Handling Infinite Temporal Data (Extended Abstract)

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

November 24, 2005

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 problematic 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 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 an information system context [Tan86, Ari86, CW83]. With few exceptions [CI88], the temporal information represented is finite and can be in-

*Email: u502802@BLIULG11.bitnet
†address: Institut Montéfiore, B28; 4000 Liège Sart-Tilman; Belgium.

corporated 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.

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 [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. 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 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 infinite temporal predicates. For instance, all periodically repeating predicates are immediately representable.

We then turn to decision problems and operations on generalized relations. We show that nonemptiness of generalized relations is decidable in PTIME whereas nonemptiness of complement is NP-complete. Concerning operations, we prove that generalized relations are closed under the standard operations of relational algebra. All these operations (except complementation) can be computed in PTIME. The closure properties of generalized relations enable us to prove some

expressiveness results. These results relate predicates definable in Presburger arithmetic to temporal predicates definable by generalized relations.

Finally, we consider the complexity of evaluating first-order queries on a database of generalized relations. We prove that evaluating positive existential queries can be done in PTIME whereas evaluating general queries is NP-hard.

2 Infinite Temporal Data Bases

2.1 Definition

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. 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 from* $-\infty$ *to* $+\infty$ *in* $\mathcal{Z}$*.*

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 atomic constraints that are arbitrary linear equalities or inequalities between 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 a set of lrps 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 The following table is an example of generalized relation.

	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	$X_1 = X_2 - 3$

∎

2.2 Operations

Let us now examine a number of decision problems and operations on generalized relations. In what follows, we only consider operations on generalized relations with restricted constraints. One difficulty in manipulating generalized relations is that different tuples may involve lrps of different periods. It is however possible to normalize generalized relations to a common period, namely the least common multiple of the periods involved. The basic idea is that for any $c \in N$, a lrp $a + kn$ of period k can be replaced by a set of lrps of period $k' = ck$, namely

$$\left\{\begin{array}{lllll} & & a & + & k'n \\ a & + & k & + & k'n \\ a & + & 2k & + & k'n \\ & & \vdots & & \\ a & + & (c-1)k & + & k'n. \end{array}\right.$$

Given this, it is easy to see how to normalize generalized relations. An important point is that the normalization procedure preserves restricted constraints (i.e.,

constraints where the coefficients of temporal attributes are all 1). Clearly, if the least common multiple of the periods is large, normalization can imply a large increase in the size of the database. There are however reasons to believe that this will not be the typical behavior in most applications. Indeed, periods will often be closely related.

From now on, let us only consider normalized relations (with restricted constraints). We first consider a decision problem, namely that of determining if a generalized relation is nonempty.

Theorem 2.1 *The problem of determining if a normalized generalized relation is nonempty (represents at least one tuple) can be solved in PTIME.*

Next we consider the usual operations on relations. We have the following theorem.

Theorem 2.2 *The projection, selection, union, intersection, and join of (a) normalized generalized relation(s) can be computed in PTIME.*

Projection deserves a few words of explanation. Computing the projection of linear constraints on the integers is notably difficult, though on the reals it is straightforward. This is where our assumption that all the coefficients of the linear constraints are 1 is useful: it enables us to compute the projection with an algorithm similar to the one used on the reals.

For complementation, the situation is not as good as for the other operations.

Theorem 2.3 *The complementation of a normalized generalized relation can be computed in EXPTIME.*

The difficulty of computing complementation is due to the fact that the negation of a conjunctive constraint is disjunctive and that various disjuncts have to be recombined into conjunctions. The following theorem substantiates the intractability of complementation.

Theorem 2.4 *The nonemptiness of complement problem for generalized relations is NP-complete.*

2.3 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 and 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 and 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 seem 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 2.5 *A unary predicate on the integers is weak lrp definable iff it is Presburger definable.*

Theorem 2.6 *A binary predicate on the integers is lrp definable iff it is Presburger definable.*

3 The complexity of Query Evaluation

We now turn to the problem of evaluating queries on the temporal database we have just defined. We use a very natural query language which is a two sorted first-order logic. In formulas, we distinguish between temporal and nontemporal arguments.

Example 3.1 A typical formula of this language looks like

$$\exists x \exists y \forall z \exists t_1 \exists t_2 \forall t_3 \forall t_4 (\text{Perform}(t_1, t_2, x, \text{task2}) \wedge t_1 \leq t_3 \leq t_4 \leq t_2 \supset \neg \text{Perform}(t_3, t_4, y, z))$$

∎

Note that when this language is used with interval (binary) temporal predicates, it is at least as expressive as most of the interval logic presently being used [Ven88]. It is for instance more expressive than the modal interval temporal logic of [HS86]. It has also recently been resuscitated as a reasonable language to use for temporal reasoning in AI [BTK89]. Also, from the point of deciding

satisfiability, it is a highly intractable logic [HS86]. As far as evaluating formulas of this logic on our temporal database, the situation is much more favorable.

We consider the data-complexity [Var82] of evaluating yes/no queries on the database. As we have seen in the previous section, the only problematic operation is complementation. Hence if we avoid complementation and universal quantification (which involves projection and complementation), queries can be evaluated efficiently.

Theorem 3.1 *Determining the truth of yes/no positive existential queries on a normalized generalized database can be done in PTIME using the data-complexity measure.*

If we consider unrestricted queries, the complexity of complementation creeps in.

Theorem 3.2 *Determining the truth of unrestricted yes/no queries on a normalized generalized database can be done in EXPTIME and is hard for NP under the data-complexity measure.*

4 Conclusions and Comparison with Other Works

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 the possibility of handling temporal predicates with infinite extensions as 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.

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 bases 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.

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].

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.

[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.

[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.

[CI88] Jan Chomicki and Tomasz Imielinsky. 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.

[EBBK89] David W. Etherington, Alex Borgida, Ronald J. Brachman, and Henry Kautz. Vivid knowledge and tractable reasoning. Submitted to IJCAI 89, 1989.

[Gra89] Gösta Grahne. Horn tables - an efficient tool for handling incomplete information in databases. In *Proceedings of the 8th ACM*, pages 75–82. Synposium on Principles of Database Systems, March 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.

[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 thought" 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 Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[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.

[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.