10

A Logic-based Calculus of Events

ROBERT KOWALSKI and MAREK SERGOT

1 INTRODUCTION

Formal Logic can be used to represent knowledge of many kinds for many purposes. It can be used to formalize programs, program specifications, databases, legislation, and natural language in general. For many such applications of logic a representation of time is necessary.

Although there have been several attempts to formalize the notion of time in classical first-order logic, it is still widely believed that classical logic is not adequate for the representation of time and that some form of non-classical Temporal Logic is needed. In this paper, we shall outline a treatment of time, based on the notion of event, formalized in the Horn clause subset of classical logic augmented with negation as failure. The resulting formalization is executable as a logic program.

We use the term "event calculus" to relate it to the well-known "situation calculus" (McCarthy and Hayes 1969). The main difference between the two is conceptual: the situation calculus deals with global states whereas the event calculus deals with local events and time periods. Like the event calculus, the situation calculus can be formalized by means of Horn clauses augmented with negation by failure (Kowalski 1979).

The main intended applications investigated in this paper are the updating of databases and narrative understanding. In order to treat both cases uniformly we have taken the view that an update consists of the addition of new knowledge to a knowledge base. The effect of explicit deletion of information in conventional databases is obtained without deletion by adding new knowledge about the end of the period of time for which the information holds.

2 A SIMPLIFIED EXAMPLE

A simple, informal example will illustrate the general idea. Consider the following narrative:

(1) Mary was hired as a lecturer on 10 May 1970.
(2) John left as lecturer on 1 June 1975.
(3) Mary left as professor on 1 October 1980.
(4) Mary was promoted from lecturer to professor on 1 June 1975.

Each sentence in the narrative can be considered as an update which adds new knowledge, starting from an initially empty knowledge base. In the spirit of many

Robert Kowalski and Marek Sergot, 'A Logic-based Calculus of Events', *New Generation Computing* 4(1986): 67–94.

natural language processing systems, the meaning of the new knowledge can be formulated in terms of event descriptions. Formulated in event description terms, the sequence of updates becomes:

(1) E1 is an event in which
Mary is hired as lecturer.
E1 has time 10 May 1970.

(2) E2 is an event in which
John leaves as lecturer.
E2 has time 1 June 1975.

(3) E3 is an event in which
Mary leaves as professor.
E3 has time 1 October 1980.

(4) E4 is an event in which
Mary is promoted from lecturer to professor.
E4 has time 1 June 1975.

A typical event causes the start of several (zero or more) periods of time and the end of several others. For example:

An event e of hiring x as y
starts a period of time
for which x has rank y.

This can be formulated as a Horn clause

x has rank y for period after(e)
if e is an event in which x is hired as y.

Here the term after(e) names the time period as a function of e. The start of after(e) can be defined by a conditionless Horn clause:

The start of after(e) is e.

The end of after(e) is undefined but might be determined by means of additional information later. Similar Horn clauses can be used to express that

an event e of x leaving as y
ends a period of time
for which x has rank y;

an event e of promoting x from y to z
ends a period of time
for which x has rank y and
starts a period of time
for which x has rank z.

By means of these rules it is possible to conclude **after update (1)** that

Mary has rank lecturer for period after(E1)
which starts 10 May 1970.

This can be represented pictorially as shown in Figure 1.

Similarly **after updates (2), (3)**, and **(4)**, it is possible to make the conclusions shown in pictorial terms in Figures 2–4 respectively.

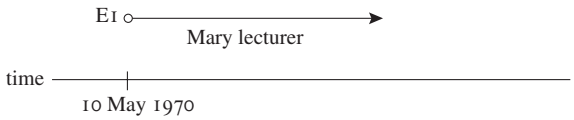

FIG. 1 After update (1).

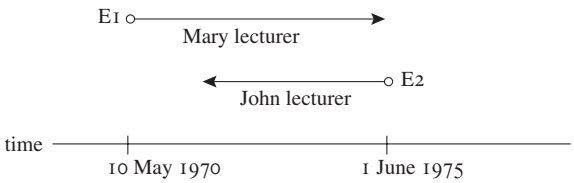

FIG. 2 After update (2).

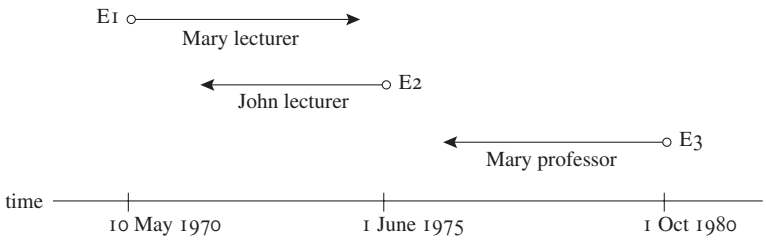

FIG. 3 After update (3).

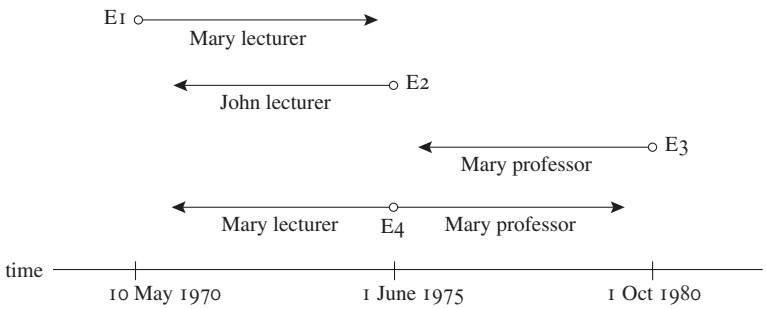

FIG. 4 After update (4).

After update (4) it would be natural to conclude that the event E4 of Mary's promotion ends her previous period after(E1) of lectureship and starts her previously identified, future period before(E3) of professorship. This can be pictured as Figure 5:

The conclusions illustrated in Figure 5 can be justified if we can prove the equalities:

after(E1) = before(E4)
after(E4) = before(E3).

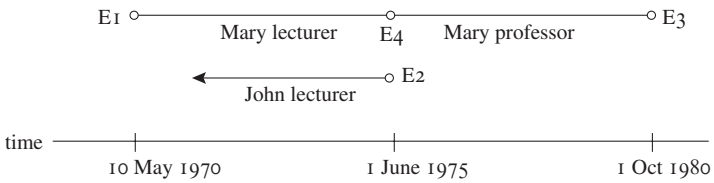

FIG. 5

Together with the rules of equality and the fact that E4 ends before(E4) and starts after(E4), these equalities imply that

E4 ends after(E1) and
E4 starts before(E3).

The two equalities can be derived by means of a rule which expresses that

two periods of time are identical
if the same individual holds the same rank for both periods,
and one period starts before the other ends,
and it cannot be shown that an event has occurred, which affects the individual's rank, after the start of the first period and before the end of the second.

This rule uses default reasoning in the expression "cannot be shown", which can be formalized by negation as failure. Such default reasoning is "non-monotonic" in the sense that conclusions derived with its assistance are automatically withdrawn if contradictory new information is later made available. This might happen in the present example if it were discovered, for instance, that Mary left temporarily in January 1978 and was rehired in October 1979.

Mary's rank at a time instant t can be determined by finding a period of time containing t and determining her rank during that period. This too can be expressed as a Horn clause

x has rank y at time t
if x has rank y for period p
and t in p.

Thus after assimilating our example narrative it should be possible to conclude that

Mary has rank lecturer on 11 May 1970 and
Mary has rank professor on 16 Feb. 1978.

Whether it should also be possible, however, to conclude that

John has rank lecturer on 30 May 1975,

for example, is more problematic. We shall deal with these and related problems later.

The simple example narrative already illustrates several general characteristics of the event calculus approach.

(1) Updates are **additive** in that they **add** but do **not delete** information about events. That a relationship no longer holds is represented by adding information which implies the end of the time period for which the relationship holds rather than by deleting the relationship. This is consistent with our use of classical logic without explicit destructive assignment.

(2) Conventional database systems, in contrast, allow arbitrary additions and deletions of relationships, qualified only by the requirement that integrity constraints be preserved. It might be permissible, for example, to replace the relationship "John has rank lecturer" by the relationship "Mary has rank professor" whether or not this corresponds to any meaningful real world event. The derivation of starts and ends of relationships from event descriptions imposes an extra level of semantic structure on database updates.

(3) Past and future are treated symmetrically. Therefore event descriptions can be assimilated in any order, independently of the order in which the events themselves actually take place. This facilitates dealing with incomplete information, both with new knowledge about the past as well as with hypothetical possibilities for the future. In the example, this is illustrated by the second update which records John's leaving without there being any previous record of his employment.

In a conventional database system the only way to express

"if a person leaves then he must already be employed"

is to formulate an integrity constraint which would reject as inconsistent any update which records an event of leaving without there already being an appropriate record of employment in the database. But such an integrity constraint combines (and confuses) two different kinds of statement: an object-level statement (about the world), that leaving implies a preceding period of employment, with a metalevel statement (about the database) that the database contains a complete record of relevant information about the past. In this paper we ignore problems concerned with the treatment of integrity constraints and preconditions of events.

(4) In the example narrative, a numerical time is associated with every event. In the general case this is not essential; rather it is the relative ordering of events which is important. Knowing the time at which events take place, of course, allows us to define an ordering relation "$<$" on events in a particularly simple way:

$e < e'$ if Time(e t) and Time(e' t')
and t is (chronologically) earlier than t'

In other cases, the event ordering relation can be defined explicitly, without reference to time. Indeed, in many cases it may not be possible to associate explicit times with events at all. For example, the meaning of the sentence

"John went to the theatre
after he came home from work."

can be represented by the event descriptions:

E1 is an event in which
John goes from work to home;
E2 is an event in which
John goes from home to the theatre;

together with a relative ordering of the two events

$E1 < E2$.

In the sequel we shall use the symbol "$<$" to signify both the ordering relation for events and the chronological (or other) ordering relation on times, and let context disambiguate between the two.

(5) The distinction between events, time periods and time instants makes it possible to deal with concurrent events. In our simple narrative we have such an example. The two events E2 and E4 are distinct even though they take place simultaneously.

(6) Although all the events considered in this paper can be treated as taking place instantaneously, we want to leave open the possibility that events can also have duration. For this reason we do not want time periods to contain wholly the events which start or end them. This is not quite the same as treating time periods as open intervals. Consider for example, an event of moving a block x from place y to place z, which consists in turn of five subevents: grasping, lifting, transporting, lowering, and ungrasping the block. The period for which x is at y ends when x is lifted and the period for which x is at z starts when x is lowered. The relationship between the event and time periods which we previously pictured (Figure 6)

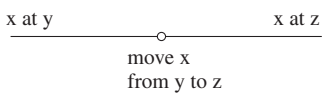

FIG. 6

can now be pictured (Figure 7).

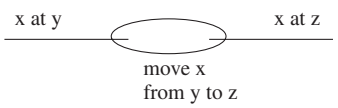

FIG. 7

To cater for this possibility it suffices to adopt the convention that events occur "after" the periods they end, "before" those they start, and are not wholly contained within them.

(7) Our formalization of events is intended as a formal analysis of the concepts rather than as a program or even as a program specification. Nonetheless, because it can be expressed by means of Horn clauses augmented with negation by failure, it is an executable analysis which in certain circumstances, after appropriate equivalence preserving transformations, runs as a PROLOG program.

(8) The most established alternative treatment of states and actions in classical, first-order logic is the situation calculus (McCarthy and Hayes 1969). Time varying relationships are qualified by a situation parameter, which can be regarded as a global, instantaneous time slice. Events transform one global situation into another.

Because situations are global, it is not possible to deal with simultaneous and partially ordered events. In the usual formalizations, it is difficult also to deal with incomplete information about a situation, and therefore to assimilate new information about the past.

The situation calculus, like the calculus of events, can be formalized by means of Horn clauses augmented with negation as failure (Kowalski 1979) and therefore can be executed as a PROLOG program. However, execution of the situation calculus gives rise to the **frame problem**, the need to reason that a relationship which holds in a situation and is not affected by an event continues to hold in the following situation. This explicit deduction, which is a consequence of the use of global situations, is so computationally inefficient as to be intolerable.

The event calculus was developed, to a large extent, in order to avoid the frame problem. It does so by qualifying relationships with time periods instead of with global

situations. Time periods associated with different relationships have different names even if they have the same duration.

(9) There is a vast, related literature (see Bolour et al. 1982) concerned with the formalization of time. Our presentation of the event calculus is similar to those treatments of time which are based on the use of time periods rather than on time instants. Among these, the approach of Allen (1981, 1984) is closest, not only because of its use of time periods, but more importantly because of its emphasis on events and the time periods they start and end. (Since writing this paper, we have discovered the still more closely related work of Lee, Coelho and Cotta (1985), which is also formulated within a logic programming framework.)

We have avoided the use of non-classical logic for two reasons: to obtain greater expressive power, and to exploit the proof procedures which have been developed for classical first-order logic in general and for logic programming in particular. Expressive power is gained by treating time and events explicitly rather than implicitly through the use of natural, but weak modal operators for notions such as "future", "since", and "while". We have potentially sacrificed the greater conciseness of modal logic for the greater expressiveness of an explicit treatment of time and events.

3 THE PROMOTION EXAMPLE IN DETAIL

Before considering the general case, we shall investigate the promotion example in greater detail.

The sequence of updates starting from the initially empty knowledge base can be represented by assertions:

Hire	(Mary lecturer E1)
Time	(E1 10.May. 1970)
Leave	(John lecturer E2)
Time	(E2 1.June. 1975)
Leave	(Mary professor E3)
Time	(E3 1.Oct. 1980)
Promote	(Mary lecturer professor E4)
Time	(E4 1.June. 1975)

The relationships which start or end as the result of events are defined by general rules:

Rank (x y after(e))	if Hire(x y e)	P1
Rank(x y before(e))	if Leave(x y e)	P2
Rank(x y before(e))	if Promote(x y z e)	P3
Rank(x z after(e))	if Promote(x y z e)	P4
Start(after(e) e)		P5
End(before(e) e)		P6

Notice that we have assumed for the time being that event descriptions are complete. In many cases incomplete event descriptions, such as

E2 is an event
in which John leaves,
E4 is an event
in which Mary is promoted to professor,

would be more natural. The advantage of complete event descriptions for the present is that they allow us to derive both started and ended relationships from the event descriptions alone. We shall deal with incomplete event descriptions later.

In order to conclude that

End(after(E1) E4)
End(after(E4) E3)
Start(before(E4) E1)
Start(before(E3) E4)

we need additional rules

End(after(e) e′) if after(e) = before(e′) P7
Start(before(e′) e) if after(e) = before(e′). P8

To derive that

after(E1) = before(E4)
after(E4) = before(E3)

we use the general rule

after(e) = before(e′) if Rank(x y after(e)) Temp1
and Rank(x y before(e′))
and $e < e'$
and not after(e) $\ll$ before(e′)

where

p1 $\ll$ p2

expresses that periods p1 and p2 are **disjoint**, with the end of p1 occurring before the start of p2.

In fact, this rule (and several of the earlier rules) will be generalized later in Section 10 to separate general axioms about events and time from those which are application specific. We shall introduce an axiom which expresses a general property of periods in the event calculus:

any two periods associated with the same relationship
are either identical, or they are disjoint.

(Note that Allen uses the same axiom.) Remembering that periods do not contain their end points, we can formalize the notion of disjoint periods as follows:

p1 $\ll$ p2 if End(p1 e) and Start(p2 e′) and $e \leq e'$ Temp2

Pictorially, the definition is illustrated in Figure 8.

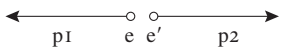

FIG. 8

Here $e \leq e'$ means that e occurs before or at the same time as e′.

Note that we allow the case where an event ends and starts the same relationship. For example the event of Mary's taking a sabattical can be regarded as ending one period of lectureship and starting another.

The negative condition in Temp1 can be interpreted either classically or by means of negation by failure. Interpreting it by negation as failure has the consequence that time periods are assumed to be equal by default, if they cannot be shown to be different.

4 EXECUTION OF THE PROMOTION EXAMPLE

Clauses P1-8 and Temp1–2 are in a form which can be executed as a PROLOG program. Unfortunately, when executed by PROLOG, the program goes into an infinite, non-terminating loop. Suppose for example that we have just the two event descriptions

Hire(Mary lecturer E1)
Promote(Mary lecturer professor E4)
$E1 < E4$

and pose the query

End(after(E1) x) ? Q1

using P1-8, Temp1-2 and an appropriate definition of $\leq$. The first three conditions of clause Temp1 are solved without difficulty, leaving the query

not after(E1) $\ll$ before(E4) ? Q2

To show this succeeds we must show that the query

after(E1) $\ll$ before(E4) ? Q3

fails. There is only one clause we can use, Temp2, and so we must show that the query

End(after(E1) e″) and Start(before(E4) e*) and e″ $\leq$ e* ? Q4

fails. PROLOG tries to solve the first condition first. But this is just like the original query, and PROLOG goes into a non-terminating loop.

It is possible to eliminate the loop, either by employing a more intelligent problem-solver than PROLOG or by using program transformation techniques. Before presenting a loop-free variant of the "program", however, we have a more serious problem to consider.

5 INCOMPLETENESS AND INCORRECTNESS OF START AND END

Negation by failure is a form of the **closed world assumption**, that the "knowledge base" is complete:

not p is judged to hold if all ways of showing p fail.

If the characterization of p is incomplete then not p may be judged to hold even though it does not. Unfortunately, our characterization of the "Start" and "End" predicates is incomplete. Consequently negation by failure can give incorrect results, allowing us to conclude that two time periods are equal when they are not.

Suppose, for example, that we are given the two event descriptions

Hire(Jack professor J1)
Hire(Jack professor J2)
$J1 < J2$

and nothing more (Figure 9).

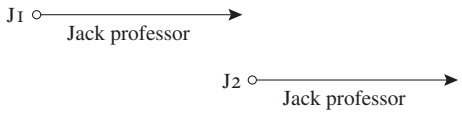

FIG. 9

Clearly some event, as yet unreported, must have occurred somewhere between J1 and J2 to end Jack's first period of professorship. Our existing rules could never find such an end for after(J1). Even if they did not loop, they would only be able to find ends which correspond to named, reported events. The rules we have for "End" are incomplete therefore; by symmetry, so are the ones for "Start".

The rule Temp1, by which we conclude that two periods are equal, relies on the completeness of the program for " $\ll$ ". The program for " $\ll$ ", Temp2, relies in turn on the completeness of "Start" and "End". This means that Temp1 may lead us to conclude that two periods are equal, when in fact we should not.

Suppose, for example, we add to the event descriptions above the information that

Leave(Jack professor J3)
$J2 < J3$.

Pictorially, we have the situation shown in Figure 10.

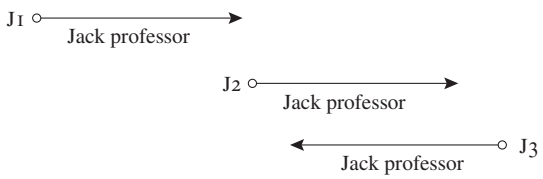

FIG. 10

Even if we eliminated loops, our existing rules could not find an end to after(J1), as argued above. Therefore, we could not show that periods after (J1) and before(J3) are disjoint, and so Temp1 would conclude they are equal. Clearly they are not.

The obvious solution is to complete the definition of the "End" and "Start" predicates. In this example we need rules which allow us to conclude that there exists some end j of after(J1), such that $J1 < j \leq J2$ (Figure 11).

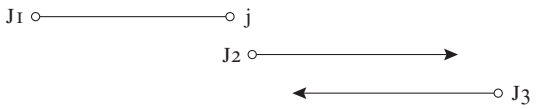

FIG. 11

In fact, as we shall see later, we need similar rules to conclude the existence of ends and starts of time periods in many other cases. In the meanwhile, however, we remark that the problems of incorrectness and looping can both be solved without having first to solve the problem of incompleteness.

If the predicate " $\ll$ " is not required for any other purpose, we can solve the problem by finding an alternative program for " $\ll$ ", which does not rely on the completeness of "Start" and "End". With such a program, the rules we have for "Start" and "End" would still be incomplete but now they would be correct.

Fortunately, there is such a program. It can be shown that, whenever

Rank(x y after(e))
Rank(x y before(e′))
$e < e'$

all hold, the condition

after(e) $\ll$ before(e′)

can be replaced by

[Rank(x y′ after(e*)) or Rank(x y′ before(e*))] and $e < e^* < e'$.

In other words the two time periods are disjoint if (and only if) some other event e* which affects the relationship takes place in between the start of one period and the end of the other.

Notice that the use of the variably y′ instead of y implicitly incorporates the "integrity constraint" that no person can hold more than one rank at the same time. We shall deal with such "incompatible" relationships in greater detail later when we deal with the general case.

With this result, we can dispense with the explicit definition of "$\ll$", and write instead

after(e) = before(e′) if Rank(x y after(e)) P9
and Rank(x y before(e′))
and $e < e'$
and not ([Rank(x y′ after(e*)) or
Rank(x y′ before(e*))]
and $e < e^* < e'$)

This is equivalent to

after(e) = before(e′) if Rank(x y after(e))
and Rank(x y before(e′))
and $e < e'$
and not[Rank(x y′ after(e*)) and $e < e^*$ and $e^* < e'$]
and not[Rank(x y′ before(e*)) and $e < e^*$ and $e^* < e'$]

This alternative to Temp 1-2 solves both the problem of looping and the problem of incorrectness.

Notice that rule P9 does not cover the case where the events e and e′ are too far apart in time for x to have rank y continuously from e to e′. To deal with this case we would need to add another condition to P9, such as

not Too-far-apart(e e′)

and define it appropriately.

6 TIME INSTANTS

Using P1-9 PROLOG can be used to determine the time periods for which relationships hold. To determine that a relationship holds at a time instant, however, we need additional rules such as

RankAt(x y t) if Rank(x y p) P10
and t in p

t in p if Start(p e1) and End(p e2) P11
and Time(e1 t1) and Time(e2 t2)
and $t1 < t$ and $t < t2$

Given the rules P1-11, an appropriate definition of $<$ for time instants, and the description of events E1-E4 in our simple narrative, we can conclude using PROLOG that, for example,

RankAt(Mary lecturer 11.May. 1970)
RankAt(Mary professor 16.Feb.1978).

The rules work for time periods which have a determined start and end. They do not work for periods which have no start or end, or for periods whose starts or ends are implied by other information but are not explicitly determined. These cases can be dealt with in a variety of ways and we shall return to them when we come to consider the general case.

7 A SPECIAL CASE OF THE PROMOTION EXAMPLE

The event calculus approach, the main features of which have been outlined above, may appear more complicated than necessary by comparison with conventional approaches to the treatment of database updates. This is partly because conventional databases deal with a **special case**: events are assimilated in the order in which they take place and the database is assumed to contain a complete record of all relevant past events. It is instructive, therefore, to see what simplifications can be made in the event calculus when we restrict ourselves to the same special case.

One of the most important simplifications is that P1-9 now constitute a complete definition of the troublesome "Start" and "End" predicates. This is because all relationships are first derived in the form

Rank(x y after(e))

before they are (redundantly) re-derived in the form

Rank(x y before(e)).

The existing definitions of "Start" and "End" cover this case. Moreover, as a further simplification, we can avoid the redundancy of deriving the same relationship twice, by restricting attention to the derivation of predicates of the form

Rank(x y after(e))
Start(after(e) e)
End(after(e) e′)

which are needed to characterize time periods of the form after(e). Clauses P1-9 can be replaced for these purposes by the clauses

Rank(x y after(e)) if Hire(x y e) P1
Rank(x z after(e)) if Promote(x y z e) P4
Start(after(e) e) P5
End(after(e) e′) if Rank(x y after(e)) and Leave(x y e′) P2′
and $e < e'$
and not [Rank(x y′ after(e*)) and $e < e^* < e'$]

End(after(e) e′) if Rank(x y after(e)) and Promote(x y z e′) P3′
and e < e′
and not [Rank(x y′ after(e*)) and e < e* < e′]

This is a significant simplification over P1-9.

The rules P10 and P11 express that a relationship holds at a particular time instant if it holds after the start of the relationship and before its end. It is appropriate in this special case to assume in addition that a relationship holds after it has started, provided it has not already ended:

t in p if Start(p e) P12
and Time(e t′)
and t′ < t
and not End(p e′)

Here, because the definition of "End" is complete for this special case, the negative condition in P12 does not lead to incorrect results, as it might in the more general case. (These rules are similar to those of Lee, Coelho, and Cotta (1985) who also use negation by failure, but restrict themselves to this special case.)

8 INCOMPLETE EVENT DESCRIPTIONS

For the purpose of simplicity we have assumed that event descriptions are sufficiently complete to derive, directly from the event description alone, the relationships which are started and ended by the event. In many cases, however, **incomplete** event descriptions such as

E2 is an event in which John leaves,

where there is insufficient information to determine directly what John's rank was when he left, are more natural.

The analysis of natural language by means of semantic networks and semantic cases suggests a way of dealing with such incomplete event descriptions. An event such as

"John gave the book to Mary",

for example, can be represented as a network (Figure 12)

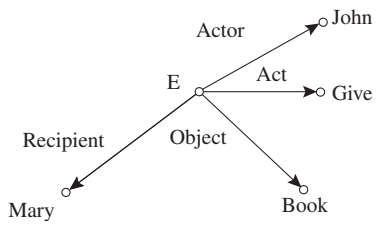

FIG. 12

which can be formalized in turn by using constant symbols to represent nodes and binary predicates to represent arcs:

Actor(E John)
Act(E Give)

Object(E Book)
Recipient(E Mary).

Missing information can be dealt with by representing only the information which is known and ignoring that which is unknown. For example, to represent that Mary was promoted on 1 June 1975:

Act(E4 Promote)
Object(E4 Mary)
Time(E4 1.June.1975).

The advantages of using semantic networks to describe events and of representing such networks in formal logic have been discussed by several authors. The discussion in Kowalski (1979) is especially relevant here.

The clauses P1-4 which presently require complete event descriptions can be modified so that they use the minimum number of conditions needed to establish the conclusion. P1-4 can then be replaced by

Rank(x y after(e))	if Act(e hire) and Object(e x) and Destination(e y)	P1′
Rank(x y before(e))	if Act(e leave) and Object(e x) and Source(e y)	P2′
Rank(x y before(e))	if Act(e promote) and Object(e x) and Source(e y)	P3′
Rank(x y after(e))	if Act(e promote) and Object(e x) and Destination(e y)	P4′

Thus, for example, P4′ does not require the condition

Source(e z)

which identifies the "object's" rank immediately before promotion. The remaining clauses are not affected by this reformulation.

Notice that the new formulation is still symmetric with respect to past and future. However, whereas a complete event description allows us to deduce all possible relationships which are started or ended by an event, an incomplete description might not contain sufficient information to allow such deductions. Nonetheless, it may be possible to complete such an event description by default reasoning.

Suppose, for example, that we are given complete descriptions of the events E1, E2, and E3 as before and then an incomplete description of E4:

Act(E4 promote)
Object(E4 Mary)
Time(E4 1.June.1975).

Pictorially the situation is shown in Figure 13.

The information about E4 is insufficient to allow the derivation of the conclusion

Rank(Mary lecturer before(E4))

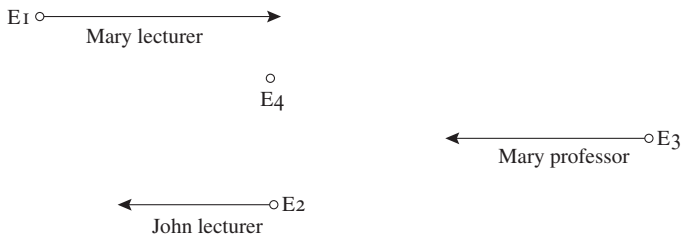

FIG. 13

by means of P3′ and therefore of the further conclusion

End(after(E1) E4).

We can derive these conclusions, however, if we can find a means of completing the event description by deducing

Source(E4 lecturer).

We can do so by adding extra information about promotions: in every event of promotion there must be a "source", even though it may be unknown.

This extra information allows us to deduce that, in event E4, Mary must have been promoted from some rank, and therefore that Mary holds some (unknown) rank throughout the period before(E4). Pictorially we have the situation in Figure 14.

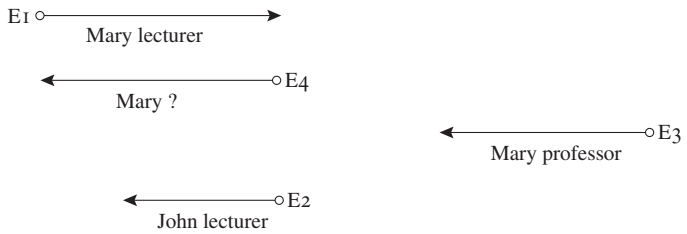

FIG. 14

Mary's ranks during periods after(E1) and before(E4) may be different, or they may be the same.

It is a natural extension of our previous use of default reasoning to assume now that

two ranks are identical
if we cannot show they are different.

This argument justifies adding the extra rule:

Source(e y) if Act(e promote)
and Object(e x)
and Rank(x y after(e′))
and $e' < e$
and not ([Rank(x y′ after(e*)) or
Rank(x y′ before(e*))]
and $e' < e^* < e$)

which uses the negative condition to reason by default.

Similarly we can use the additional information that every event of promotion has a "destination" (persons are promoted to **some** rank) to justify the extra rule:

Destination(e y) if Act(e promote)
and Object(e x)
and Rank(x y before(e′))
and $e < e'$
and not ([Rank(x y′ after(e*)) or
Rank(x y′ before(e*))]
and $e < e^* < e'$)

This allows us to deduce

Destination(E4 professor)
Start(before(E3) E4)
End(after(E4) E3).

These conclusions are shown pictorially in Figure 15.

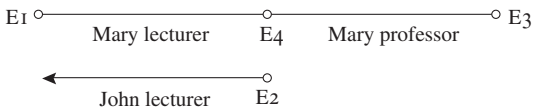

FIG. 15

As usual, conclusions based on default assumptions may be automatically withdrawn after the assimilation of new information.

9 ANOTHER EXAMPLE

Before turning to the general case, it is useful to consider an example in which an event starts and ends more than one relationship. Consider the following narrative:

John exchanged his orange for Mary's apple.
Then Mary exchanged the orange for a pear.

An act of exchanging has two actors and two objects. Suppose we call them the actor, coactor, object and coobject. We can then formalize the narrative by the clauses

Act(E1 exchange)	Act(E2 exchange)
Actor(E1 John)	Actor(E2 Mary)
Object(E1 orange)	Object(E2 orange)
Coactor(E1 Mary)	Coobject(E2 pear)
Coobject(E1 apple)	$E1 < E2$

Notice that since each exchange event e starts and ends two relationships we need to distinguish the two periods associated with the relationships. We can do so by using terms before(e x) and after(e x), where the second parameter x distinguishes between the two periods. One of the easiest ways of doing this is to use the name of the actor or coactor as the second parameter.

Possesses(x y before(e x)) if Act(e exchange) Ex1
and Actor(e x)
and Object(e y)

Possesses(x y before(e x))	if	Act(e exchange)	Ex2
	and	Coactor(e x)	
	and	Coobject(e y)	
Possesses(x y after(e x))	if	Act(e exchange)	Ex3
	and	Coactor(e x)	
	and	Object(e y)	
Possesses(x y after(e x))	if	Act(e exchange)	Ex4
	and	Actor(e x)	
	and	Coobject(e y)	

In the given example, these clauses allow us to derive

Possesses(John orange before(E1 John))
Possesses(Mary apple before(E1 Mary))
Possesses(John apple after(E1 John))
Possesses(Mary orange after(E1 Mary))
Possesses(Mary orange before(E2 Mary))
Possesses(Mary pear after(E2 Mary))

To derive starts and ends of time periods we need, to begin with, the clauses

Start(after(e x) e) Ex5
End(before(e x) e). Ex6

To conclude

after(E1 Mary) = before(E2 Mary)

and therefore that

End(after(E1 Mary) E2)
Start(before(E2 Mary) E1)

we need the clauses

after(e x) = before(e′ x)	if	Possesses(x y after(e x))		Ex7
	and	Possesses(x y before(e′ x))		
	and	e < e′		
	and	not([Possesses(x′ y after(e* x′)) or Possesses(x′ y before(e* x′))] and e < e* < e′)		
End(after(e x) e′)	if	after(e x) = before(e′ x)		Ex8
Start(before(e′ x) e)	if	after(e x) = before(e′ x)		Ex9

Here the negative condition in Ex7 also incorporates the constraint that more than one person cannot "possess" an object at one time.

10 THE GENERAL CASE

We are now in a position to generalize the preceding examples and consider the general case. For this purpose, in order to deal uniformly with events which start or end more than one relationship, it is convenient to name time periods by means of terms

after(e u) and before(e u)

where the second parameter u names the relationship associated with the time period. Moreover, instead of treating time periods as a parameter of time-varying relations, it is convenient to use a general predicate

Holds(p)

which expresses that the relationship associated with p holds for the time period p. Thus we will now write

Holds(before(E2 rank(John lecturer)))

instead of the earlier, simpler notation

Rank(John lecturer before(E2)).

Although in most cases the new notation will be more complicated than necessary, it has the advantage of greater generality. This notation is similar to one we have used elsewhere for the situation calculus. (Kowalski 1979).

Instead of writing rules such as

```
Holds(before(e rank(x y))) if   Act(e leave)
                            and Object(e x)
                            and Source(e y)
Holds(before(e possesses(x y))) if   Act(e exchange)
                                 and Actor(e x)
                                 and Object(e y)
```

similar to those we have written before, we can write a single general rule and several specific rules for different applications:

```
Holds(before(e u)) if Terminates(e u)                                  G1
Terminates(e rank(x y)) if   Act(e leave)
                         and Object(e x)
                         and Source(e y)
Terminates(e possesses(x y)) if   Act(e exchange)
                              and Actor(e x)
                              and Object(e y).
```

Similarly

```
Holds(after(e u)) if Initiates(e u)                                    G2
Initiates(e rank(x y)) if   Act(e hire)
                       and Object(e x)
                       and Destination(e y)
Initiates(e possesses(x y)) if   Act(e exchange)
                             and Actor(e x)
                             and Coobject(e y)
```

Notice, however, that to achieve such generality we have had to introduce the new predicates "Initiates" and "Terminates".

The remaining rules are very similar to those we have used for the preceding examples:

```
Start(after(e u) e)                                                    G3
End(before(e u) e)                                                     G4
Start(before(e′ u) e)  if after(e u) = before(e′ u)                    G5
```

End(after(e u) e′) if after(e u) = before(e′ u) G6

after(e u) = before(e′ u) if Holds(after(e u))
and Holds(before(e′ u))
and e < e′
and not Broken(e u e′) G7

Broken(e u e′) if Holds(after(e* u*)) and Exclusive(u u*)
and e < e* < e′ G8

Broken(e u e′) if Holds(before(e* u*)) and Exclusive(u u*)
and e < e* < e′ G9

Here "Broken" has been introduced largely as an abbreviation for reuse later on. It is intended that the predicate Exclusive(u u′) holds when the relationships u and u′ are either identical or incompatible in the sense that not both can hold simultaneously, i.e.

Exclusive(u u)
Exclusive(u u′) if Incompatible(u u′)

The predicate "Incompatible" needs to be defined by specific rules for particular applications. For example

Incompatible(rank(x y) rank(x y′)) if not y = y′
Incompatible(possesses(x y) possesses(x′ y)) if not x = x′

y = y.

(Notice that to deal with the case that e and e′ are too far apart for u to hold continuously from e to e′ we could add extra application-specific rules for the "Broken" predicate.)

To determine that a relationship holds at a time instant we need to modify P10:

HoldsAt(u t) if Holds(after(e u))
and t in after(e u)
HoldsAt(u t) if Holds(before(e u))
and t in before(e u)

The rule P11

t in p if Start(p e1) and End(p e2)
and Time(e1 t1) and Time(e2 t2)
and t1 < t and t < t2

is adequate as it stands. As before, the rule P12

t in p if Start(p e)
and Time(e t′)
and not End(p e′)

is appropriate and not incorrect for the special case where events are recorded in the order in which they occur and the database contains a complete record of all relevant past events (and the time between t and t′ is not too long for the relationship concerned to hold continuously). However it is incorrect in the general case because our definition of the "End" (as well as "Start") predicate is incomplete. We shall attempt to remedy this defect now.

11 OTHER CASES OF THE START AND END PREDICATES

So far we have rules for the cases

Start(after(e u) e)
End(before(e u) e).

We also have rules which derive end points when time periods are identical (Figure 16):

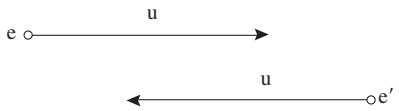

FIG. 16 Case 0.

There are other, more difficult, cases which we shall now consider. Pictorially these are shown in Figures 17–19.
In Fig. 17 u and u′ are "exclusive" in the sense defined in Section 10.

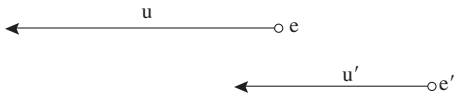

FIG. 17 Case 1.

In Fig. 18 u and u′ are exclusive. (This case is symmetric to case 1.)

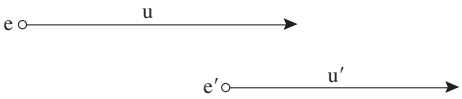

FIG. 18 Case 2.

In Fig. 19 u and u′ are "incompatible".

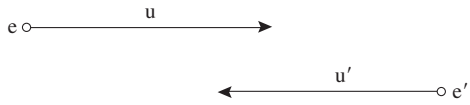

FIG. 19 Case 3.

It can be argued that these four cases exhaust all the situations where time periods interact to imply the existence of end points. In fact, the rules for determining end points in all four cases 0–3 can be systematically derived from a small number of general principles, the most important of which are:

p1 = p2 or p1 ≪ p2 or p2 ≪ p1 if p1 instance of u1 — Ax1
and p2 instance of u2
and Exclusive(u1 u2)

not[p1 = p2] if p1 instance of u1 — Ax2
and p2 instance of u2
and Incompatible(u1 u2)

after(e u) instance of u if Holds(after(e u)) Ax3

before(e u) instance of u if Holds(before(e u)) Ax4

$x < y$ if Start(p x) and End(p y) Ax5

$p1 \ll p2$ if and only if Ax6
there exist e1 and e2 [End(p1 e1) and
Start(p2 e2) and
$e1 \leq e2$]

Notice that we have previously made use of the "if half" of Ax6 to determine end points in case 0. To determine end points in cases 1–3 we need to use the "only if half". We will not show the derivation of the rules here, but only present the rules themselves.

In **Case 1**, there must exist a start i of before(e′ u′), at or after e. Pictorially it is shown in Figure 20.

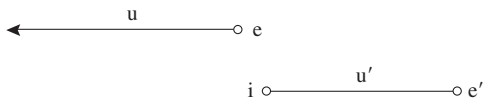

FIG. 20

The new end point can be named as a function of the time period, say init(before(e′ u′)), and the end point can be derived by the general rule

[Start(before(e′ u′) init(before(e′ u′)))
and
$e \leq$ init(before(e′ u′))]
if Holds(before(e u))
and Holds(before(e′ u′))
and Exclusive(u u′)
and $e < e'$
and not Broken(e u′ e′)

Here we have used the notation

[A and B] if C

as shorthand for the two clauses

A if C
B if C.

Case 2 is similar to case 1:

[End(after(e u) fin(after(e u)))
and
fin(after(e u)) $\leq$ e′]
if Holds(after(e u))
and Holds(after(e′ u′))
and Exclusive(u u′)
and $e < e'$
and not Broken(e u e′)

Notice that an attractive consequence of the use of negation as failure is that the **implicit end point** derived by these rules disappears if new information makes it possible to derive the end point explicitly.

Case 3 is similar to cases 1 and 2 but slightly more complicated. In this case there exists an end of after(e u) at or before the start of before(e′ u′). These implicit start and end points are shown pictorially in Figure 21.

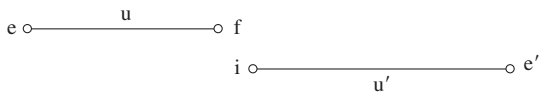

FIG. 21

[fin(after(e u)) ≤ init(before(e′ u′))
and
Start(before(e′ i′) init(before(e′ u′)))
and
End(after(e u) fin(after(e u)))]
if Holds(after(e u))
and Holds(before(e′ u′))
and Incompatible(u u′)
and e < e′
and not Broken(e u e′)

These clauses complete the definition of the "Start" and "End" predicates.

Notice, however, that our treatment in cases 1 and 2 of both identical and incompatible relationships in the same way suggests the possibility of extending case 3 to include the case where u and u′ are identical.

This would mean that in the situation (Figure 22)

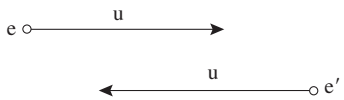

FIG. 22

where we earlier concluded that

after(e u) = before(e′ u)

we would need non-Horn clause logic to express that either the equality holds, or the period after(e u) ends before the period before(e′ u) starts. Such an expression would have the form

(A or B) if C

where A and B are mutually exclusive. The approach we have taken so far, which rewrites the statement in the form

A if C and not B

and interprets negation as failure, gives disjunction an asymmetric interpretation:

prefer conclusion A to conclusion B
unless it is inconsistent to do so.

12 CONCLUSION

The event calculus attempts to provide a general framework for reasoning about time and events. It is based upon general axioms concerning the notions of events, relationships, and the periods for which they hold. In this paper, we have presented some consequences of these axioms which can be executed as a PROLOG program.

In order to deal with simultaneous and partially ordered events, and to impose semantic structure on knowledge base transitions, events are treated as primitive concepts, and knowledge base states are derived from event descriptions. Event descriptions are symmetric with respect to past and future, implying information about past states as well as about future ones.

In this paper we have concentrated on applications of the event calculus to assimilating both database updates and simple narratives. In particular, we stressed how default reasoning, implemented by negation as failure, deals with the case in which event descriptions are assimilated independently of the order in which they occur. When an update description conflicts with information derived by default reasoning, the update is accepted and the conflicting information previously derived by default is automatically and non-monotonically withdrawn.

In contrast, conventional databases choose to reject updates which are inconsistent with information already in the database. This strategy is appropriate only when updates are reported and assimilated in the order in which they occur, and when the database can be assumed to hold complete information about the past. Making explicit these extra assumptions in the event calculus simplifies the treatment significantly. We have not discussed, however, the processing which is necessary in these circumstances to validate attempted updates and to avoid the violation of database integrity.

These two contrasting approaches to database updates represent extreme ends of a spectrum of possibilities. In general, database systems faced with an attempted update inconsistent with their contents could choose to restore consistency either by rejecting the update or by withdrawing some of the information in the database.

The clauses we presented for assimilating updates and narratives run reasonably efficiently as a PROLOG program. However, they should be regarded not as a program but as a specification. In practice, the clauses would be further transformed and optimized to run more efficiently in specific applications.

A number of extensions can be incorporated straightforwardly into the event calculus. In particular, it is possible to extend the representation of periods to deal with information like

"Mary was a professor when Jim was promoted"

where neither the start nor the end of her period of professorship is known. Important extensions which do need further investigation include the representation of negated facts, and the ability to associate arbitrary sentences, not just conditionless facts, with the periods for which they hold.

Our formalization of the event calculus is deliberately neutral with respect to whether or not events have duration. Fariba Sadri has investigated the treatment of events which have duration, so that we can say, for example, that one event occurs completely, or partially, while another is taking place.

Somewhat more speculatively perhaps, we believe that the assimilation of updates without explicit deletion will contribute to the problem of updating data structures

without destructive assignment in logic programming itself. These and other applications remain to be investigated in greater detail.

ACKNOWLEDGEMENTS

We are indebted to Fariba Sadri for her valuable comments, particularly those concerned with completing the definitions of the "Start" and "End" predicates. We also owe thanks to Dov Gabbay for his useful and stimulating discussions about temporal logic and its relationship with classical logic.

This research was supported by the Science and Engineering Research Council.

REFERENCES

Allen, J. F. (1981). 'Maintaining knowledge about temporal intervals', *TR-86*, Computer Science Dept., University of Rochester, January, 1981. Also in *Communications of the (II) (1983): ACM, 26*. 832–43.

——(1984). 'Towards a General Theory of Action and Time', *Artificial Intelligence 23:* 123–54.

Bolour, A., Anderson, T. L., Dekeyser, L. J., and Wong, H. K. T. (1982). 'The role of time in information processing: a survey', *ACM SIGMOD Review 12 (3, April)*.

Kowalski, R. A. (1979). *Logic for Problem Solving*. New York: North-Holland/Elsevier.

Lee, R. M., Coelho, H., and Cotta, J. C. (1985). 'Temporal Inferencing on Administrative Databases', *Information Systems 10 (2):* 197–206.

McCarthy, J. and Hayes, P. J. (1969). 'Some philosophical problems from the standpoint of artificial intelligence', in B. Meltzer and D. Michie, (eds.), *Machine Intelligence, 4*. Edinburgh: Edinburgh University Press, 463–502.