# Temporal Queries for Active Database Support

(position paper)

**N. H. Gehani**
AT&T Bell Laboratories
nhg@research.att.com

**H. V. Jagadish**
AT&T Bell Laboratories
jag@research.att.com

**Inderpal Singh Mumick**
AT&T Bell Laboratories
mumick@research.att.com

**Oded Shmueli**
Technion - Israel Institute of Technology
oshmu@cs.technion.ac.il

June 1993

**Abstract**

An active database monitors events (such as access, insert, delete, and update of tuples/ objects, and invocation of methods on objects). Each event potentially changes the database state. Applications may wish to react to sequences of events and database states satisfying certain properties. Specification of such sequences can be viewed as a formulation of temporal queries. Further, monitoring such sequences, or equivalently, the evaluation of such temporal queries, must be efficient, and must not require storage of the entire database history.

In this paper, we present a language for specifying temporal queries that are of interest in an active database, and for which we can hope to develop efficient evaluation techniques.

## 1 Introduction

Of late, there has been a surge of interest in active databases [DBB+88, MD89, BM91, SSU91, GJ91, LLPS91, SK91, WF90]. In an active database, a trigger fires when an event of interest happens and some condition is satisfied.

An event is usually interpreted as a single activity. In object-oriented databases, for example, events are related to object manipulation actions such as creation, deletion, and update or access by an object method (member function). Similarly, in a relational database, events are related to actions such as insert, delete, and update. Events can also be associated with time, for example, clock ticks, and the recording of the passage of a day, an hour, a second, or some other time unit. Most work on active databases has focussed on the trigger firing mechanism and the execution of the triggered action. However, recent work [DHL91, CM91, Ch92, GJS92a] has recognized the importance of relating multiple events occurring at different times, and has argued that the specification and detection of such *temporal events* is important for an active database.

We believe that the above broader definition implies that an event be viewed as a "composite activity", that is the end result of a sequence of activities on the database, and the result of the database states resulting after each such activity.

We further believe that events must have attributes. For instance, an insert event has information regarding the specific relation (or object collection) updated as well as the specific tuple (or object) inserted. Such

information can be considered an attribute of the event. In addition, event attributes may include system-level information such as transaction id, user id, and time. In object-oriented systems, method invocation events may carry as attributes the parameters with which the respective methods are invoked. The database state existing at the time of occurrence of the event can be represented as an attribute of the event.

A temporal event comprises a sequence of constituent basic events, where the exact nature of the sequence is given by a temporal predicate. The attributes of a temporal event are derived from the attributes of the constituent basic events. The temporal event thus takes the form of a temporal query on successive states of the database as captured by the attributes of individual events. The attributes in the temporal event can be used to (1) specify parameters for the action part of a trigger, and (2) in condition predicates to restrict the specification and firing of the trigger.

The difficulty in dealing with attributes for temporal events is that not all of the constituent events occur simultaneously. Thus attributes of events that have occurred in the past must be remembered. Clearly, it is impractical to remember the entire history of the system from start-up time, noting each event and the attributes with which it occurred. It is thus critical to define a language of temporal queries such that the temporal events expressible in the language can be computed without remembering the entire history of the database.

In this position paper, we propose a language to express temporal events that we believe will be efficiently computable. A computation using finite automata for temporal events without attributes was presented in [GJS92b], and computations using extended automata for a large class of temporal events with attributes is discussed in [JMS92, JMS93].

# 2 Temporal Event Expressions - Events without Attributes

An "event" is a happening of interest. Events, including composite events, happen instantaneously at specific points in time. *Basic events*, such as `before insert` or `after delete` occur at points in time just before an insertion into a relation, or just after a deletion from a relation. *Composite events* are defined to occur at the same instant as the last event where the composite event occurs.

Event specification must start with a set of *basic* events, such as the ones mentioned above, which are supported by the database system. Basic events can be combined to create *composite* events using logical operators and special event specification operators. We discuss this in the next section.

## 2.1 Syntax and Semantics

An *event occurrence* (informally referred to as an event) is a tuple of the form *(basic event, event-identifier)*. Event-identifiers (eids) are used to define a total ordering, denoted by $<$, on event occurrences. An example of an event identifier is a time-stamp specifying the time at which the basic event occurred. Two event occurrences $e_1$ and $e_2$ refer to the same event occurrence if their eids are identical.

An *event history*, or simply a *history*, is a finite set of event occurrences in which no two event occurrences have the same event identifier. When this set is empty, the history is called the *null* history. The event occurrences in a history are sometimes referred to as *points*.

An *event expression* $E$, which specifies a basic or composite event, is a mapping from (domain) histories to (range) histories:

$$E : histories \rightarrow histories$$

The result of applying an event expression to a history $h$ is itself a history which contains the events in $h$ at which the events specified by $E$ take place. These intuitive notions are formalized below.

Let $E$ be an event expression. $E[h]$ denotes the application of $E$ to history $h$. It is always the case that $E[h] \subset h$. We say that $E$ *takes place* at event $e$ in $h$ iff $e \in E[h]$. An event occurrence $e \in h$ *satisfies* expression $E$ iff $e \in E[h]$; $E$ is said to be *satisfied* by $e$.

An event occurrence $e_1$ takes place after (before) an event occurrence $e_2$ if the eid of $e_1$ is larger (smaller) than the eid of $e_2$.

An *event expression* is formed using basic events and the operators (connectives) described below. An event expression can be any basic event $a$, or an expression formed using the operators ||, ! (not), *relative*, and linear recursion. The semantics of event expressions are defined as follows ($E$ and $F$ are used to denote event expressions):

1. $E[null] = null$ for any event $E$, where $null$ is the empty history.
2. $a[h]$, where $a$ is a basic event, is the maximal subset of $h$ composed of event occurrences of the form $(a,\ eid)$.
3. $(E \mid\mid F)[h] \equiv E[h] \cup F[h]$. (That is, an event occurrence satisfies $(E \mid\mid F)[h]$ if the event occurrence satisfies either $E[h]$ or $F[h]$.)
4. $(!E)[h] = (h - E[h])$.
5. $relative(E,\ F)[h]$ are the event occurrences in $h$ at which $F$ is satisfied assuming that the history started immediately following *some* event occurrence in $h$ at which $E$ takes place.

   Formally, $relative(E,\ F)[h]$ is defined as follows. Let $E^i[h]$ be the $i^{\text{th}}$ event occurrences in $E[h]$; let $h_i$ be obtained from $h$ by deleting all event occurrences whose eids are less than or equal to the eid of $E^i[h]$. Then

   $relative(E,\ F)[h] = \bigcup_i F[h_i]$, where $i$ ranges from 1 to the cardinality of $E[h]$.
6. Linear Recursion: An expression $E$ may be defined as

   $$E \equiv E_0 \ \texttt{||}\ \texttt{relative(}E\texttt{,}L\texttt{)}\ \texttt{||}\ \texttt{relative(}R\texttt{,}E\texttt{)},$$

   where one of the recursive disjuncts may be dropped. The expression $E[h]$ is TRUE iff either $E_0[h]$ is TRUE, or if there exists a partitioning of the history into two histories such that one of the other two disjuncts is satisfied (recursively, as they both include $E$).

**EXAMPLE 2.1** Let $E \wedge F$ denote that both (composite) events $E$ and $F$ occur at the same point in the history. We can express conjunction as

$$E \wedge F = !(!E \vee !F).$$

Let *any* denote the disjunction of all the basic events. We wish to define an operator $prior(E,\ F)$, which specifies that an event $F$ takes place after an event $E$ has taken place. Note that the basic events composing $E$ and $F$ may overlap.

We can express *prior* as

$$prior(E,\ F) = relative(E,\ any) \wedge F$$

□

**EXAMPLE 2.2** The United State Federal Reserve Board raises and lowers a key interest rate, called the discount rate, to control inflation and economic growth. Three or more successive discount rate cuts (`D`) without an intervening discount rate increase (`I`) is a rare phenomenon and is of interest to the financial community. Many other events can occur, for example, the prime rate may be cut and the stock market can crash, but these events do not interest us here. Our problem is to write an event expression that is satisfied by such cuts in the discount rate.

Here is an example history (Figure 1) with the dots marking the events in the history with the discount rate cut events labeled by `D` (decrease) and increases labeled by `I` (increase):

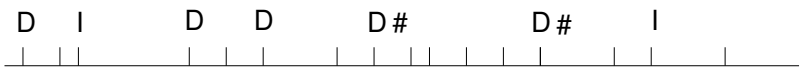


Figure 1: Discount rate cut

The composite event of interest occurs at the last two `D` events (marked with #).

We will now give an event expression that specifies a composite event satisfied when three or more successive discount rate cut events `D` take place without an intervening rate increase event `I`. We specify this composite event in steps. First, the event expression

```
prior(I, D)
```

specifies `D` events that are preceded by an `I` event. Expression

```
!prior(I, D)
```

specifies all events except the occurrences of `D` that are preceded by `I`. Expression

```
!prior(I, D) ∧ D
```

specifies `D` events that are not preceded by an `I` event.
The expression

```
relative(D, !prior(I, D) ∧ D)
```

specifies a `D` event followed eventually by another `D` event with no intervening `I` events. This expression gives us a pair of `D` events with no intervening `I` events. Note that in this case, the **`relative`** operator is used to look at the history starting after a `D` event.
Finally, the event that we are interested in can be specified as

```
relative(relative(D, !prior(I, D)∧ D), !prior(I, D) ∧ D)
```

The outermost **`relative`** finds another `D` without a preceding `I` giving us three `D` events without an intervening `I` event. □

Event expressions can be related to regular expressions. Let an event expression *accept* a history $h$ if the last event in history $h$ appears in $E[h]$. We define *∉-regular expressions.* A regular expression $G$ is said to be a *∉-regular expression* if (1) $G$ does not contain the empty string $\epsilon$, and (2) $G$ does not use $*$ but may use $+$, where $A^+ = A \cup AA \cup AAA \ldots$.

**Theorem 2.1 (Properties of Event Expressions):**

1. *Every regular expressions $E$ can be written as $\epsilon \cup E'$ or $E'$, where $E'$ is a* $\not\epsilon$-regular expression. *(This is a "folk theorem".)*

2. *For a* $\not\epsilon$-regular expression $E$, *there exists an event expression $F$ denoting the same regular language. Similarly, if $F$ is a sequence expression, there exists a* $\not\epsilon$-regular expression *denoting the same regular language. Thus, sequence expressions denote the regular languages not containing the empty sequence. (By explicit constructions.)*

3. *The given set of sequence operators is the minimal set required to obtain property 2 above. (this is a non-trivial result.)*

□

# 3 Temporal Event Expressions with Attributes

Every basic and composite event has a fixed set of `typed` attributes. The attributes of a basic event could be a part of the event itself (stock price), or the attributes could be any tuple or set of tuples extracted from the database at the time the event occurs, or the attributes could be the result of a function executed at the time the event occurs (ex. clock time). The attributes of a composite event must be derived from attributes of the component event sub-expressions. Each attribute must have a well-specified, though possibly infinite, domain.

An *event expression with attributes* $E(\bar{X})$, where $\bar{X}$ are the free variables in $E$, when evaluated against a sequence $h$, returns a set of tuples for attributes $\bar{X}$ for each point in history $h$ that satisfies the event expression $E(\bar{X})$. This set is empty at points that do not satisfy $E(\bar{X})$. We will define satisfiability of an expression for a history with attributes in a formal manner in this section.

Once attributes are available, predicates on these attributes may be defined. For instance, consider the event history:

$$a(1)\ a(2)\ a(8)\ a(4)\ b(5)$$

The event expression $E(X,Y)$ $\equiv$ `relative(`$a(X)$`,` $b(Y)$`) & (`$Y > X$`)`, is satisfied when the event $b(5)$ occurs, once on account of each of three of the four $a$ events (all except the third). Thus the event expression $E(X,Y)$ evaluates the relation {(1,5),(2,5),(4,5)} at the last event in the history.

We now define inductively, via generic examples, when an expression with attributes is satisfied for a history with attributes. Thereby, we define the relation returned by each event expression.

**Basic Events** The expression $E(\bar{X}) = a(\bar{X})$ is satisfied by a history if `a` is the last event in it. The relation output is the singleton containing a tuple whose values are the values of the $\bar{X}$ attributes in $a(\bar{X})$.

**Disjunction** The event expression $E(\bar{X}) = E_1(\bar{X})$ || $E_2(\bar{X})$ is permitted iff $E_1$ and $E_2$ have the same signature (number and type of attributes). These attributes must be given identical variable names when used in the above expression. The expression $E(\bar{X})$ is satisfied by a history if either of the expressions $E_1(\bar{X})$ or $E_2(\bar{X})$ is satisfied. The set of attribute values of $E(\bar{X})$ is the union of the attribute values for which $E_1(\bar{X})$ or $E_2(\bar{X})$ are satisfied.

**Negation** The event $!a$, for some primitive event $a$, occurs in the event history at the point that an event $b! = a$ occurs. The attributes of event $b$ will in general differ in number and type from those in event $a$. We interpret the attributes of an expression using ! with respect to some universe. We require attributes to be typed, and each type has a well-defined domain. (This domain could be infinite). The cartesian product of the respective domains defines the universe with respect to which negation is applied to a tuple of attributes.

The expression $E(X) = !F(X)$ is thus equivalent to the expression $E(X) \equiv !F(X)$ `&` $dom(X)$. At every point $p$ in the history, the event $E(X)$ is satisfied for an attribute value $X$, if $X$ is in the domain of $E$'s attribute, and if the expression $F(X)$ is not satisfied.

**Conjunction** The event expression $E(\bar{X}, \bar{Y}, \bar{Z}) = E_1(\bar{X}, \bar{Y}) \wedge E_2(\bar{Y}, \bar{Z})$ is satisfied by a sequence $h$ for the set of tuples $\{(\bar{x}, \bar{y}, \bar{z})\}$, if the expressions $E_1(\bar{X}, \bar{Y})$ and $E_2(\bar{Y}, \bar{Z})$ are satisfied by $h$ for tuples $(\bar{x}, \bar{y})$ and $(\bar{y}, \bar{z})$, respectively. Each of the vectors $\bar{X}, \bar{Y}$, or $\bar{Z}$ is optional.

**Relative** The event expression $E(\bar{X}, \bar{Y}, \bar{Z}) =$ `relative(`$E_1(\bar{X}, \bar{Y})$`,` $E_2(\bar{Y}, \bar{Z})$`)` is satisfied by a history $h$ for a tuple $(\bar{x}, \bar{y}, \bar{z})$ iff the history $h$ can be expressed as a concatenation of two histories $h_1$ and $h_2$ such that expression $E_1(\bar{X}, \bar{Y})$ is satisfied by $h_1$ for the value $(\bar{x}, \bar{y})$ (among others) and the expression $E_2(\bar{Y}, \bar{Z})$ is satisfied by $h_2$ for the value $(\bar{y}, \bar{z})$.

**Existential quantification** The event expression $E(\bar{X}) = (\exists \bar{Y})(E_1(\bar{X}, \bar{Y}))$ has the attribute $\bar{X}$, and is satisfied for a particular $\bar{X}$ value if there is a $\bar{Y}$ value for which the expression $E_1(\bar{X}, \bar{Y})$ is satisfied.

**EXAMPLE 3.1** Consider the two events (1) selling of a stock $X$ by customer $Y$ at price $Z$, represented by $s(X, Y, Z)$, and (2) buying of a stock $X$ by customer $Y$ at price $W$, represented by $b(X, Y, W)$. A `buy` issued by a customer for a stock previously sold may indicate returning confidence and is an event of interest. Possible returning confidence in stock $X$ can be represented as:

$$\texttt{c}(X) \ = (\exists Y)(\texttt{relative}((\exists Z)(s(X, Y, Z)), (\exists W)(b(X, Y, W)))) \ .$$

□

**Linear Recursion** The general linear recursive expression can be expressed as follows, with one of the two recursive disjuncts being optional:

$$E(\bar{X}, \bar{Y}) = E_0(\bar{X}, \bar{Y}) \ \texttt{||} \ (\exists \bar{Z}) \ \texttt{relative(}E(\bar{X}, \bar{Z})\texttt{,}L(\bar{Z}, Y)\texttt{)} \ \texttt{||} \ (\exists \bar{Z}) \ \texttt{relative(}R(\bar{X}, Z)\texttt{,}E(\bar{Z}, \bar{Y})\texttt{)}.$$

The expression $E(\bar{X}, \bar{Y})$ is satisfied by history $h$ for a tuple $(\bar{x}, \bar{y})$ iff either expression $E_0(\bar{X}, \bar{Y})$ is satisfied by sequence $h$ for tuple $(\bar{x}, \bar{y})$, or if one of the other two disjuncts is satisfied. To satisfy, say, the first disjunct one needs to find a value $\bar{z}$ for $\bar{Z}$ such that $relative(E(\bar{X}, \bar{z})$,$L(\bar{z}, Y))$ is satisfied. As this disjunct refers to $E$, this is a "recursive" definition.

**EXAMPLE 3.2** Consider the expression:

$$F(X) = a(X) \ \texttt{||} \ (\exists Z) \ \texttt{relative(}F(Z)\texttt{,}b(Z, X)\texttt{)}.$$

Substituting basic terms in the recursive reference to $F(X)$, we can create expansions for $F(X)$, first three of which are shown:

$$\begin{aligned} F(X) \;=\; & a(X) \\ & \;||\; (\exists Z)\ \texttt{relative(}a(Z)\texttt{,}b(Z,X)\texttt{)} \\ & \;||\; (\exists Z)\ \texttt{relative(((}\exists W)\ \texttt{relative(}a(W)\texttt{,}b(W,Z)\texttt{)))}\texttt{,}b(Z,X)\texttt{)} \\ & \;\vdots \end{aligned}$$

When applied to history $h = \{a(2), b(3,5), b(2,4), a(5), b(4,6)\}$. the expression $F(X)$ is satisfied at four points: (i) $a(2)$ by virtue of the first expansion, (ii) $b(2,4)$ by virtue of the second expansion, with $Z$ bound to 2, and (iii) $a(5)$ by virtue of the first expansion, and (iv) $b(4,6)$ by virtue of the third expansion, with $Z$ bound to 4 and $W$ bound to 2. □

**Masking** A boolean function $p(\bar{X})$ is called a *mask*, since it can be used to reject (mask out) a tuple $\bar{X}$ that does not satisfy the predicate $p$. The event expression $E(\bar{X}) \equiv F(\bar{X})$ `&` $p(\bar{X})$, is satisfied for an $\bar{x}$ value if $F(\bar{X})$ is satisfied for the $\bar{x}$ value, and the mask predicate $p(\bar{x})$ evaluates to true.

# 4 More Operators For Specifying Events

We present some additional operators (connectives) that make composite events easier to specify. These operators do not add to the expressive power provided by the operators introduced in the previous section.

Let $h$ denote a non-null history, and $E$, $F$, and $E_i$ denote event expressions. The new operators are

1. *any* denotes the disjunction of all the basic events, and can have attributes that are common to all basic events in the history.

2. $prior(E,\ F)$ specifies that an event $F$ takes place after an event $E$ has taken place. $E$ and $F$ may overlap.
   Formally, $prior(E(\bar{X}),\ F(\bar{Y})) = relative(E(\bar{X}),\ any)\ \wedge\ F(\bar{Y})$.

3. $prior(E_1, ...,\ E_m)$ specifies occurrences, in order, of the events $E_1$, $E_2$ ,..., $E_m$.
   $prior(E_1(\bar{X_1}), ...,\ E_m(\bar{X_m})) = prior((prior(E_1(\bar{X_1}), ...,\ E_{m-1}(\bar{X_{m-1}})),\ E_m(\bar{X_m}))$.

4. $sequence(E_1, ...,\ E_m)$ specifies immediately successive occurrences of the events $E_1$, $E_2$ ,..., $E_m$:

   (a) $sequence(E_1(\bar{X_1}), ...,\ E_m(\bar{X_m})) = sequence((sequence(E_1(\bar{X_1}), ...,\ E_{m-1}(\bar{X_{m-1}})),\ E_m(\bar{X_m}))$.

   (b) $sequence(E_1(\bar{X_1}),\ E_2(\bar{X_2})) = relative(E_1(\bar{X_1}), !(relative(any,\ any)))\ \wedge\ E_2(\bar{X_2})$.
   The first operand of the conjunction specifies the first event following event $E_1$. The second operand specifies that the event specified by the complete event expression must satisfy $E_2$.

5. *first* identifies the first event in a history, and can have attributes that are common to all basic events in the history.
   $first\ = !relative(any,\ any)$.

6. $(E(\bar{X})|F(\bar{Y}))[h]\ =\ F(\bar{Y})[E(\bar{X})[h]]$; i.e., $F$ applied to the history produced by $E$ on $h$. Operator | is called *pipe*, with obvious similarity to the UNIX pipe operator.

**EXAMPLE 4.1** Using the pipe and sequence operators, we can write the composite event for the three successive discount rate cuts (Example 2.2) simply as

```
(I || D) | sequence(D, D, D)
```

□

7. $(< n > E)$ specifies the $n^{\text{th}}$ occurrence of event $E$. Formally,

   $(< n > E(\bar{X})) \;=\; ((E(\bar{X})|seq(any_1(\bar{X}),\ ,...,\ any_n(\bar{X})))|first(\bar{X}))$, where each $any_i$ is simply $any$. Note that each event in the histories fed to the $any$ and $first$ operators has attributes $(\bar{X})$.

8. `relative+`(F) specifies that either $F$ occurs in the full history, or that $F$ occurs relative to a past occurrence of `relative+`$(F)$. Formally,

$$E(\bar{X}) \texttt{ = relative+(}F(\bar{X})\texttt{) = } F(\bar{X}) \texttt{ || relative(}E(\bar{X})\texttt{, } F(\bar{X})\texttt{).}$$

9. $(every\ \ < n > E)$ specifies the $n^{\text{th}}$, $2n^{\text{th}}$, ..., occurrences of event $E$. Formally,

   $(every\ \ < n > E(\bar{X})) = (E(\bar{X})|relative + (< n > any(\bar{X})))$.

10. $before(E(\bar{X})) \;=\; prior(E(\bar{X}),\ any)$.

11. $happened(E(\bar{X})) \;=\; E(\bar{X}) \;\vee\; prior(E(\bar{X}),\ any)$.

12. $con(Y)$`(conrel+(`$F(X)$`))`, where $con$ is an incrementally computable function, specifies an aggregation over all the $X$ values for which event $F$ occurs. Formally,

$$E(Y) = con\texttt{(Y)(conrel+(}F(X)\texttt{))}$$

   is equivalent to

$$E(Y) \;=\; F(X) \texttt{ \& } Y \leftarrow con1(X) \texttt{ || relative(}E(X)\texttt{,}F(Z)\texttt{) \& } Y \leftarrow con2(X, Z)\texttt{.}$$

   where $\leftarrow$ denotes assignment, and functions $con1$ and $con2$ are derived from the incrementally computable function $con$ such that :

   - Function $con1$ takes one tuple $\bar{t}$ as argument, and returns the tuple $con(\{\bar{t}\})$.
   - Function $con2$ takes two tuples $\bar{t_1} = con(S_1)$ and $\bar{t_2} = con(S_2)$ representing the aggregation over two sets $S_1$ and $S_2$, and returns the tuple $con2(\bar{t_1}, \bar{t_2}) = con(S_1 \cup S_2)$.

**EXAMPLE 4.2** For the $con$ function `sum`, $con1(X) = X$, and $con2(X, Y) = X + Y$. Thus,

$$E(Y) \;= sum(Y)(conrel + (F(X))).$$

is equivalent to

$$E(Y) \;=\; F(Y) \;||\; relative(E(X), F(Z)) \;\&\; Y \leftarrow X + Z.$$

Consider a history in which composite events $F(2)$ and $F(3)$ occur at some event occurrence, followed by an $F(5)$ occurrence relative to these. Finally, suppose that at the last (current) event in the history we have $F(1)$ occurs relative to the first pair of $F$ events, but not relative to $F(5)$, and $F(9)$ occurs relative to all previous $F$ occurrence and also with respect to the entire history. There are no other $F$ occurrences in the history or any sub-history thereof.

The valid ways of satisfying `sum(`$Y$`)conrel+(`$F(X)$`)` are:
$< F(2)\ F(5)\ F(9) >$ giving $E(16)$,
$< F(3)\ F(5)\ F(9) >$ giving $E(17)$,
$< F(2)\ F(1) >$ giving $E(3)$,
$< F(3)\ F(1) >$ giving $E(4)$,
$< F(2)\ F(9) >$ giving $E(11)$,
$< F(3)\ F(9) >$ giving $E(12)$, and
$< F(9) >$ giving $E(9)$.
Thus $E(X)$ is satisfied by the set of $X$ values, $\{3, 4, 9, 11, 12, 16, 17\}$. □

# 5 Event Specification Examples

We give a few examples of temporal events written as event expressions with attributes.

**EXAMPLE 5.1** Consider the basic event `deposit`$(C, D, A)$ representing a deposit of amount $D$ by customer $C$, where $A$ was the cash balance in the bank before the deposit was made. Define `withdraw`$(C, D, A)$ similarly.

1. The $5^{\text{th}}$ occurrence of `deposit` when the total balance in the bank is less than $5000:

   ```
   (<5>(deposit(A) & A < 5000))
   ```

2. `deposit` followed immediately by a `withdraw` of the deposited amount by the same customer:

   ```
   sequence(deposit(C, D, A_1), withdraw(C, D, A_2))
   ```

3. `deposit` followed eventually by `withdraw` of the deposited amount by the same customer.

   ```
   prior(deposit(C, D, A_1), withdraw(C, D, A_2))
   ```

□

**EXAMPLE 5.2** Consider the event: selling of a stock $X$ by customer $Y$ at price $Z$, represented by $s(X, Y, Z)$. Suppose we are interested in three consecutive sales of the same stock at the same prices, with no intervening sales:

$$\exists(A, B, C)(sequence(s(X, A, P), s(X, B, P), s(X, C, P)))$$

If we are interested in three consecutive sales of the same stock with increasing prices:

$$\exists(A, B, C)(sequence(s(X, A, P1), s(X, B, P2), s(X, C, P3))\ \&\ P1 \leq P2\ \&\ P2 \leq P3)$$

□

**EXAMPLE 5.3** Passage of time. Clock tick events can be used to mark time. For instance, a tick could occur every second. But then, a temporal event expression that says within 1 hour, would have to count 3600 tick events. If we make the time (as read at the same one second granularity, from a system clock) an additional attribute of the sell event, then we can simply take the difference of times. Thus the number of events that need to be tracked by the system is reduced, increasing system efficiency.

Let the following event be of interest: two IBM stock sales by the same customer within one hour of each other. By making the time an attribute, we only track IBM sale events. We don't need to track 3600 time (second elapsed) events. Thus, we write the expression:

$$relative(s(\text{“}IBM'', Y, A_1, T_1), s(\text{“}IBM'', Y, A_2, T_2))\&(T_2 - T_1 < 3600).$$

□

# 6 Conclusions

We argue for the importance of defining a temporal event/query language that can be efficiently implemented without storing the entire history of the database states. We propose a declarative temporal language that we believe has this property. A subset of this language has been implemented at AT&T Bell Laboratories. Algorithms for implementation a large subclass of the language (except for negation) are discussed in [JMS92, JMS93]. The implementation uses extended automata with a relational store on each state.

*A*lert [SPAM91], Snoop [CM91], [Ch92], [GM91], and [Ric92] provide some form of temporal querying. In *A*lert , there is no representation for sequencing of events, except what may be visible as the net effect to the final state of the database. [Ch92] extends propositional past temporal logic, [GM91] extends first order logic with operators *until* and *since*, and develop a temporal version of SQL. [Ric92] uses past and future temporal operators for queries on data organized as finite lists of items, presumably in a temporal order. Clearly, all such facilities express some form of temporal queries [MS91].

# References


[BM91] Catriel Beeri and Tova Milo. A model for active object oriented database. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 337–349, Barcelona, Spain, September 3-6 1991.

[Ch92] Jan Chomicki. Real-Time Integrity Constraints In *Proceedings of the Eleventh Symposium on Principles of Database Systems (PODS)*, pages 274–281, San Diego, CA, June 2-4 1992. ACM SIGACT-SIGMOD-SIGART.

[CM91] S. Chakravarthy and D. Mishra. An event specification language (snoop) for active databases and its detection. Technical Report CIS TR-91-23, University of Florida, 1991.

[DBB+88] Umeshwar Dayal, Barbara Blaustein, Alex Buchmann, U. Chakravarthy, Meichun Hsu, Rivka Ladin, Dennis R. McCarthy, Arnon Rosenthal, S. Sarin, Michael J. Carey, Miron Livny, and R. Jauhari. The hipac project: Combining active databases and timing constraints. *ACM-SIGMOD Record*, 17(1):51–70, March 1988.

[DHL91] Umeshwar Dayal, Meichun Hsu, and Rivka Ladin. A transaction model for long-running activities. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 113–122, Barcelona, Spain, September 3-6 1991.

[GJ91] Narain Gehani and H. V. Jagadish. Ode as an active database: Constraints and triggers. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 327–336, Barcelona, Spain, September 3-6 1991.

[GJS92a] Narain Gehani, H. V. Jagadish, and Oded Shmueli. Composite event specification in active databases: Model and implementation. In *Proceedings of the Eighteenth International Conference on Very Large Databases (VLDB)*, pages 327–338, Vancouver, Canada, August 23-27 1992.

[GJS92b] Narain Gehani, H. V. Jagadish, and Oded Shmueli. Event specification in an active object-oriented database. In *Proceedings of ACM SIGMOD 1992 International Conference on Management of Data*, pages 81–90, San Diego, CA, June 2-5 1992.

[GM91] D. Gabbay and P. McBrien. Temporal logic and historical databases. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 423–430, Barcelona, Spain, September 3-6 1991.

[JMS92] H. V. Jagadish, Inderpal Singh Mumick, and Oded Shmueli. Events with attributes in an active database. Technical Report 921214-18-TM, AT&T Bell Laboratories, December 1992.

[JMS93] H. V. Jagadish, Inderpal Singh Mumick, and Oded Shmueli. Sequences with attributes. In *Submitted for Publication*, 1993.

[LLPS91] Guy M. Lohman, Bruce Lindsay, Hamid Pirahesh, and K. Bernhard Schiefer. Extensions to starburst: Objects, types, functions, and rules. *Communications of the ACM*, 34(10):94–109, October 1991.

[MD89] Dennis R. McCarthy and Umeshwar Dayal. The architecture of an active database management system. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data*, pages 215–224, Portland, OR, May 1989.

[MS91] L. E. McKenzie, Jr. and R. T. Snodgrass. Evaluation of Relational Algebras Incorporating the Time Dimension. *ACM Computing Surveys*, 23(4):421–500, December 1991.

[Ric92] Joel Richardson. Supporting lists in a data model (a timely approach). In *Proceedings of the Eighteenth International Conference on Very Large Databases (VLDB)*, pages 127–138, Vancouver, Canada, August 23-27 1992.

[SK91] Michael Stonebraker and Greg Kemnitz. The postgres next-generation database management system. *Communications of the ACM*, 34(10):78–93, October 1991.

[SPAM91] Ulf Schreier, Hamid Pirahesh, Rakesh Agrawal, and C. Mohan. Alert: An architecture for transforming a passive dbms into an active dbms. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 469–478, Barcelona, Spain, September 3-6 1991.

[SSU91] Abraham Silberschatz, Michael Stonebraker, and Jeffrey D. Ullman. Database systems: Achievements and opportunities. *Communications of the ACM*, 34(10):110–120, October 1991.

[WF90] Jennifer Widom and Sheldon J. Finkelstein. Set-oriented production rules in a relational database system. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 259–270, Atlantic City, NJ, May 23-25 1990.