A Typed Logic of Events

Mark Tarver,
mark@uk.ac.leeds.scs.,
UK 0113-335459,
Division of Artificial Intelligence,
University Of Leeds

Keywords: types, events.

Abstract

A typed logic E_0 is described in which situations are identified with the types of all the events that will bring them about. This identification lends itself to an elegant treatment of various systems from the blocks world of AI to the operations of the digital computer and the construction of geometrical objects.

Introduction

One shortcoming of formal logic is that it was originally developed to model the static world of the mathematician in which no events took place. Since its introduction into AI, formal logic has had to cope with the idea of *change* and *action*. Various formalisms have developed to meet this problem, including temporal logic and situation calculus. There is however, a powerful class of unexplored logics, which it is the purpose of this paper to introduce, which tackle the task of representing change, action and situation by means of a simple yet profound identification.

A situation is the type of all the events that will bring it about.

The logics of this class I call *typed event logics*. The formulae of a typed event logic are *typed expressions* (T-exprs) of the form e : S where e is a term denoting an event and S is a situation. Such a formula can be read as ”e inhabits the type S” or ”e brings it about that S”.

Events

Events include actions. An action is an event brought directly about by a living being. Events can have parts that are other events. We suppose that there is a class of *atomic events* that are events with no events as parts. *Molecular events* are composed out of atomic events. A molecular event is finite if it consists of a finite number of atomic events. We shall suppose that our event logics are finitistic; that is all events are assumed to be finite.

In *simultaneous* event systems, events may overlap or occur simultaneously. In *non-simultaneous* event systems, events cannot occur simultaneously. If we borrow the language of temporal reasoning and use $\leq$ to indicate the relation of simultaneity or beforeness then in simultaneous event systems, $\leq$ is a partial ordering whereas in non-simultaneous event systems it is a total ordering. We shall be concerned with non-simultaneous event systems in this paper. Every event occurs in its own time slot.

The operator O is a binary operation of composition on the set E of all events where O is closed in respect of E. (O a b) represents the event that is formed when a occurs and then b. O is associative so (O (O a b) c) is the same as (O a (O b c)). Consequently we drop the brackets and adopt a multiplicative notation where (a b c) is the same as (O (O a b) c) or (O a (O b c)). An event is thus an atomic event or a tuple of atomic events. Note that this notation suffices for non-simultaneous event systems since we suppose that there is a total

ordering whereby no two distinct events can occur simultaneously. In a simultaneous event system this notation would not suffice.

First Order Typed Event Logics

A typed event logic is *first order* iff for every T-expr e : S, S is a formula of first-order logic. By extension there are *second order* etc. typed event logics. E_0 is a first-order, finitistic, non-simultaneous typed event logic.

Syntax of E_0

An atomic event in E_0 consists of the application of an event operator (of which we shall see several examples) to a tuple of terms representing objects. We shall assume that the vocabulary of first-order logic is familiar to the reader and that expressions like term, logical constant etc are not unfamiliar.

1. e : S is an atomic wff of E_0 iff e is an event and S is a wff of quantifier-free first-order logic.

2. If A and B are wffs of E_0, so is A & B, A $\vee$ B, $\sim$A, A $\rightarrow$ B, $\forall$ x A and $\exists$ x A.

3. Nothing else is a wff of E_0.

Events are described by the following rules.

1. Where F is a symbol denoting an event operator and t_1,....t_n are terms of first-order logic, then (F t_1,....t_n) is an atomic event.

2. If e_x and e_y are events then so is the molecular event (e_x e_y).

Proof Theory of E_0

The proof theory of E_0 is given in figure 1. Negation is defined in terms of implication.

$\frac{e_1 : A,\ e_2 : B}{e_1 : A \,\&\, e_2 : B}$ $\qquad$ $\frac{e : A \,\&\, B}{e : A}$ $\qquad$ $\frac{e : A \,\&\, B}{e : B}$

$\frac{e_1 : A \vee e_2 : B \quad e_1 : A \rightarrow e : C \quad e_2 : B \rightarrow e : C}{e : C}$ $\qquad$ $\frac{e_1 : A}{e_1 : A \vee e_2 : B}$ $\qquad$ $\frac{e_2 : B}{e_1 : A \vee e_2 : B}$

$\frac{e_1 : A \quad e_1 : A \rightarrow e : B}{e : B}$ $\qquad$ $\frac{e_1 : A \vdash e_2 : B}{e_1 : A \rightarrow e_2 : B}$

$\frac{e : false}{e : A}$

where a is fresh
$\frac{(e : A)_{a/x}}{\forall\, x\ e : A}$ $\qquad$ where a is any term
$\frac{\forall\, x\ e : A}{(e : A)_{a/x}}$

where a is any term
$\frac{(e : A)_{a/x}}{\exists\, x\ e : A}$ $\qquad$ where a is fresh
$\frac{\exists\, x\ e : A}{(e : A)_{a/x}}$

Figure 1: The Axioms of E_0

e : ($\sim$ A) is defined as e : (A $\rightarrow$ false)
e : ($\sim$ ($\sim$ A)) reduces to e : A

We treat the following as equivalent:-

e : (A $\vee$ B) and e : A $\vee$ e : B
e : (A & B) and e : A & e : B
e : (A $\rightarrow$ B) and e : A $\rightarrow$ e : B

A paradox arises if we add the axiom

$\frac{(\sim e : A)}{e : A \rightarrow e : false}$

since then ($\sim$ e : A) entails e : $\sim$ A. (Proof: Assume $\sim$ e : A; then e : A $\rightarrow$ e : false

which is equivalent to e : (A $\rightarrow$ false) which is equivalent to e : ($\sim$ A).) But to say

Jones brought it about that the meeting was not on time.

and

It is not the case that Jones brought it about that the meeting was on time.

assert different things. The second assertion is weaker than the first which attributes responsibility to Jones which the second assertion does not. So intuitively the entailment does not hold between ($\sim$ e : A) and e : $\sim$ A. A disavowal of causal relations between an event and a situation is a weak claim that does not allow us to assert causal relations between that event and another situation.

The Empty Action

A useful action (oddly enough) is the empty action 0. The empty action is the action of doing nothing. We assume the following

Empty Action Axiom

(0 e) : A iff (e 0) : A iff e : A

Note that this axiom would not seem to be true of all systems. If we pull a pin off a grenade, do nothing for five seconds, and then try to throw it, the result will not be the same as if we pulled the pin off the grenade, threw it and waited for five seconds. In other systems like the blocks world this axiom does hold. If the computer does nothing; nothing changes. Thus the empty action axiom holds only of systems where there is one agent and the actions of that agent are the only source of change. Such systems I call solopsistic closed systems. Adding the Empty Action Axiom to E_0 makes it applicable only to solopsistic closed systems.

By assuming a solopsistic closed system we can defined truth and falsity. Something is true or actual if 0 inhabits it; in other words something is already true if nothing is needed to bring it about. Something is actualisable if an event inhabits it and some situation is unactualisable if nothing inhabits it.

Identity in E_0

In the Tractatus Logico-Philosophicus, Wittgenstein describes the world as consisting of objects which are indestructable which change only in respect of the relations between them. In a logically perfect language, there would be a one and only symbol for each object and no two objects would have the same symbol. Wittgenstein concluded that in such a language there would be no need of identity:-

5.5303 Roughly speaking, to say of two things that they are identical is nonsense, and to say of one thing that it is identical with itself is to say nothing at all.

So Wittgenstein concludes:-

5.533 The identity-sign, therefore, is not an essential constituent of conceptual notation.

E_0 is supposed to be a language of the sort Wittgenstein envisaged and the objects which are subject to E_0 are objects which are neither created nor destroyed by events. Events can only change relations and properties. However, the corollary that Wittgenstein imposed, that = must then become redundant since a = a is trivial and a = b false, is not strictly true. There are occasions when it is necessary to state that an axiom must apply to objects that are distinct and in such cases to formally state the axiom we need a concept of identity. However, Wittgenstein's observation that in a logically perfect language, evaluating statements of identity is a trivial exercise is still true. This fact will become apparent when we come to formalize the blocks world.

Immediate from casting E_0 as a Wittgensteinian perfect language are two axioms

$\Delta \vdash e : x = x$

$\Delta \vdash e : (\sim (x = y))$ where x and y are distinct names

Every event brings it about that any object is identical with itself and any event brings it about that any two distinct objects are not the same.

Interlude: the Strong Conditional

Philosophical logicians distinguish between the strong conditional of ordinary speech and the material conditional of logic. A strong conditional asserts a causal or conceptual connection between two situations whereas the material conditional asserts only a truth-functional relation. Philosophical logicians also believe that the strong conditional entails the material conditional, but not vice versa.

Interestingly, the strong conditional is definable in typed event logic. Let $\Rightarrow$ denote the strong conditional.

Axiom of the Strong Conditional

$e_1 : (A \Rightarrow B)$ iff $(e_1\ e_2) : A \vdash (e_1\ e_2) : B$ where e_2 is any term

The idea behind this definition is essentially simple. If an event or action brings about a causal connection between two situations then any action or event carried out immediately after that action to bring about the antecedent will also be an action that brings about the consequent. Thus if I connect a plunger to detonate dynamite then I establish a causal connection between the plunger being depressed and the dynamite exploding. The existence of this causal connection is shown by the fact that if the action of connecting the plunger is immediately followed by any event that depresses it, the same molecular event of connecting the dynamite and depressing the plunger also brings about the explosion of the dynamite.

Immediate from this definition follows the consequence that the strong conditional entails the weaker material conditional. The proof makes essential use of the empty action

axiom.

Theorem: If e_1 : (A $\Rightarrow$ B) then e_1 : (A $\rightarrow$ B)

Assume e_1 : (A $\Rightarrow$ B); then (e_1 e_2) : A $\vdash$ (e_1 e_2) : B for all e_2. In which case (e_1 0) : A $\vdash$ (e_1 0) : B. By the empty action axiom this reduces to e_1 : A $\vdash$ e_1 : B which by E_0 suffices for e_1 : (A $\rightarrow$ B).

We shall not incorporate the definition in E_0, though it casts an interesting light on the conceptual relations between cause, and the strong and weak conditionals.

The Blocks World in E_0

The language of E_0 suffices for the exploration of a number of solopsistic closed systems. A classic instance of this is the blocks world. In this section we show how to axiomatise the operations of a robot arm manipulating the blocks and how to mimic the operations of STRIPS - the archetypal planning system for the blocks world. The operations of UNSTACKing and STACKing within STRIPS are axioms framed in the language of E_0.

stack

$\forall$ e,A,B (e : (CLEAR A) & e : (CLEAR B)) $\rightarrow$ (e (STACK A B)) : (ON A B)

The event (STACK a b) is treated here as an atomic event. The axiom states that performing e and then stacking A on B brings it about that A is on B provided that e brings it about that A and B are clear. One point to note here is that E_0 makes provision for the use of dependent type expressions that are essential in describing the relations between events and situations. A dependent type expression is one in which the variables used in its mention straddle the dividing : linking the inhabitant to the type. In the context of E_0, it is often impossible to describe a causal series of events without describing some of the same objects that whose relations constitute the caused situation.

unstack
$\forall$ e,A,B (e : (ON A B) & e : (CLEAR A)) $\rightarrow$ (e (UNSTACK A B)) : (CLEAR B)

Performing e and unstacking A from B brings it about that B is clear provided that e brings it about that A is on B and A is clear.

To these two axioms we add 4 frame axioms:-

unstack-frame-clear
$\forall$ e,A,B,C e : (CLEAR A) $\rightarrow$ (e (UNSTACK B C)) : (CLEAR A)

Unstacking something keeps everything clear that was clear.

stack-frame-clear
$\forall$ e,A,B,C $\exists$ x (e : (CLEAR A) & x : ($\sim$ (C = A)) $\rightarrow$ (e (STACK B C)) : (CLEAR A)

Stacking B on C keeps a clear as long as A and B are not the same and A was clear before.

unstack-frame-on
$\forall$ e,A,B,C,D $\exists$ x (e : (ON C D) & x x : (($\sim$ (A = C)) $\vee$ ($\sim$ (B = D)))) $\rightarrow$ (e (UNSTACK A B)) : (ON C D)

Unstacking A from B keeps C on D as long as either A is not the same as C or B is not the same as as D and C was on D before.

stack-frame-on
$\forall$ e,A,B,C,D e : (ON A B) $\rightarrow$ (e (STACK C D)) : (ON A B)

Finally, stacking C on D results in A being on B if A was on B before.

A classic problem in the blocks world is to find a plan of action that will transform the situation in figure 2.

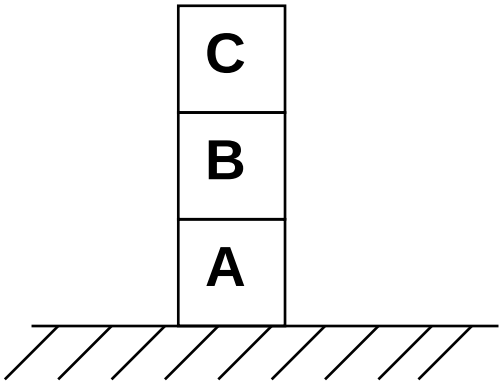

Figure 2: The Blocks World - First Configuration

to one like this (figure 3):-

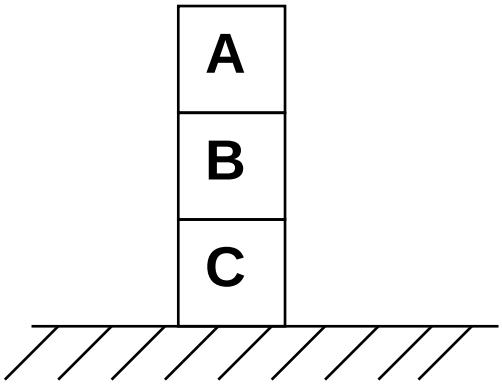

Figure 3: The Blocks World - Second Configuration

The initial state of the blocks world can be stated by asserting the inhabitation status of the empty action. Thus to state that C is clear and C is on B is on A we write:

0 : (clear C), 0 : (ON C B), 0 : (ON B A).

We are required to find some series of action that stack these blocks in reverse order so that A is on B is on C. Formally:-

0 : (ON B A), 0 : (ON C B), 0 : (clear C) $\vdash \exists$ x x : ((ON A B) & (ON B C) & (CLEAR C))

It is left as an exercise for the reader to prove that a value for x for which the above is a theorem is:-

((UNSTACK C B) (UNSTACK B A) (STACK B C) (STACK A B))

Representing a Shift Register in $\mathbf{E}_0$

Computer hardware is obviously a fruitful application for typed event logic since proofs involving hardware involve representing the states of electronic devices and the way in which the events that take place within the hardware affect the states of these devices. Our exposition takes as its illustration, the representation of the *shift register* in figure 3. This

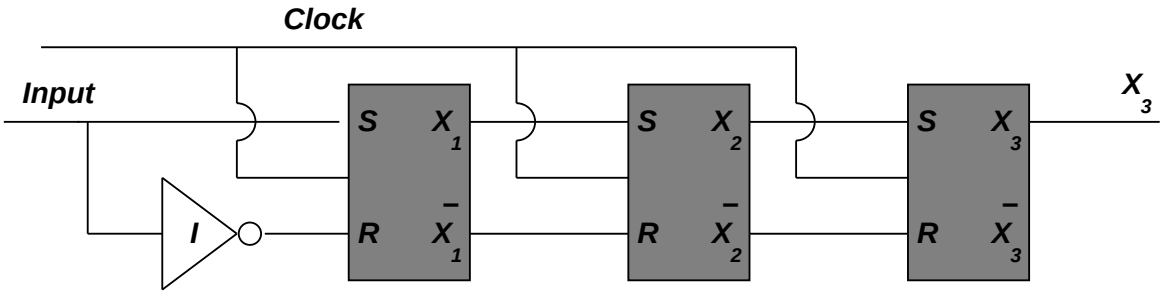

Figure 4: A Shift Register

circuit accepts a signal from some input source and then shifts this signal along the chain of flip-flops, moving it one flip-flop each time a positive-going clock cycle occurs. The

signals are regulated by a clock so that in each time period one and only one signal occurs. The signal has one of two forms, high or low, conventionally represented as 1 and 0 but to avoid confusion with the empty action we shall use *high* and *low*. Conventionally, the X-line of a flip-flop is thought of as the state of the flip-flop, so I shall say that a flip-flop is set to a value meaning that the the X-line of that flip-flop is set to that value.

When a high signal occurs the X line of A is set to 1 since the S-line of A is high and the R-line low (due to the inverter in the diagram) and by the rules governing the behavior of flip-flops this sets the X line to 1. Conversely when a low signal occurs the X line of A is set to 0. This is summarised in the first two axioms.

$\forall$ e (e high) : (X-line A 1)

that is, a high signal sets the X line of A to 1 no matter what the preceding event.

$\forall$ e (e low) : (X-line A 0)

and a low signal sets the X line of A to 0 no matter what the preceding event.

The B flip-flop is set to the value of the A flip-flop at the preceding clock cycle and the value of the C flip-flop is set to the value of the B flip-flop at the preceding clock cycle.

$\forall$ e_1, e_2, x (e_1 : (X-line A x) $\rightarrow$ (e_1 e_2) : (X-line B x))

That is, a molecular event composed of two events e_1 and e_2 sets the B flip-flop to a state x if e_1 sets the A flip-flop to a state x. An analogous axiom applies to flip-flop C.

$\forall$ e_1, e_2, x (e_1 : (X-line B x) $\rightarrow$ (e_1 e_2) : (X-line C x))

From these axioms we are required to show an event that sets all the flip-flops to 1. That is:-

Prove $\exists$ x x : ((X-line A 1) & (X-line B 1) & (X-line C 1)).

A value for x for which this is a theorem is x = (high high high); i.e. three successive high or 1 signals down the I wire will set all flip-flops to 1. The precise proof is left to the reader.

Executing Graphics in $\mathbf{E}_0$

One way of describing a graphical object is to give the means whereby it can be constructed. In terms of the situations-as-types principle, this amounts to specifying a molecular event that inhabits the situation whereby the graphical object is brought into being. The axiom systems needed to describe 2-D graphical objects can be given as theories in the language of E_0. These axioms often have a recursive structure to them and repay study for that reason.

We assume the existence of an infinite domain given by an infinite set of Cartesian co-ordinates. There is but one primitive operation in respect of this domain and that is the operation of placing a point P on a coordinate $< x, y >$ given by the operation (PLACE

$< x, y >$). We wish to define the inhabitation conditions for a line to exist from coordinates $< x_1, y_1 >$ to $< x_2, y_2 >$. written (LINE $< x_1, y_1 > < x_2, y_2 >$). The simplest case is where x_1 = x_2 and y_1 = y_2.

line-base
∀x,y (PLACE $< x, y >$) : (LINE $< x, y > < x, y >$)

Suppose $< x_1, y_1 > \neq < x_2, y_2 >$. In that case we can draw a line from $< x_1, y_1 >$ to $< x_2, y_2 >$ by calculating where the midpoint M of the line must occur and placing a point on M. The problem of drawing a line of the desired kind is reduced to that finding an inhabitant of the situations (LINE $< x_1, y_1 >$ M) and (LINE M $< x_2, y_2 >$). By recursing on an axiom that encapsulates this procedure we may calculate an inhabitant for any given line. This inhabitant, when interpreted as (e.g.) a bitmap on a computer, can be used to draw lines on a screen. The process is shown diagramatically in figure 4.

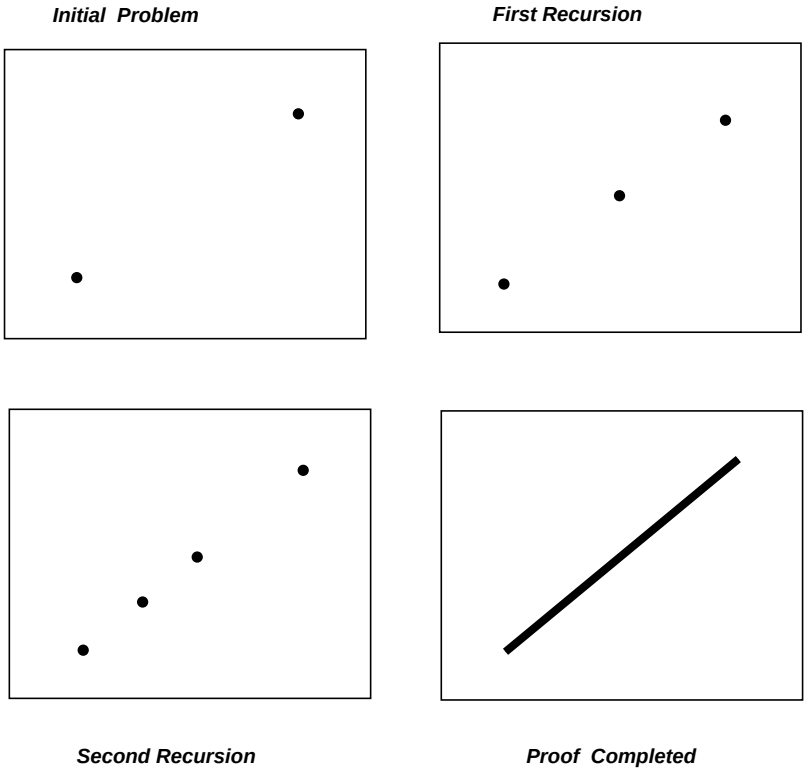

Figure 5: Drawing a Line

To calculate the midpoint we use the equation M = $< (r(+x_1x_2)/2), (r(+y_1y_2)/2) >$ where r rounds the result to the nearest integer.

line-recursive
where M = $< (r(+x_1x_2)/2), (r(+y_1y_2)/2) >$
and $< x_1, y_1 > \neq < x_2, y_2 >$

∀ e_1,e_2,x_1, y_1, x_2, y_2
(e_1 : (LINE $< x_1, y_1 >$ M) & e_2 : (LINE M $< x_2, y_2 >$))

$\rightarrow$ (e_1 e_2) : (LINE $< x_1, y_1 > < x_2, y_2 >$)

The axioms for drawing more complex geometrical shapes can be calculated by using standard results from coordinate geometry together with the axiomatic definitions of some standard shapes. For example, the axiom for SQUARE describes how to draw a SQUARE of sides length l with bottom left-hand corner at coordinates $< x, y >$.

square
$\forall$ e_1,e_2,e_3,e_4,x, y, l
(e_1 : (LINE $< x, y >$, $< x + l, y >$)
& e_2 : (LINE $< x + l, y >$, $< x + l, y + l >$)
& e_3 : (LINE $< x, y >$, $< x, y + l >$)
& e_4 : (LINE $< x, y + l >$, $< x + l, y + l >$))
$\rightarrow$ (e_1 e_2 e_3 e_4) : (SQUARE $< x, y > l$)

Typed Event Logic and Situation Calculus

Typed event logic has common ground with situation calculus (McCarthy (1963), Kowalski (1979), Elkan (1992)). In situation calculus, change is viewed as a succession of situations linked by actions. To explain change, we declare *effect axioms* that state how situations change when actions are performed. Predicates are given extra arguments places held by situations so that (IN John Kansas S_0) states that John is in Kansas in situation S_0.

E_0 is probably one the weaker systems of typed event logic, but it has much the same expressive capacities as situation calculus. But typed event logic is in many ways rather more elegant than situation calculus, the necessity for extra argument places in predicates is not required in typed event logic. Moreover the clean seperation between the notation for events and that for situations not only improves readability - it also suggests ways in which typed event logic can be developed by investigating the use of modal and temporal logics on the right-hand side of the :. An exciting possibility is to look into non-monotonic typed event logics as an alternative to using numerous frame axioms.

Typed Event Logic and Temporal Logic

Temporal logic as practised by computer scientists is heavily dependent on earlier work carried out by philosophers into what was then called tense logic (see Prior (1967)). The main contribution of computer scientists has been to enrich the notation of ense logic to include extra operators and to adapt the ideas of the philosophers towards the representation of systems of interest to computer science - such as software and hardware. In a tense logic like system K, four extra operators G, F, P and H are added to first-order logic meaning 'It will always be the case that', 'It will sometimes be the case that' 'It was always the case that' and 'It was sometimes the case that'. Tense logic and its temporal relatives like linear-time temporal logic are founded on the idea that there is a privileged instant of time - the present - against which the truth or falsity of tensed propositions is to be assessed.

E_0 is somewhat indifferent to the idea of present. It is natural perhaps to read '0 : P' as

meaning 'P is currently true' as well as 'Doing nothing makes P true'. However even this is doubtful, for '(0 e) : P' reduces to 'e : P' which, through its lack of 0 gives no clue to its relation to the present. E_0 is thus essentially tenseless. We may nominate a particular event, say e_0, and say "This is our present" in the same way that we can nominate an inertial frame in Einsteinian physics and say "This is at rest"; but the choice is observer dependent.

Even given such a convention the analogy to temporal logic tends to peter out. For instance, the statement F(P) (meaning "It will sometimes be the case that P") has no analogue in E_0. The reason is that E_0 is not a language designed for prediction but a language concerned with saying *how things can be brought about*. E_0 tells us that certain actions will issue in certain results rather than telling us that certain things are inevitable.

However there is a natural interpretation to typed event logic that allows it to express easily and naturally, the idea of something being true in the future. To say that P will, at some time in the future, is to say that *no matter what we do, P will come about*. In terms of typed event logic this means that all event sequences, if continued from our nominated present, e0, will eventually result in a molcular event inhabiting P. To say P will always be the case is to say that all event sequences continued from e0 inhabit P; the cases for "was always true" and "was once true" are fairly obvious. Actually E_0, for reasons of syntax, cannot express these propositions although a extension of syntax and proof theory to E_0 would create a logic that was able to do so.

A logic which is closer in its philosophy to typed event logic is branching temporal logic. In the semantics of linear temporal logic the relation R of temporal succession is a functional one. Every instant has at most one and only one successor. In the semantics of branching temporal logic an instant has *at least* one successor and possibly more, so that instead of a linear series of moments of time, branching temporal logic presents us with a tree. The philosophy of branching temporal logic is essentially non-deterministic. To say that P must occur (from the perspective of a nominated present t0) is to say that from t0 all paths on the tree lead to P when t0 is taken as the starting point and this is written as $\forall$(F p) where the $\forall$ indicates that this statement is true of all paths leading from t0. To say that P could occur is to say that there is at least one path from t0 at which (F p) is true or $\exists$(F p).

Typed event logic is similar to branching temporal logic in seeing the future as consisting of a series of alternatives. However where it differs is that typed event logic offers an account of how these different possibilities are actualised. Rather than presenting a passive path of instants of time at which propositions choose to be true or not, typed event logic can show us the conditions under which these different possibilities can be brought about. Moreover while instants of time are essentially objects without structure, in typed event logic atomic events can be objects with structure, involving non-event objects like blocks (as the blocks world example showed). Hence typed event logic is conceptually rather richer than even branching temporal logic.

Conclusion

Typed event logic arises from the exploitation of one profound and simple idea: that a situation can be identified with the type of all the events that will bring it about. This principle engenders an indefinite number of formal systems that I have called typed event

logics. One of the simplest members of this family, E_0, has a representational capacity to rival that of branching temporal logic. The situation-as-types principle is naturally to be compared with the propositions-as-types principle in which a proposition is identified with the type of all its proofs. The propositions-as-types principle faces carries us deep into the Platonic regions of the lambda calculus and constructive proof. The situation-as-types principle carries the thinker into the physical world and to a deeper understanding of the relations between actions and situations.

References

Elkan C. Reasoning about Action in First-Order Logic in *Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence,* Vancouver, British Columbia. (1992).

Kowalski R. Logic for Problem-Solving, North-Holland (1979).

McCarthy J. Situations, Actions and Causal Laws in Semantic Information Processing, ed. Minsky, MIT Press, (1968).

Prior A.N. Past, Present and Future, OUP (1967).

Thompson S. Type Theory and functional Programming, Addison-Wesley (1992).

Whitehead A.N. The Concept of Nature, CUP, (1919).

Wittgenstein L. Tractatus Logico-Philosophicus, Routledge & Kegan Paul, Second Edition, (1971).