most $(N+1)^{m(m+1)}$ tuples. ∎

The theorem above proves that, for m fixed, the size of the negation of a relation with one free extension is polynomial. However, it does not show that this negation can be computed in polynomial time since the outlined procedure requires that the formula corresponding to the negation of the constraints be exponentially expanded before being reduced. It is actually possible to modify the procedure in such a way that it operates in polynomial time. The simplest way to achieve this is to convert the conjunction of Formula (3) into disjunctive normal form in an incremental way. Precisely, one uses the following recursion

$$\begin{aligned} \bigwedge_{i=1}^{N_1} \quad & (\neg a_{i1} \vee \neg a_{i2} \vee \ldots \vee \neg a_{il}) = \\ & \bigwedge_{i=2}^{N_1} (\neg a_{i1} \vee \neg a_{i2} \vee \ldots \vee \neg a_{il}) \\ & \wedge (\neg a_{11} \vee \neg a_{12} \vee \ldots \vee \neg a_{1l}) \end{aligned} \tag{4}$$

and applies the reduction described above at each step. If one does this, it is clear that for any k, the size of

$$\bigwedge_{i=k}^{N_1} (\neg a_{i1} \vee \neg a_{i2} \vee \ldots \vee \neg a_{il}) \tag{5}$$

is at most $(N+1)^{m(m+1)}$ tuples. Each step can be computed in time $O(l(N+1)^{m(m+1)})$ and since $l = m(m+1)$, the complexity of the computation is thus $O(m^2(N+1)^{m(m+1)})$. Note that using a more elaborate computation scheme, it is possible to reduce the time needed for the computation to $O((N+1)^{m(m+1)})$.

Since the negation of a relation r is obtained by considering all k^m possible free extensions and negating the constraints of those appearing in r, it can be computed in time $O(k^m + N(N+1)^{m(m+1)})$.

A.7 Subtraction

Given two lrps, $c_1 + kn_1$ and $c_2 + kn_2$,

$$(c_1 + kn_1) - (c_2 + kn_2) = \begin{cases} \emptyset & if\ c_1 \equiv_k c_2 \\ c_1 + kn & \text{otherwise.} \end{cases}$$

Recall that for two tuples t_1 and t_2, $t_1 - t_2 = (t_1 - t_2^*) \cup (\overline{t_2} \cap t_1)$.

- $t_1 - t_2^*$ results in one or zero tuple depending on whether $t_1^* = t_2^*$ or not.
- $\overline{t_2}$ results in $m(m+1)$ tuples.
- $\overline{t_2} \cap t_1$ result in $m(m+1)$ or zero tuple(s) depending on whether $t_1^* = t_2^*$ or not.

Consider two relations with the same free extension: r_1 containing N_1 tuples and r_2 containing N_2 tuples. Their subtraction is given by:

$$r_1 - r_2 = \bigcup_{i=1}^{N_1} \underbrace{(\cdots \underbrace{(\underbrace{(t_{1i} - t_{21})}_{m(m+1)\text{ tuples}} - t_{22})}_{(m(m+1))^2\text{ tuples}} \cdots) - t_{2N_2})}_{(m(m+1))^{N_2}\text{ tuples}} .$$

This shows that subtraction leads to a relation containing at most $N_1(m(m+1))^{N_2}$ tuples.

Fortunately, using an argument similar to the one we developed above (see A.6) for the complexity of negation, one can easily prove that this very big set of tuples contains at most $N_1(N_2 + N_1 + 1)^{m(m+1)}$ different tuples since the $r_1 - r_2$ will only contain tuples with a free extension appearing in r_1. Hence, if we make sure that redundant tuples are suppressed each time an intersection is computed (this can for instance be achieved by keeping relations sorted), the fixed-schema complexity of subtraction is polynomial whereas its general complexity is exponential: $O(N^{m(m+1)+1})$.

If we start with a nonnormalized database, the subtraction procedure produces a relation in which all lprs have the same period. Hence an implicit normalization is done and the complexity results have to be multiplied by the factor k^m.

B Summary of complexity results

Tables 2 and 3 summarize the complexity results. N represents the number of tuples in the database, m the number of columns and k the lcm of all the periods (k_i) appearing in the database. The emptiness of complement results are obtained from the complexity of negation. Indeed, to check that the complement of a relation is nonempty, it is sufficient to check that its complement contains a tuple without constraints or a tuple with satisfiable constraints.

References

[AH85] J.F. Allen and P.J. Hayes. A common-sense theory of time. In *9th IJCAI*, pages 528–531, Los Angeles, 1985.

[AK83] J.F. Allen and J.F. Koomen. Planning using a temporal world model. In A. Bundy, editor, *8th IJCAI*, pages 741–747, 1983.

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

	normalized relation	
Complexity type	Fixed-schema	General
Union	$O(N)$	$O(m^2 N)$
Cross-product	$O(N^2)$	$O(m^2 N^2)$
Intersection	$O(N^2)$	$O(m^2 N^2)$
Join	$O(N^2)$	$O(m^2 N^2)$
Projection	$O(N)$	$O(m^2 N)$
Emptiness of a relation	$O(N)$	$O(m^3 N)$
Negation	$O(N^c)$	$O(k^m + N^{(c'm^2)})$
Emptiness of the complement relation	$O(N^c)$	$O(N^{(c'm^2)})$

Table 2:

<table>
<tr><td></td><td colspan="2">normalized relation</td></tr>
<tr><td>Complexity type</td><td>Fixed-schema</td><td>General</td></tr>
<tr><td>Union</td><td rowspan="6" colspan="2">computable in PTIME</td></tr>
<tr><td>Cross-product</td></tr>
<tr><td>Intersection</td></tr>
<tr><td>Join</td></tr>
<tr><td>Projection</td></tr>
<tr><td>Emptiness of a relation</td></tr>
<tr><td>Negation</td><td rowspan="2"></td><td>computable in EXPTIME</td></tr>
<tr><td>Emptiness of the complement relation</td><td>NP-Complete</td></tr>
</table>

Table 3:

[All84] James F. Allen. Toward a general theory of action and time. *Artificial Intelligence*, 23:123–154, July 1984.

[AM87] Martin Abadi and Zohar Manna. Temporal Logic Programming. In IEEE Computer Society, editor, *Symposium on Logic Programming*, pages 4–16, September 1987.

[AN88] A.A. Aaby and K.T. Narayana. Propositional Temporal Interval Logic is PSPACE complete. In *9th International Conference on Automated Deduction*, volume 310, pages 218–237. Lecture Notes in Computer Science, Springer-Verlag, May 1988.

[Ari86] G. Ariav. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, 11:499–528, 1986.

[Bau89] Marianne Baudinet. Temporal Logic Programming is Complete and Expressive. *16th Annual ACM Symposium on Principles of Programming Languages*, pages 267–280, January 1989.

[BTK89] Fahiem Bacchus, Josh Tenenberg, and Johannes A. Koomen. A non-reified temporal logic. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceding of the first international conference on Principles of Knowledge Representation and Reasoning*, pages 2–10, Toronto Ontario Canada, may 1989. Morgan Kofmann.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[Cho90] Jan Chomicki. Polynomial time query processing in temporal deductive databases. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 379–391, Nashville Tennessee, 1990.

[CI88] Jan Chomicki and Tomasz Imielinski. Temporal deductive databases and infinite objects. In *Proceedings of the 7th ACM Symposium on Principles of Database Systems*, pages 61–73, Austin Texas, 1988.

[CW83] J. Clifford and D.S. Warren. Formal Semantic for Time in Database. *ACM Transactions on Database Systems*, 8(2):214–254, 1983.

[Den82] D. E. Denning. *Cryptography and Data Security*, pages 43–44. Addison-Wesley Publishing Company, 1982.

[Duf74] R.J. Duffin. On fourier's analysis of linear inequality systems. *Mathematical Programming Study*, 1:71–95, 1974.

[EBBK89] David W. Etherington, Alex Borgida, Ronald J. Brachman, and Henry Kautz. Vivid knowledge and tractable reasoning. In *IJCAI-89*, volume 2, pages 1146–1152, aug 1989.

[End72] Herbert B. Enderton. *A Mathematical Introduction to Logic*, pages 188–192. Academic Press, 1972.

[Gra89] Gösta Grahne. Horn tables - an efficient tool for handling incomplete information in databases. In *Synposium on Principles of Database Systems*, pages 75–82. Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART, Mar 1989.

[HS86] Joseph Y. Halpern and Yoav Shoham. A Propositionnal Modal Logic of Time Intervals. *IEEE*, pages 279–292, 1986.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. *Proceeding of ACM symposium on Principles of Programming Languages*, pages 111–119, 1987.

[KKR90] P. Kanelakis, G.M. Kuper, and P.Z. Revesz. Constraint query languages. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, pages 299–313, Nashville Tennessee, 1990.

[Lad88] Peter B. Ladkin. First-order Constraint Satisfaction for Time Intervals. In *AAAI-88*, volume 2, pages 512–517, August 1988.

[Lev86] H.J. Levesque. Making beleivers out of computers. *Artificial Intelligence*, 30(1):81–108, October 1986. (originaly given as the "computers and thaught" lecture at IJCAI 85).

[LP85] O. Lichtenstein and A. Pnueli. Checking that Finite State Concurrent Programs Satisfy their Linear Specification. In *Proceedings of the Twelfth ACM Symposium on the Principles of Programming Languages*, pages 97–107, New Orleans, Louisiana, January 1985.

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchen die addition als einzige Operation hervortritt. In *Comptes Rendus, I. Congrès des Mathématiques des Pays Slaves*, pages 192–201, 395, Warsaw, 1929.

[PZ86] A. Pnueli and L. Zuck. Probabilistic verification by tableaux. In *Proc. 1st Symp. on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[Sch88] Albrecht Schmiedel. Temporal constraint networks. KIT-Report 69, Technical University of Berlin, November 1988.

[Sho86] Yoav Shoham. *Reasoning about Change: Time and Causation from the Standpoint of Artificial intelligence.* PhD thesis, Yale University, Computer Science Department, New Haven, CT, 1986.

[Tan86] A.U. Tansel. Adding Time Dimension to Relational Model and Extending Relational Algebra. *Informations Systems*, pages 343–355, 1986.

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

[Ven88] Yde Venema. Expressiveness and Completeness of an Interval Tense Logic. Technical report, University of Amsterdam, February 1988.

[VK86] Marc Vilain and Henry Kautz. Constraints propagation algorithms for temporal reasoning. In *AAAI-86*, volume 1, pages 377–382. Fith National Conference on Artifitial Intelligenge, Morgan Kaufmann, August 1986.

[VW86] Moshe Y. Vardi and Pierre Wolper. Automata-Theoretic Techniques for Modal Logics of Programs . *Journal of Computer and System Sciences*, 32(2):183–221, April 1986.