A logical framework for temporal deductive databases

S. M. Sripada

Department of Computing
Imperial College of Science & Technology
180 Queen's Gate, London SW7 2BZ

Abstract

Temporal deductive databases are deductive databases with an ability to represent both valid time and transaction time. The work is based on the Event Calculus of Kowalski & Sergot. Event Calculus is a treatment of time, based on the notion of events, in first-order classical logic augmented with negation as failure. It formalizes the semantics of valid time in deductive databases and offers capability for the semantic validation of updates and default reasoning. In this paper, the Event Calculus is extended to include the concept of transaction time. The resulting framework is capable of handling both proactive and retroactive updates symmetrically. Error correction is achieved without deletions by means of negation as failure. The semantics of transaction time is formalised and the axioms of the Event Calculus are modified to cater for temporal databases. Given a description of events, axioms are presented for deducing relationships and the time periods for which they hold with respect to any past/present state of the database .

(1) Introduction

Interest in research concerning the study of time in databases has been growing steadily over the past few years. The study of time has attracted researchers from various fields such as artificial intelligence, databases, natural language processing, logic and concurrent systems. Many researchers have incorporated time into conventional databases using various schemes providing different capabilities for handling temporal information. In particular, work has been done on adding temporal information to conventional databases to turn them into temporal databases. However, no work is done on handling both transaction time and valid time in deductive databases. In this paper we propose a framework for dealing with time in temporal deductive databases.

Snodgrass & Ahn [16,17] describe a classification of databases depending on their ability to represent temporal information. They identify three different concepts of time - valid time, transaction time and user-defined time. Of these, **user-defined time** is temporal information of some kind that is of interest to the user but does not concern a DBMS. It is just treated as any other non-temporal attribute. **Valid time** is the time for which information models reality and corresponds to the actual time for which a relationship holds in the real world. **Transaction time**, on the other hand, is the time at which information is stored in the database. Databases which represent only the latest snapshot of the world being modelled are called **snapshot** databases. All conventional databases come under this category, for example, INGRES[20]. Databases which represent transaction time alone and therefore treat valid time and transaction time as identical are called **rollback** databases since it is possible to rollback to a past state of the database and pose a query with respect to that state. The POSTGRES[21] model is a database in this category. The problem with representing transaction time alone is that a history of the database activities is recorded (since every database state is stored, in effect), rather than the history of the world being modelled. Thus it is not possible to make proactive/retroactive updates and errors in a past state cannot be corrected.

Databases which store the history of the real world as is best known are called **historical** databases[2,4,5,9,10,12]. These databases have the concept of valid time alone but it is possible to make changes to the history of each tuple. Thus changes could be made to reflect the past as is best known. Finally, databases which store all the past history as is best known at every state of the database are called **temporal** databases[19]. These databases involve a representation of both valid time and transaction time for each tuple. An extensive bibliography of research concerning time in databases is given in [13,18].

Temporal databases are especially useful when updates have retrospective effect. For example, in [15], Sergot et al. discuss the representation of the British Nationality Act as a logic program. A database might be used to record the details of a person. The logic program can then be used to deduce whether or not that person is, according to the act, a British citizen. As the person's details change, their status can change accordingly. It is possible for a person to become a British citizen and for this status to have effect from the date of their birth. A temporal database preserves the distinction between what was **previously considered** to be their status at the time of their birth and what is **now considered** to be their status at that time. This capability is achieved by an explicit storage of the time at which information becomes available. In the case of databases, this time is referred to as the transaction time.

In this paper, we extend the Event Calculus of Kowalski & Sergot[8] to formalize the semantics of time for temporal deductive databases. In the following section, we outline the treatment of time in the Event Calculus. In section 3, we outline the framework, in section 4 we formalize the semantics of transaction time in temporal databases and, in section 5, we modify the axioms of the Event Calculus to accommodate the concept of transaction time thereby forming a theory of time for temporal deductive databases.

(2) Event Calculus

The Event Calculus of Kowalski & Sergot[8] is a treatment of time formalized in first-order classical logic augmented with negation as failure[3]. It is based on the notion of an event as primitive. Various relationships and the time periods for which they hold are derived from a *description of events* that occur in the real world. The approach is closely related to Lee et al.'s[10,14] treatment of time in deductive databases and Allen's interval temporal logic[1]. In his paper, Kowalski[9] deals with database updates using the Event Calculus and compares this approach with updates in a conventional relational database and the use of modal temporal logic, situation calculus and case semantics for storing and updating information in a database. It is argued that the Event Calculus combines the expressive power of both case semantics and situation calculus, derives the computational power of logic programming and overcomes the frame problem of the situation calculus. Schemes for specializing the Event Calculus so as to make it comparable in efficiency to that of relational databases with destructive assignment are also presented in Kowalski[9]. In the following, we give an outline of the treatment of time in the Event Calculus.

In the Event Calculus, event descriptions are used to deduce the existence as well as the initiation and termination of time periods. Each time period is uniquely identified by a combination of the relationship that holds during the time period and the event which initiates or terminates the period. This is so, because, each event may initiate or terminate more than one time period and there may be several different time periods in which the same relationship holds. The term[7,11] after(e r) is used to denote a time period started by an event e in which the relationship r holds. Similarly, the term before(e r) is used to denote a time period terminated by the event e in which the relationship r holds. The atom[7,11] Holds(after(e r)) is a shorthand version for the atom Holds(r after(e r)) and means that the relationship r holds in the time period after(e r). Similarly, the atom Holds(before(e r)) asserts the fact that the relationship r holds in the period before(e r).

For example, let E1 be an event in which *John gave a book to Mary* and E2 be an event in which *Mary gave the book to Bob*. Assume that E2 occurred after E1. Then, given these event descriptions, we could deduce that there is a period started by the event E1 in which Mary possesses the book and that there is a period terminated by E1 in which John possesses the book. This is represented pictorially in Fig 1.

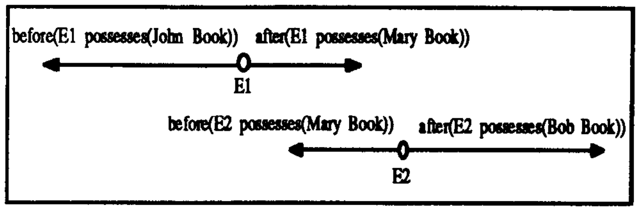
before(E1 possesses(John Book))
after(E1 possesses(Mary Book))
E1
before(E2 possesses(Mary Book))
after(E2 possesses(Bob Book))
E2

Fig 1

The axioms of the Event Calculus for deducing such time periods are as follows:

Holds(before(e r)) if Terminates(e r) EC1

Holds(after(e r)) if Initiates(e r) EC2

These axioms are used along with application specific rules such as

```
Initiates(e possesses(x y)) if
                    Act(e give)  and
                    Recipient(e x)  and
                    Object(e y)
and

Terminates(e possesses(x y)) if
                    Act(e give)   and
                    Donor(e x)    and
                    Object(e y)
```

These rules state the relationships that each kind of event initiates and terminates.The types of events for which such rules are given depends upon the domain of application. It is then possible to deduce that the following are true:

Holds(after(E1 possesses(Mary Book)))
Holds(before(E1 possesses(John Book)))
Holds(after(E2 possesses(Bob Book)))
Holds(before(E2 possesses(Mary Book))).

Having deduced the existence of time periods, there is a need to deduce the Start and End of each time period. The axioms for this are the following:
(The atom Start(x y) denotes that the start of the time period x is the event y.)

```
Start(after(e r) e)                                 EC3

End(before(e r) e)                                  EC4

Start(before(e' r) e) if
                after(e r) = before(e' r)           EC5

End(after(e r) e') if
                after(e r) = before(e' r)           EC6
```

These axioms allow us to deduce that
(i)the period *after(E1 possesses(Mary Book))* is started by the event E1 {using axiom EC3}
(ii)the period *before(E1 possesses(John Book))* is terminated by the event E1 {axiom EC4}
(iii)the period *before(E2 possesses(Mary Book))* is started by the event E1 if the periods *after(E1 possesses(Mary Book))* and *before(E2 possesses(Mary Book))* are identical {axiom EC5} and
(iv)the period *after(E1 possesses(Mary Book))* is terminated by the event E2 if the periods *after(E1 possesses(Mary Book))* and *before(E2 possesses(Mary Book))* are identical {axiom EC6}.

However, deducing that two time periods are identical involves some default reasoning since it depends on the assumption that there is no event in between which initiates or terminates a conflicting time period. This is captured in the following axioms(readers are referred to [8] for detailed explanation and examples):

```
after(e r) = before(e' r)  if
                  Holds(after(e r))    and
                  Holds(before(e' r))    and
                  e < e'   and
                  NOT Broken(e r e')      EC7
```

Here, the relation < is a chronological ordering on events. $e < e'$ means that the event e occurs chronologically earlier than the event e'. The relation $\leq$ has a similar meaning. The NOT is interpreted as negation as failure.

```
Broken(e r e')  if  Holds(after(e* r*))  and
                    Exclusive(r r*)  and
                    e < e*    and
                    e* < e'                EC8
```

Broken(e r e') if Holds(before(e* r*)) and
Exclusive(r r*) and
e < e* and
e* < e' EC9

The atom Exclusive(r r*) is true when the relationships r and r* are identical *or* r and r* are incompatible(cannot be true simultaneously).

Exclusive(r r*) if r = r*
Exclusive(r r*) if Incompatible(r r*)
and, to reason with equality,
r = r

The incompatibility of relationships are defined by application specific rules such as

Incompatible(owns(Mary Book1)
owns(Bob Book1))

which states that different persons cannot own the same book at the same time.

There are other cases whereby time periods interact to imply the existence of Starts and Ends. These are described below by means of examples (in figures) along with the corresponding axioms.

The case where there are two events, one initiating a relationship and the other terminating the relationship, with no event that affects the relationship occurring between these two events is shown in Fig 2. Axioms EC5, EC6 and EC7 deal with this case.

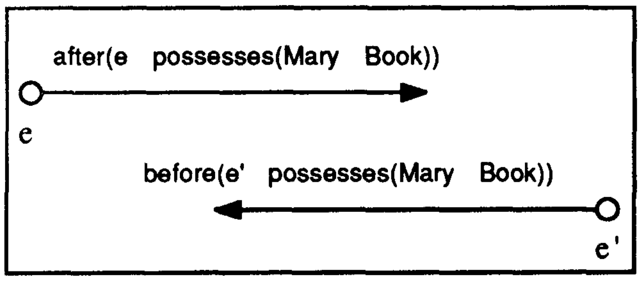
after(e possesses(Mary Book))
e
before(e' possesses(Mary Book))
e'

Fig 2

When we know of two events which terminate relationships that are mutually exclusive but with no known event initiating the later of these relationships, it is possible to deduce, by default, the existence of an event initiating that relationship. This case is shown in Fig 3. The relationship r' in Fig 3 can be either possesses(Mary Book) or some other relationship which is incompatible with it, such as, possesses(Bob Book).

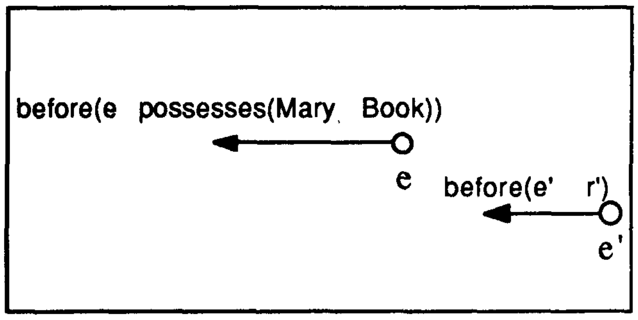

Fig 3

For the case shown in Fig 3, the following axioms are needed:
(The functions *init* and *fin* give the start and end points of a time period respectively.)

[Start(before(e' r') init(before(e' r')))
and
e $\leq$ init(before(e' r'))] if
Holds(before(e r)) and
Holds(before(e' r')) and
Exclusive(r r') and
e < e' and
NOT Broken(e r' e') EC10

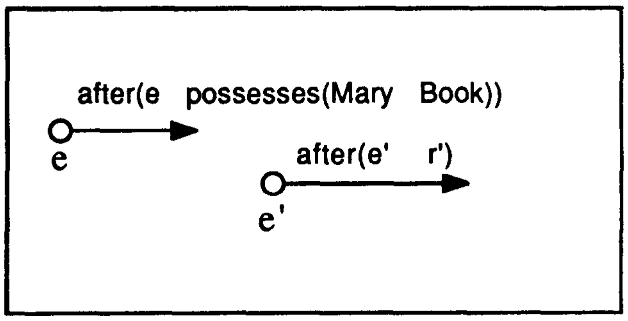

Fig 4

The case shown in Fig 4 is symmetric to that of Fig 3 but with the terminating event unknown. Again, the relationship r' is mutually exclusive to the relationship possesses(Mary Book). This is formalized by the following axiom:

```
[ End(after(e r) fin(after(e r)))
          and
  fin(after(e r)) ≤ e' ]   if
            Holds(after(e r))  and
            Holds(after(e' r'))  and
            Exclusive(r r')  and
            e < e'  and
            NOT Broken(e r e')        EC11
```

For the case shown in Fig 5 the following axiom is needed:

```
[ fin(after(e r)) ≤ init(before(e' r'))
          and
  Start(before(e' r') init(before(e' r')))
          and
  End(after(e r) fin(after(e r))) ]   if
            Holds(after(e r))    and
            Holds(before(e' r'))  and
            Exclusive(r r')    and
            NOT  r = r'  and
            e < e'  and
            NOT  Broken(e r e')      EC12
```

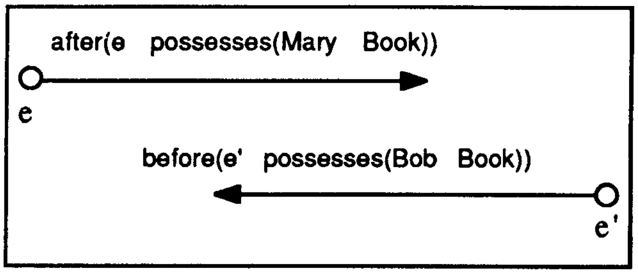

Fig 5

Finally, knowing that a particular relationship holds for a certain time period, the axioms for deducing that the relationship holds at a given point in time are as follows:

```
HoldsAt(r t)  if  Holds(after(e r))  and
                  t in after(e r)        EC13

HoldsAt(r t)  if  Holds(before(e r))  and
                  t in before(e r)       EC14

t in x     if     Start(x e')  and
                  End(x e")  and
                  Time(e' t')  and
                  Time(e" t")  and
                  t' < t  and
                  t < t"                 EC15

t in x     if     Start(x e')  and
                  Time(e' t')  and
                  NOT End(x e")  and
                  t' < t                 EC16
```

The axiom EC16 is valid only for the special case where events are recorded in the order in which they occur and the database contains a complete record of all the relevant past events.

(3) Overview of the framework

One way to use the Event Calculus for database applications would be to record all the events of interest that have taken place in the real world in a database. The axioms of the Event Calculus can then be used to deduce relationships along with the time periods for which they hold. Thus the Event Calculus gives a treatment of valid time in databases. Event Calculus, applied to databases, formalizes a historical database[9,10,17] without a facility for correcting errors.

In addition to the above, **temporal databases** need to be able to deal with transaction time as well. Transaction time is concerned with the time at which information is stored in a database. Every event is entered into the database through a transaction. Thus a transaction is a meta-event. When information regarding an event e is entered into a database through a transaction **tr**, a belief period(also called transaction time period) is started in which the information regarding the event e is believed. The predicate Basis(tr e) is used to express the fact that the basis for believing in the event e is the transaction tr. However, such a belief period ends when another event e' is entered into the database and e' revises(corrects) the information given by the event e.

With the aid of belief periods, it is possible to find out what are all the events that are believed to be true at a given point in transaction time(alternatively, in any given database state). Once the set of such events is known, it is possible to deduce the relationships that are implied by these events using the axioms of the Event Calculus(in a modified form).

Since the history of the world as represented in the database may contain some errors, there is a need for the ability to revise some of the information. In our scheme, we achieve this by means of an additional relation Revises(e e') which states that the information given by event e overrides the information given by the event e'. Thus corrections to history are always achieved by asserting the existence of a **new event** which revises an earlier event. No event description is ever deleted.

From an application point of view, the scheme is implemented as follows:

(i) Record an event e (occurring in the real world) in the database through a transaction tr. The relations that are added to the database are
Basis(tr e) and Time(tr t). For example, if the event E10 is entered in the database through a transaction TR5, the following relations are added to the database:

Basis(TR5 E10)
Time(TR5 5January).

(ii) Enter the information regarding the event itself. For example,

Act(E10 Give)
Donor(E10 Bob)
Recipient(E10 Mary)
Object(E10 Book1)
Time(E10 2January).

(iii) If the newly entered event e is supposed to be a correction to the information given by another event e', add the relation Revises(e e') to the database. For example,

Revises(E10 E8).

Note that
(a)both past and future events can be entered into the database. It does not matter whether an event is yet to occur in the future i.e. the time of occurrence of the event is greater than *now*, the current time. This provides an ability to handle proactive updates
(b)every event has a basis(for belief) which is the database transaction through which it was entered
(c)information regarding events may be entered into the database in any order with respect to their actual order/time of occurrence in the real world
(d)transactions, of course, can only be entered into the database in the order in which they actually take place. Obviously, the time of a transaction cannot be greater than the current time.

With the above scheme it is possible to answer temporal queries. Before that we need to formalize the concept of transaction time which is done in the following section.

(4) Transaction time

Transaction time represents the time at which information is stored in a database. The addition of the concept of transaction time to a database enables one to answer queries such as "What was Mary's rank on 5 January as per the database of 10 January?". The reasoning involved in answering this query can be split into two parts: (i) what are the events that are believed to be true as per the database of 10 January? and (ii)what are the inferences that can be drawn from this subset of events regarding the rank of Mary on 5 January? The second of these is handled to a large extent by the treatment of time in the Event Calculus. The reasoning involved in solving the first part needs to be formalized. In this section we formalize the deduction of belief periods so that these rules could be used in conjunction with the axioms of the Event Calculus, modified suitably, to form a framework for the treatment of time in temporal databases. Consider the following example which involves a **correction** as well as belief periods:

<u>Example 1</u>

E3 is an event in which *John was hired as a lecturer on 1 January*. The information regarding E3 was entered into the database on 3 January, through a transaction TR1. It was discovered subsequently that *John was hired as a lecturer on 5 January* and this information was entered into the database by another transaction TR2 on 8 January.

Transaction TR1 adds the following information to the database :

Basis(TR1 E3)
Time(TR1 3January)
Act(E3 hire)
Object(E3 John)
NewRank(E3 lecturer)
Time(E3 1January)

The predicate Basis(tr e) indicates that tr is the transaction which is the basis for believing the information regarding event e. The transaction TR1 initiates a belief period in which it is believed that John was hired as a lecturer on 1 January. Transaction TR2 introduces a correction (a revision) to the data already existing in the database. Since old data is not deleted, the correct information is entered as a **new** event, say E4, along with the relation Revises(E4 E3). The database contains the following additional information after transaction TR2:

Basis(TR2 E4)
Time(TR2 8January)
Act(E4 hire)
Object(E4 John)
NewRank(E4 lecturer)
Time(E4 5January)
Revises(E4 E3)

Since all the information, old and new, is now present in the database, it should be possible to reason that after 3 January and before 8 January it was believed that John was hired as lecturer on 1 January. After 8 January, however, it is believed that John was hired as a lecturer on 5 January. If the information regarding the event E4 is not revised by any subsequent transactions, we continue to believe that John was hired as a lecturer on 5 January. Note that this belief will not be affected by normal updates, such as, John getting promoted to the rank of professor at a later date.

Thus, as a result of transaction TR2, the belief period in which it is believed that John was hired as a lecturer on 1 January was terminated. TR2 involves the **revision** of information given by another event. We now proceed to formalize the reasoning involved in deducing belief periods.

Since each transaction may start and/or terminate more than one belief period, we use a notation similar to that of the Event Calculus to identify belief periods. The term after(tr e) denotes a time period started by the transaction tr in which the information regarding the event e is believed to be true. The term before(tr e) has a similar meaning. The predicate BHolds(e p) expresses the relationship that event e is believed to be true in the period p. The above reasoning can be expressed by the following rules:

BHolds(e after(tr e)) if Basis(tr e) A1

BHolds(e before(tr e)) if Basis(tr e') and Revises(e' e) A2

There are two main differences between the concepts of transaction time and valid time. The first is that transactions are entered into databases strictly in a chronologically ascending order (transaction time moves only into the future) whereas events themselves may be entered in any order in relation to their actual time of occurrence in the real world. In other words, at any point in time, the database contains a complete record of all the transactions that have taken place, whereas the database may not contain a complete record of all the events that occurred in the world. The second difference is that for a belief period to be terminated by a transaction, it must have been started earlier by another transaction. Otherwise, it means that there is an event in the database which is believed to be true but there is no transaction which is responsible for entering the event in the database - a contradiction. However, the database might be informed of an event signalling the end of a valid time period without any prior information regarding its beginning as emphasized in the Event Calculus. For these reasons, it is sufficient to consider belief periods in the forward direction alone (i.e. into the future). Therefore rule A2 is redundant.

The rule A1 is not valid for all cases as is illustrated by the following example:

Example 2

Same as Example 1 except for the transaction times which are as follows:

Time(TR1 10January)
Time(TR2 8January).

In other words, information regarding event E3 is entered into the database on 10 January even though it is known that event E4 revises event E3, and E4 is already in the database. Since it is known on 10 January that E4 revises E3, the information regarding E3 is not believed in at any time for deducing relationships.
However, rule A1 allows one to deduce that BHolds(E3 after(TR2 E3)) is true. The rule A1 therefore needs to be modified as follows:

```
BHolds(e after(tr e))  if   Basis(tr e)  and
                 NOT [ Revises(e' e)  and
                       Basis(tr' e')  and
                       tr' ≤ tr ]              TT1
```

Again, the information regarding an event e is believed to be true at transaction time t if t is a time instant within the belief period of e. This is given by the following rules:

```
BHoldsAt(e t)  if  BHolds(e after(tr e))  and
                   t within after(tr e)      TT2

t within p   if   BStart(p tr')  and
                  BEnd(p tr")  and
                  Time(tr' t')  and
                  Time(tr" t")  and
                  t' < t    and
                  t < t"                      TT3
```

The predicates BStart and BEnd denote the start and end of a belief period respectively and are analogous to the Start and End predicates of section 2.In the case where a belief period is not terminated by any transaction, it extends into eternity and requires the following axiom:

```
t within p   if   BStart(p tr')   and
                  Time(tr' t')    and
                  t' < t    and
                  NOT BEnd(p tr")             TT4
```

The starts and ends of belief periods are given by the axioms TT5 and TT6. It can be argued that these definitions of BStart and BEnd are complete.

```
BStart(after(tr e) tr)                        TT5

BEnd( after(tr e) tr')  if   Basis(tr' e') and
                             Revises(e' e)    TT6
```

Handling proactive and retroactive updates:

Both proactive and retroactive updates are handled symmetrically. All that needs to be done to record such an update is to enter information in the form of new events in the usual way and assert relationships regarding the revision of events previously entered, if any. The following example makes the procedure clear:

Example 3

Mary was hired as a lecturer **on** 5 April, 1985(event E1). This data was entered into a database **on** 7 April, 1985(transaction TR1). Mary was then promoted to the rank of a professor **on** 12 November, 1987 **with effect from** 1 October, 1987(event E2). This information was entered into the database **on** 15 November, 1987(transaction TR2).

The relations of interest that are added to the database as a result of these two transactions are

```
Basis(TR1 E1)
Time(TR1 7April1985)
Time(E1 5April1985)
Act(E1 hire)
Basis(TR2 E2)
Time(TR2 15November1987)
Act(E2 promote)
Time(E2 1October1987)
```

Notice that there is no need for the revision of any past event. The time 12November1987 comes under the category of user-defined time which does not contribute to the reasoning involved. It could just be asserted like any other non-temporal information.

Axioms TT1-TT6 then imply that the following are true:

BHolds(E1 after(TR1 E1))
BHolds(E2 after(TR2 E2))
BStart(after(TR1 E1) TR1) {Therefore we believe in event E1 from 7 April onwards}
BStart(after(TR2 E2) TR2) {Therefore we believe in event E2 from 15 November onwards}

Hence, after 15 November we believe in the occurrence of both the events E1 and E2. This situation is analogous to the one shown in Fig 4. The use of Event Calculus axioms on the event descriptions E1 and E2 then allow us to conclude that:
Mary was hired as a lecturer on 5 April 1985,
Mary was promoted to the rank of professor on 1 October 1987, and
Mary was a professor on 2 October 1987.
However, on 14 November 1987, the belief period after(TR2 E2) has not started yet.

Therefore, there is only one event E1 in which the database believes. The axioms of the Event Calculus applied to the event E1 alone would allow us to conclude that:
Mary was hired as a lecturer on 5 April 1985 and Mary was a lecturer on 2 October 1987 {axiom EC13}.
This is how the inferences that follow from any state of the database are deduced.

We now give another example wherein the need for a revision of some of the events arises in a proactive/retroactive update. Explanatory comments are enclosed in { }.

<u>Example 4</u>

Mary was hired as a lecturer **in** April 1985(event E1,transaction TR1){simple update}. A decision was taken **in** June 1985 to promote Mary to the rank of assistant professor for a period of 2 years **starting from** August 1985 (event E2, transaction TR2) and then to promote her to the rank of professor at the end of that period(event E3, transaction TR2){proactive updates}. Mary turned out to be extremely brilliant and was therefore promoted to the rank of professor **in** October 1985 **with effect from** May 1985(event E4, transaction TR3){a retroactive update affecting earlier proactive updates, some of which have already taken effect}.

The relations of interest that are added to the database as a result of these three transactions are

Basis(TR1 E1)
Time(TR1 April1985)
Act(E1 hire)
NewRank(E1 lecturer)
Time(E1 April1985)
Basis(TR2 E2)
Basis(TR2 E3)
Time(TR2 June1985)
Act(E2 promote)
OldRank(E2 lecturer)
NewRank(E2 assistantProfessor)
Time(E2 August1985)
Act(E3 promote)
OldRank(E3 assistantProfessor)
NewRank(E3 professor)
Time(E3 August1987)
Basis(TR3 E4)
Time(TR3 October1985)
Act(E4 promote)
OldRank(E4 lecturer)
NewRank(E4 professor)
Time(E4 May1985)
Revises(E4 E2)
Revises(E4 E3)

Belief periods and the relationships that hold are then deduced using the axioms TT1-TT6 and EC1-EC16 as explained in Example 3.

(5) Modification of the Event Calculus

Since the belief in an event and the corresponding inferences may change with time, the belief that a relationship r holds in the period after(e r) is valid only when the time at which that belief is held is specified. In the case of databases, this time will be along the transaction time axis. Therefore, it is necessary to augment the Holds predicate with an additional parameter representing the transaction time. This is also true of all the other predicates of the Event Calculus. The set of augmented predicates and their definitions are as follows :

The axiom EC1 yields two new rules in the context of a temporal database. One rule is required for inferring that a valid time period before(e r) was believed to exist for a transaction time period p (rule TD1). A second rule is required for inferring that a valid time period before(e r) was believed to exist at a transaction time τ (rule TD2).

```
PHolds(before(e r) p) if
        Terminates(e r) and
        BHolds(e p)                    TD1
```

The predicate PHolds(before(e r) p) means that there is a belief period(transaction time period) p in which it is believed that the relationship r holds in the valid time period before(e r).

```
PHoldsAt(before(e r) τ) if
        PHolds(before(e r) p) and
        τ within p                     TD2
```

Alternatively, this could be reformulated as

```
PHoldsAt(before(e r) τ)   if
        Terminates(e r)   and
        BHoldsAt(e τ)
```

The predicate PHoldsAt(before(e r) τ) means that the relationship r holds for the time period before(e r) according to the state of the database at time τ.

Similarly, the axiom EC2 of the Event Calculus yields

```
PHolds(after(e r) p)   if
        Initiates(e r)   and
        BHolds(e p)                    TD3

PHoldsAt(after(e r) τ)   if
        PHolds(after(e r) p)   and
        τ within p                     TD4
```

The axioms for Start and End also need to be modified. In the general case, inferring the Start and End of a time period involves default reasoning and therefore depends upon the amount of information that is present in the database at **that** time τ. (The case where the Start and End of a valid time period are inferred to hold over a transaction time period is more involved and is not considered in this paper because of limitations of space.)

Axioms EC3 to EC6 are modified to TD5 - TD8

```
Start(after(e r)  e  τ)                TD5
End(before(e r)  e  τ)                 TD6

Start(before(e' r)  e  τ)   if
        after(e r) =_τ before(e' r)    TD7
End(after(e r)  e'  τ)   if
        after(e r) =_τ before(e' r)    TD8
```

In the example of Fig 1, the conclusion that the valid time periods
after(E1 possesses(Mary Book)) and
before(E2 possesses(Mary Book)) are identical is correct only if the database at time τ does not contain any information to the contrary (this is taken care of by the use of negation as failure in TD9). Note that an event E5 may be entered at a later time making the above conclusion incorrect in the new state of the database. Thus the rule EC7 is modified as

```
after(e r)  =_τ  before(e' r)   if
        PHoldsAt(after(e r) τ)   and
        PHoldsAt(before(e' r) τ)   and
        e < e'   and
        NOT Broken(e r e' τ)           TD9
```

The predicate Broken takes in an additional term to allow for the effect of events such as E5 mentioned above. Note that the ordering relation on events, <, is not subject to change with time since changes are affected only by asserting a new event along with an appropriate Revises relation. The exclusivity relation is also invariant in time.

```
Broken(e r e' τ)   if
        PHoldsAt(after(e* r*) τ)   and
        Exclusive(r r*)   and
        e < e*   and
        e* < e'                        TD10

Broken(e r e' τ)   if
        PHoldsAt(before(e* r*) τ)   and
        Exclusive(r r*)   and
        e < e*   and
        e* < e'                        TD11
```

The rest of the rules for deducing the Start and End are obtained by modifying EC10-EC12 as shown below:

```
[ Start(before(e' r') init(before(e' r')) τ)
        and
    e ≤ init(before(e' r'))]   if
        PHoldsAt(before(e r) τ)   and
        PHoldsAt(before(e' r') τ)   and
        Exclusive(r r')   and
        e < e'   and
        NOT Broken(e r' e' τ)          TD12
```

```
[ End(after(e r) fin(after(e r)) τ )
            and
  fin(after(e r)) ≤ e' ]       if
            PHoldsAt(after(e r)  τ )  and
            PHoldsAt(after(e' r')  τ )  and
            Exclusive(r r')  and
            e < e'  and
            NOT Broken(e r e'  τ )    TD13
```

```
[ Start(before(e' r') init(before(e' r'))  τ )
            and
  Fin(after(e r)) ≤ init(before(e' r'))
            and
  End(after(e r) fin(after(e r))  τ ) ]  if
        PHoldsAt(after(e r)   τ )  and
        PHoldsAt(before(e' r')    τ )  and
        Exclusive(r r')  and
        NOT  r = r'  and
        e < e'  and
        NOT  Broken(e r  e'   τ )      TD14
```

Again, the truth of the statement that a relationship holds at point in time(valid time) is subject to the state of a database. In Example 1, John was a lecturer on 2 January as per the database of 6 January but not as per that of 9 January. Therefore EC13-EC16 are to be modified as

```
HoldsAt(r  t  τ)   if
            PHoldsAt(after(e r)  τ)  and
            t  in  after(e r)  at  τ     TD15

HoldsAt(r  t  τ)   if
            PHoldsAt(before(e r)  τ)  and
            t  in  before(e r)  at  τ   TD16
```

Since the Start and End of time periods are subject to change with transaction time, that a point in valid time is **in** the period before(e r) or after(e r) is meaningful only when the state of the database(transaction time) is specified. Thus we have

```
t  in  p  at  τ   if   Start(p  e'  τ)   and
                       End(p   e"   τ)   and
                       Time(e'   t')   and
                       Time(e"   t")   and
                       t' < t   and
                       t < t"              TD17

t  in  p  at  τ   if   Start(p  e'  τ)  and
                       Time(e'  t')  and
                       t'  <  t   and
                     NOT End(p   e"   τ)  TD18
```

TD18 is correct only for the special case in which the database contains a complete record of all relevant past events.

The axioms TD1-TD18 along with the axioms TT1-TT6 form a logical framework for treatment of time in temporal deductive databases.

(6) Conclusions

We described a framework for representing and dealing with the concepts of valid time and transaction time in the context of temporal deductive databases. The formalization is done in the Horn clause subset of first-order classical logic augmented with negation as failure and is executable as a logic program.

We have restricted our attention to ground unit clauses as the relationships that are derivable from a description of events. However, the formalization may be extended for sentences of first-order logic. When extended thus, the sytem will be capable of reasoning with rules that are effective only for certain periods of time and then generate the corresponding inferences. Domains such as legislation require such a capability.

The framework is capable of handling both proactive and retroactive updates symmetrically. Historical and rollback databases may be obtained as special cases.

Acknowledgements

I would like to thank Robert Kowalski, Marek Sergot, Fariba Sadri and Murray Shanahan for many helpful discussions. I am grateful to Fariba Sadri for many useful comments on an earler draft of this paper. Special thanks to Cheryl Anderson for her help in typesetting the paper.

The research is sponsored by the Jawaharlal Nehru Memorial Trust(UK). The work is also supported by an ORS(UK) award.

References

[1]Allen, J. F. Maintaining Knowledge about Temporal Intervals. CACM, November 1983, Vol. 26, No. 11, pp.832-843

[2]Ariav, G. Clifford, J. & Jarke, M. Panel on Time and Databases. ACM SIGMOD Record, Vol. 13, No. 4, Sanjose, 1983

[3]Clark, K.L. Negation as Failure, in Logic and Databases, Eds. Gallaire, H. Minker, J. Plenum Press, NY 1978

[4]Clifford, J. & Warren, D. S. Formal Semantics for Time in Databases. ACM Transactions on Database Systems, Vol.8, No.2, 1983

[5]Dean, T. L. & McDermott, D. V. Temporal Data Base Management. Artificial Intelligence 32, No.1 (1987), pp. 1-55

[6]Gallaire, H. Minker, J. & Nicolas J. M. Logic and Databases: A Deductive Approach. Computing Surveys, June 1984, pp.153-185

[7]Kowalski, R. A. Logic for Problem Solving. Elsevier North Holland. 1979

[8]Kowalski, R.A. & Sergot, M.J. A Logic-based Calculus of Events. New Generation Computing, 4, No. 1 (1986), pp. 67-95

[9]Kowalski, R.A. Database updates in the Event Calculus. Research Report DoC 86/12, Department of Computing, Imperial College, London

[10]Lee, R. M. Coelho, H. & Cotta, J. C. Temporal Inferencing on Administrative Databases. Information Systems, Vol. 10, No. 2, pp.197-206

[11]Lloyd, J. W. Foundations of Logic Programming. Springer Verlag. 1984

[12]Martin, N. Navathe, S. & Ahmed, R. Dealing with Temporal Schema Anomolies in History Databases in Proceedings of the thirteenth international conference on Very Large Data Bases, Brighton, 1987, pp.177-184

[13]McKenzie, E. Bibliography: Temporal Databases. ACM SIGMOD Record, Vol. 15, No. 4, December 1986

[14]Sadri, F. Three recent approaches to temporal reasoning. Research Report DoC 86/23, Revised November, 1986, Department of Computing, Imperial College, London

[15]Sergot, M.J. Sadri, F. Kowalski, R.A. Kriwaczek, F. Hammond, P. & Cory, H.T. The British Nationality Act as a Logic Program. CACM, Vol. 29, No. 5, May 1986, pp. 370-386

[16]Snodgrass, R. & Ahn, I. A Taxonomy of time in databases, in Proceedings of ACM SIGMOD International Conference on Management of Data, Ed. S. Navathe. Association for Computing Machinery. Austin, Texas, May 1985, pp. 236-246

[17]Snodgrass, R. & Ahn, I. Temporal Databases. IEEE Computer, 19, No.9, Sep. 1986, pp. 35-42

[18]Snodgrass, R. (Ed.) Research Concerning Time in Databases Project Summaries. SIGMOD Record, Vol. 15, No. 4, December 1986

[19]Snodgrass, R. The Temporal Query Language TQuel. ACM Transactions on Database Systems, Vol.12, No.2, June 1987, pp.247-298

[20]Stonebraker, M. Wong, E. & Kreps, P. The Design and Implementation of INGRES. ACM Transactions on Database Systems, Vol.1, No.3, September 1976, pp.189-222

[21] Stonebraker, M. The Design of the POSTGRES Storage System in Proceedings of the thirteenth international conference on Very Large Data Bases, Brighton, 1987, pp.289-300