

# THE EVENT CALCULUS

MICHIEL VAN LAMBALGEN

## 1. ONTOLOGY FOR THE EVENT CALCULUS

The event calculus is an attempt to codify intuitive reasoning about action and change in such a way that the frameproblem is avoided. It is based upon the supposition that all change must be due to a cause—spontaneous changes do not occur. In this way it achieves a formalization of the *common sense principle of inertia*: ‘normally, nothing changes’. That is, if an action  $a$  does not affect a property  $F$ , then if  $F$  is true before doing  $a$ , it will be true after. Of course, the meat in this intuitive idea is the notion of ‘affect’. This refers to a kind of causal web which specifies the influences of actions on properties. The full force of this causal web appears if the event calculus is formulated in logic programming with negation as failure.

Formally, the event calculus requires a many-sorted first order logic with sorts for the following:

- (1) individual objects (humans, chairs, tables, ...)
- (2) real numbers, to represent time and variable quantities
- (3) time-dependent properties, such as states and activities
- (4) variable quantities
- (5) event types, whose instantiations mark the beginning and end of time-dependent properties.

The event calculus actually formalizes two notions of cause, and their relation. The first notion of cause is concerned with instantaneous change, as when two balls collide. We are thus concerned with an event (type) *collision*, which for simplicity is assumed to occur instantaneously. An event *type* together with a time at which it occurs (or happens) will be referred to as an event *token*. We furthermore need time-dependent properties such as, for example, ‘ball  $b$  has momentum  $m$ ’. In the case at hand, the property ‘ball 1 has momentum  $m$  and ball 2 has momentum 0’ will be true until the time of collision  $t$ , after which ‘ball 2 has momentum  $m$  and ball 1 has momentum 0’ is true. Such time-dependent properties are called *fluents*<sup>1</sup>. A fluent is a function which may contain variables for individuals and reals, and which is interpreted in a model as a set of time points. We now want to be able to say that fluents are initiated and terminated by events, and that a fluent was true at the beginning of time. If  $f$  is a variable over fluents,  $e$  a variable over events, and  $t$  a variable over time points, we may write the required predicates as

- (1) *Initially*( $f$ )
- (2) *Happens*( $e, t$ )
- (3) *Initiates*( $e, f, t$ )

---

<sup>1</sup>The name is appropriated from Newton’s treatise on the calculus, where all variables are assumed to depend implicitly on time.

(4) *Terminates*( $e, f, t$ )

These predicates are to be interpreted in such a way, that if  $Happens(e, t) \wedge Initiates(e, f, t)$ , then  $f$  will begin to hold after (but not at)  $t$ ; if  $Happens(e, t) \wedge Terminates(e, f, t)$ , then  $f$  will still hold at  $t$ <sup>2</sup>.

The second notion of causality is more like change due to a force which exerts its influence continuously. The paradigmatic example here is the acceleration of an object due to the gravitational field, but other examples abound: pushing a cart, filling a bucket, drinking a glass of wine, writing a letter, ... Continuous change requires its own special predicates, namely

5 *Trajectory*( $f_1, t, f_2, d$ )6 *Releases*( $e, f, t$ )

In the *Trajectory* predicate, one should think of  $f_1$  as a force, and of  $f_2$  as a variable quantity which may change under the influence of the force. The predicate then expresses that if  $f_1$  holds from  $t$  until  $t + d$ , then at  $t + d$ ,  $f_2$  holds. In applications,  $f_2$  will have a real number as argument, and will be of the form  $f(g(t + d))$  for some continuous function  $g$ .

The predicate *Releases* is necessary to reconcile the two notions of cause with each other. Cause as instantaneous change leads to one form of inertia: after the occurrence of the event marking the change, properties will not change value until the occurrence of the next event. This however conflicts with the intended notion of continuous change, where variable quantities may change their values without concomitant occurrences of events. The solution is to exempt those properties which we want to vary continuously from the inertia of the first form of causation by means of a special predicate, *Releases*.

The axioms will be seen to have the form: if there are no ‘ $f$ -relevant’ events between  $t_1$  and  $t_2$ , then the truth value of  $f$  at  $t_1$  is the same as that at  $t_2$ . We introduce two special predicates to formalize the notion of ‘ $f$ -relevant’ events. The first predicate expresses that there is a terminating or releasing event between  $t_1$  and  $t_2$ ; the second predicate expresses that there is an initiating or releasing event between  $t_1$  and  $t_2$ .

7 *Clipped*( $t_1, f, t_2$ )8 *Declipped*( $t_1, f, t_2$ )

Lastly, we need the truth predicate

9 *HoldsAt*( $f, t$ ).

In order to derive predictions, e.g. on when a robot will reach its destination, one needs additional formulas, conveniently divided in axioms, holding for every situation, and a scenario, laying down properties of a particular situation. We first study the axioms.

## 2. THE AXIOM SYSTEM EC

The axioms of the event calculus given below are modified from [?], the difference being due to the fact that we prefer a logic programming approach, whereas Shanahan uses circumscription. In the following, all variables are assumed to be universally quantified.

<sup>2</sup>Strictly speaking this convention makes sense only for events which are not extended in time. For the general case one needs some axioms additional to those presented below; we will omit these for the sake of simplicity.

**Axiom 1.**  $Initially(f) \wedge Clipped(r, f, t) \rightarrow HoldsAt(f, 0)$

**Axiom 2.**  $Happens(e, t) \wedge Initiates(e, f, t) \wedge t < t' \wedge \neg Clipped(t, f, t') \rightarrow HoldsAt(f, t')$

**Axiom 3.**  $Happens(e, t) \wedge Terminates(e, f, t) \wedge t < t' \wedge \neg Declipped(t, f, t') \rightarrow \neg HoldsAt(f, t')$

**Axiom 4.**  $Happens(e, t) \wedge Initiates(e, f_1, t) \wedge t < t' \wedge t' = t + d \wedge Trajectory(f_1, t, f_2, d) \wedge \neg Clipped(t, f_1, t') \rightarrow HoldsAt(f_2, t')$

**Axiom 5.**  $Happens(e, s) \wedge t < s < t' \wedge (Terminates(e, f, s) \vee Releases(e, f, s)) \rightarrow Clipped(t, f, t')$

**Axiom 6.**  $Happens(e, s) \wedge t < s < t' \wedge (Initiates(e, f, s) \vee Releases(e, f, s)) \rightarrow Declipped(t, f, t')$ .

It will be observed that the syntactic form of the axioms is such that they can be used in logic programming plus negation as failure: apart from axiom 4, all axioms have an atomic formula as head, and literals in the body. Actually, 4 is useful but superfluous, since its effect can be obtained by using negation as failure.

The set of axioms of the event calculus will be abbreviated by *EC*. In the absence of further axioms, one can construct a model for *EC* by interpreting fluents as finite unions of halfopen intervals (of the form  $(a, b]$ ), and assuming that each event either initiates or terminates a fluent, and that fluents are initiated or terminated by events only.

### 3. SCENARIOS

The structure of specific situations must be described by micro-theories in the language of the event calculus. There is a canonical way to do this, given by means of three definitions.

**Definition 1.** A state  $S(t)$  at time  $t$  is a conjunction of literals involving only

- (1) literals of the form  $(\neg)HoldsAt(f, t)$ , for  $t$  fixed and possibly different  $f$ ,
- (2) equalities between fluent terms, and between event terms
- (3) equations and inequalities involving real numbers

**Definition 2.** A scenario is a conjunction of statements of the form

- (1)  $Initially(f)$ , or
- (2)  $\forall t(S(t) \rightarrow Initiates(e, f, t))$ , or
- (3)  $\forall t(S(t) \rightarrow Terminates(e, f, t))$ , or
- (4)  $\forall t(S(t) \rightarrow Releases(e, f, t))$ , or
- (5)  $\forall t(S(t) \rightarrow Happens(e, t))$ ,

where  $S(t)$  is a state in the sense of definition 1. These formulas may contain additional constants for objects<sup>3</sup>, reals or time points and can be prefixed by universal quantifiers over time points, reals and objects.

**Definition 3.** A dynamics is a statement of the form  $S(f_1, f_2, t, d) \rightarrow Trajectory(f_1, t, f_2, d)$ , where  $S(f_1, f_2, t, d)$  is a state in the sense of definition 1.

<sup>3</sup>We do not allow functions on objects.

## 4. EXAMPLE: CAUSE AS INSTANTANEOUS CHANGE

Consider a room in which there are two lights and two switches, each serving one light. We have the following event types and fluents:

- (1)  $a_1 = \text{switch}_1 \text{ on}$
- (2)  $a_2 = \text{switch}_1 \text{ off}$
- (3)  $e_1 = \text{switch}_2 \text{ on}$
- (4)  $e_2 = \text{switch}_2 \text{ off}$
- (5)  $f_1 = \text{light}_1 \text{ on}$
- (6)  $f_2 = \text{light}_2 \text{ on}$

At time 5 light 1 is switched on, at time 10 it is switched off. Light 2 is on initially. Describe the light situation at times  $t$  for  $0 \leq t \leq 15$ . Does anything change if we do not specify anything about light 2?

## 5. EXAMPLE: CAUSE AS CONTINUOUS CHANGE.

Suppose we want to show that a bucket of height 10 units, into which water flows continuously, will ultimately overflow. This can be formalised by assuming fluents *filling*, a fluent function  $height(x)$  (where  $x \in \mathcal{N}$ ), a fluent *spilling* and events *overflow*, *tap-on*, *tap-off* which are connected by axioms such as the following (this list is not exhaustive)

- (1)  $Initially(height(0))$
- (2)  $Happens(tap-on, 5)$
- (3)  $Initiates(tap-on, filling, t)$
- (4)  $Terminates(overflow, filling, t)$
- (5)  $HoldsAt(height(10), t) \wedge HoldsAt(filling, t) \rightarrow Happens(overflow, t)$
- (6)  $Releases(tap-on, height(x), t)$ .

For the following, assume that time ranges over the natural numbers.

5.1. **Exercise.** Complete the specification under the assumption that after  $d$  units of time the water level has also increased  $d$  units; this gives a *scenario*.

5.2. **Exercise.** Try to show that  $HoldsAt(height(10), 20)$ . What do you need?

5.3. **Exercise.** Suppose we would add the statements  $Happens(tap-off, 10)$  to the above list. Would  $HoldsAt(height(10), 20)$  still be derivable?

5.4. **Exercise.** What would happen if the statement  $Releases(tap-on, height(x), t)$  were left out?

5.5. **Exercise.** Give a complete description of the model of EC plus the scenario; that is, describe what happens when, and which fluents are true or false at which time points.

## 6. NEGATION AS FAILURE AND THE COMPLETION OF A PROGRAM

The axioms of the event calculus are not Horn clauses, due to the occurrence of the negative literals  $\neg Clipped(s, f, t)$  and  $\neg Declipped(s, f, t)$  in the bodies of the axioms. Also, in Shanahan's paper 'Reinventing Shakey' some statements describing the situation at hand, e.g. K4.3 and K4.6, use negation in the body. This means

that unit resolution does not suffice as a derivation mechanism. One possible mechanism for handling negation is *negation as failure*<sup>4</sup>. To explain this notion, we need a few definitions.

**Definition 4.** A (general) query is a finite sequence of literals.

**Definition 5.** A (general) rule is an expression  $M \rightarrow A$ , where  $A$  is an atom (the head) and  $M$  is a (general) query (the body). A (general) program is a finite set of such rules.

Conceived of as a clause, i.e. a disjunction of literals, a general rule is thus a clause with one designated positive literal, the head. Note that general rules, conceived of as clauses, may contain several positive literals, so that there may be several different ways to write a clause as a rule (by singling out a different head in each case).

Now suppose we are given a program  $P$  consisting of general rules, and a general query  $?A_1, \dots, A_n, \neg B_1, \dots, \neg B_k$ , where the  $A_i, B_j$  are positive. We have to define the action of  $P$  on the query, and to investigate what it means semantically. A formally precise definition of negation as failure would be too complex, so we give a definition that works for our case. An excellent reference for a fuller treatment is [?].

As always, the goal of a derivation is to derive the empty clause. Thus, we have to have a mechanism for erasing a literal from a query. For positive literals we can still use unit resolution. For a negative literal  $\neg B_j$ , we start a derivation beginning with the query  $B_j$ . If the resulting resolution tree is finite, but does not have the empty clause at one of its end nodes, then we say that the query  $B_j$  fails finitely. In this case, we may erase  $\neg B_j$  from the query  $?A_1, \dots, A_n, \neg B_1, \dots, \neg B_k$ . If the resulting resolution tree does have the empty clause at one of its end nodes, then the query  $B_j$  is successful. In this case the query  $\neg B_j$  fails, and therefore also the query  $?A_1, \dots, A_n, \neg B_1, \dots, \neg B_k$ .

The attentive reader will have noticed that the above attempt at a definition really hides an inductive characterisation, since the derivation tree starting from the query  $B_j$  may itself involve applications of negation as failure. This may lead to infinities, for example when  $P = \{\neg A \rightarrow A\}$  and the query is  $?A$ .

As we have seen, the operational definition of negation as failure is reasonably straightforward; but what does it *mean*? In order to explain the question, let us briefly return to ordinary Prolog programs  $P$ . In this case, unit resolution is sound and complete, that is, a derivation starting from a query  $?A$  is successful if and only if  $P \models A$ . One would like to have as well  $? \neg A$  is successful if and only if  $P \models \neg A$ , but this is not true in general. Here is a counterexample: let  $P = \{\forall x(B(x) \rightarrow A(x))\}$  and consider the query  $? \neg A(a)$ . Since  $?A(a)$  fails,  $? \neg A(a)$  is successful, but  $\neg A(a)$  does not follow logically from  $P$ . Let us look at the precise reason why  $\neg A(a)$  does not follow from  $\forall x(B(x) \rightarrow A(x))$ . A countermodel is given by making  $A(a), B(a)$  true. However, this model is rather luxurious, since it assumes the truth of  $B(a)$ , something not given by  $P$ . By contrast, the *minimal* model of  $P$  has  $A, B$  both empty; hence  $\neg A(a)$  is true on the minimal model. Note that in this case the minimal model of  $P$  is axiomatised by  $\forall x(A(x) \leftrightarrow B(x)) \wedge \forall x \neg B(x)$ . This is an instance of a general procedure called *completion* of a program  $P$ .

<sup>4</sup>There are others, such as the closed world assumption, or constructive negation.

Given a program  $P$  the sets of sentences  $IF(P)$  and  $IFF(P)$  are constructed from  $P$  by means of the following steps. Fix a sequence of new variables  $x_1, x_2, \dots$

*Step 1 Remove terms from rule heads.*

Replace every rule of the form  $B_1 \wedge \dots \wedge B_m \rightarrow r(t_1, \dots, t_k)$  by  $B_1 \wedge \dots \wedge B_m \wedge x_1 = t_1 \wedge \dots \wedge x_k = t_k \rightarrow r(x_1, \dots, x_k)$ .

*Step 2 Introduce existential quantifiers.*

Transform each formula  $F \rightarrow r(x_1, \dots, x_k)$  obtained in the previous step into  $\exists y_1 \dots \exists y_n F \rightarrow r(x_1, \dots, x_k)$ , where the  $y_1, \dots, y_n$  are all the free variables of  $F$  minus  $x_1, \dots, x_k$ .

*Step 3 Group formulas with the same head*

If  $F_1 \rightarrow r, \dots, F_l \rightarrow r$  are all the formulas with head  $r$ , replace them by a single formula  $F_1 \vee \dots \vee F_l \rightarrow r$ . This formula will be called the *definition* of  $r$ .

*Step 4 Handle undefined relation symbols*

If a relation symbol  $r(x_1, \dots, x_i)$  does not occur as head of a rule in  $P$ , replace  $r$  by  $\forall x_1 \dots \forall x_i \neg r(x_1, \dots, x_i)$ .

*Step 5 Replace each formula by its universal closure.*

This gives us the set  $IF(P)$ .

*Step 6 IFF(P)*

To obtain  $IFF(P)$  from  $IF(P)$ , replace each  $\rightarrow$  by  $\leftrightarrow$ .

Lastly, to obtain the completion  $comp(P)$  from  $IFF(P)$ , we have to add *uniqueness of names* assumptions  $UNA$ , also called the Clark Equality Theory  $CET$ . This is the following set of statements for function symbols  $f, g$  and terms  $t$  of the language:

- (1)  $f(y_1, \dots, y_n) = f(z_1, \dots, z_n) \rightarrow y_1 = z_1 \wedge \dots \wedge y_n = z_n$
- (2)  $f(y_1, \dots, y_n) \neq g(z_1, \dots, z_m)$ , where  $f, g$  are different
- (3) if  $y$  occurs in  $t$ ,  $y \neq t$ .

In the context of the event calculus (referring to water-flowing-into-bucket example), these statements say for example that  $height(x) = height(y) \rightarrow x = y$ , and that  $start$  is different from  $overflow$ .

**Definition 6.** *If  $P$  is a (general) program,  $comp(P)$  is the conjunction of  $IFF(P)$  and  $UNA$ .*

## REFERENCES

- [1] K. Doets. *From logic to logic programming*. The M.I.T. Press, Cambridge, MA, 1994.
- [2] M.P. Shanahan. *Solving the frame problem*. The M.I.T. Press, Cambridge MA, 1997.