KRISTER SEGERBERG

Applying Modal Logic*

Abstract. The main purpose of the paper is to introduce philosophers and philosophical logicians to dynamic logic, a subject which promises to be of interest also to philosophy. A new completeness result involving both "after" – and "during" – operators is announced.

1. Introduction

Modal logic, like other kinds of philosophical logic, comes in two varieties, pure and applied. It is not quite the same distinction as the corresponding one in mathematics. Pure modal logic is the formal study of certain abstract structures. Applied modal logic is the study, not necessarily formal, of certain structures, not necessarily abstract, and with a certain purpose: to relate to something else – to some part of (some conception of) reality. Therefore the terminology is a bit misleading. For one thing, applied modal logic need not be pure modal logic applied to something.

Throughout the history of modal logic, the pure and applied varieties have been intertwined. Sometimes the dividing line is difficult to discern. But that it is there may be seen from the different kinds of criticism there are. Critics will not dispute that there is a definite body of established results in pure modal logic, even though they may find those results trivial or unexciting. In other words, the existence of this discipline is not in question. The situation is different in applied modal logic, for that is the discipline critics have in mind when they say, as they sometimes do, that there is no such thing as modal logic, or even that there can be no such thing. A disturbing number of competent philosophers remain unconvinced that applied modal logic makes sense, let alone has anything to offer philosophy.

One may ask why this is so. It seems to this author that there are at least two important answers. One: key concepts of Kripke semantics – which is the stuff that today's applied modal logic is made of – are evidently too obscure or too badly explained to be generally intelligible. Two: the usual analyses in terms of modal logic of many philosophically interesting concepts fail to bring out their logical form. Events, processes,

* The work on this paper was supported in part by an Academy of Finland fellowship *för längre hunna vetenskapsidkare* during the former half of 1979. The paper itself was read as an invited address at the Conference on Practical and Philosophical Motivations of Non-classical Logics held at Toruń, August 15-19, 1979.

and actions are examples of concepts for whose analysis modal logic has proved rather unilluminating so far.

These are sweeping verdicts! But even those who take a less pessimistic view than the author will probably agree that any development that offers the slightest promise to improve the situation deserves the attention of all friends of modal logic. One recent effort emanating from the Massachusetts Institute of Technology that surely falls under this heading is dynamic logic, a creation by Vaughan Pratt and his collaborators. Their ideas provide a striking interpretation of possible-worlds semantics which, in addition to being precise and intelligible, is also claimed by its creators to be useful. Even more, dynamic logic could provide a new way of analysing action!

Dynamic logic was born in a computer science department, and in available accounts it appears as the offspring of computer science that it is. This paper is perhaps the first attempt to present it to a philosophical audience. The plan of the paper is as follows. Section 2 is devoted to giving an illustrative example. This example, which falls slightly outside the intended range of interpretations of dynamic logic, is meant to be helpful for the continued intuitive discussion in Section 3 and 4. Section 5 contains a general definition of propositional languages of dynamic logic. In Section 6 the fundamentals of formal semantics are developed. Some possible implications for philosophy are sketched in Section 7.

Much of the material presented here is already known. The author's debt to Professor Pratt is great, both to his papers and to conversations with him. Among other things Pratt should be credited with the idea that programs are intimately connected with actions. Thus the author's contribution is to have recast Pratt's ideas in a form that is more recognizably modal logic as this discipline is known traditionally; the modelling in Section 6 should be compared with that of [4]. The two theorems in Section 6 are announced in print for the first time; they were presented to the Fifth Scandinavian Logic Symposium in Aalborg, Denmark, on January 17, 1979. The author is also responsible for the material of Section 7: that one might try to analyse human action with tools devised to analyse machine action has not been suggested by Pratt, and the idea may indeed prove unrealistic.

It should be mentioned that Pratt's dynamic logic is not the only theory of its kind. There are predecessors, like the theories of Floyd and Hoare, and there are present day alternatives, like Salwicki's theory. For Pratt's own work, see [2, 3, 4]. A survey of the field is given in Pratt [5]. A previous attempt by the author to steer dynamic logic into the mainstream of traditional modal logic was made in [7].

2. A simple-minded example

My family's washing machine is a black box-like object in the basement. It is operated as follows. You load the dirty laundry into the barrel.

You put detergent in the holder intended for this purpose. You select a program by inserting a program key, made of pink plastic, into a certain slot. You check that the water and the electric power are on. You press the start button. Then a lamp lights up, and the machine takes over. If there is no malfunction or interference, the machine will have carried out the chosen program in less than half an hour. At the end of a successful wash the lamp goes out, indicating that the machine has stopped. If half an hour or so after the start button was pressed everything is quiet but the lamp is still on, you suspect that something is wrong — for some reason there is a failure. As minutes pass with no further action and without the lamp going out, your suspicion rapidly grows into conviction: very soon you "know" that something is wrong.

The programs on the program keys have short mnemonic names or labels, like WHITE; WHITE, VERY DIRTY; COLORED; COLORED, VERY DIRTY; COTTON; NYLON; DELICATE; DRIP-DRY; SYNTHETIC DRAPES; etc. The effect of each program is outlined briefly in the instruction book. For example, this is the information given about WHITE, VERY DIRTY:

Fills cold water to high level.
Heats to 50°-C during tumbling.
Fills cold water to low level.
Heats to 95°-C during tumbling.
Fills cold water.
4 rinse cycles with short intermittent spin cycles.
Spin cycle 3 minutes 45 seconds.

Exactly how the machine functions is not explained to the customer. There exists of course an extensive theory for the hardware of the machine, a theory which may be said to form a subtheory of physics (mainly mechanics and the theory of electricity). The local distributor prospers due to the fact that this theory is known to his service man but not to customers like myself. For all I know, this theory may be interesting in its own right, but as a customer I take no interest in it. What directly interests me is that my laundry gets done properly. Meaning: that the laundry gets washed without getting damaged. The reasoning I employ when I set out to use the machine on my laundry will not be in terms of electricity etc. but rather in terms resembling the idiom of the instruction book.

Suppose, for example, that I have a load of very dirty terylene shirts. A quick check shows that none of the programs on the program keys is designed expressly for this category. So I will have to think about which program, if any, would be suitable for this load. It is easy to think of necessary conditions on such a program a. Above all there is this:

After a, the load is clean.

To effect this, it may be that the following condition would be sufficient:

During some of α, the temperature is at least 40°C.

On the other hand we are also concerned that the shirts be handled with care:

After α, the load is not spoiled.

For this reason we require, say,

During all of α, the temperature is less than 50°C.

There may be various other conditions to consider (on torque, for example). If I can find at least one program satisfying all these conditions, fine: then I can use that one. If not, then I shall have to wash the shirts by hand.

This is no doubt a simplified picture of what goes on in a launderer's mind. But hopefully it is not too far from the truth either. Now, this kind of reasoning can be described in a slightly more abstract way as follows. First I enumerate a number of conditions $\mathbf{A}_0, \ldots, \mathbf{A}_{m-1}$, all of which are to hold at all time during the wash, and a number of conditions $\mathbf{B}_0, \ldots, \mathbf{B}_{n-1}$, each of which is to hold at some time during the wash. Then I go through the finite list of available programs to see if there is one, α say, such that every member of the set

$$\Gamma = \{\ulcorner\text{during all of } \alpha, \mathbf{A}_i\urcorner : i < m\}$$

is true; every member of the set

$$\Theta = \{\ulcorner\text{during some of } \alpha, \mathbf{B}_i\urcorner : i < n\}$$

is true; and, furthermore,

$$\Gamma, \Theta \vdash \ulcorner\text{after } \alpha, \text{ the load is clean}\urcorner,$$
$$\Gamma, \Theta \vdash \ulcorner\text{after } \alpha, \text{ the load is not spoiled}\urcorner,$$

where the turnstyle refers to the theory I have developed on the basis of whatever knowledge I have of fabrics, detergents and many other things as well as my previous experience with the washing machine. Any such program α can be used to wash my shirts. On the other hand, if there is no such α, then the shirts will either be ruined or not properly washed if I foolishly insist on using the machine anyway.

Thus, even though I don't know (and really don't care to know) anything about what goes on inside the machine, I can — and do — reason about aspects or consequences of these goings-on.

Now some final remarks on the deterministic/nondeterministic distinction. The washing machine has been presented as deterministic: given the initial conditions (the loading etc.) and the program, the action of the machine is determined. At least, this is what the instruction book

gives you to understand. Actually, things sometimes go awry. That is to say, my presentation of the machine does not give the whole picture. A presentation closer to the truth may be obtained by representing the machine as nondeterministic in the following way.

During the five years we have had the machine, every breakdown I have encountered has fallen under one of the following headings: a fuse blows; the barrel jams, the pump breaks; and (most expensively) something happens to the computer unit. For some reason, each of these breakdowns occurs only in connection with certain programs. Thus a fuse is known to blow only if WHITE, VERY DIRTY or COLORED, VERY DIRTY is run. Fortunately, it doesn't happen every time one of those programs is run. In fact, it happens rather infrequently (not that I have been able to figure out a frequency). But so far, every time a fuse has blown, it has been while one of these programs was being run. Furthermore, none of the other breakdowns has ever occurred in connection with either one.

It appears, then, that it would be more informative if the program labels WHITE, VERY DIRTY and COLORED, VERY DIRTY would be replaced by new program labels like WHITE, VERY DIRTY, OR FUSE, BLOW A and COLORED, VERY DIRTY, OR FUSE, BLOW A, respectively. To be sure, this is not a change that the manufacturer is likely to initiate. Yet to a customer such a change would make sense. When he uses the program labeled WHITE, VERY DIRTY he may think of himself as issuing a command to his servant, the machine, "Run the very-dirty-white routine!". But at least in the light of the experience reported here it would be more realistic if he thought of himself as telling the machine, "Run the very-dirty--white routine, or blow a fuse!". For this is what the machine will do if programmed with the WHITE, VERY DIRTY program, no matter what the label of the program.

Notice that nothing has been said about probabilities here. However, under some additional assumptions the washing machine could be given a probabilistic representation as well. Suppose that, contrary to what was said above, I have been able to figure out the frequency with which a fuse blows during a run of the WHITE, VERY DIRTY program. Suppose that the frequency in question is p, where thus p is a real number between 0 and 1. Then it would make sense to replace the manufacturer's label with another one: $(1 - p)$ (WHITE, VERY DIRTY) OR p (FUSE, BLOW A). This label brings to mind the intuitive command, "With probability $(1 - p)$ run the very-dirty-white routine without breaking a fuse, or, if you don't run the very-dirty-white routine, blow a fuse!". Under the present assumptions, this is what the machine will do if programmed with the WHITE, VERY DIRTY program, no matter how the program is labelled.

However, probabilistic machines are a different topic, and from now on we shall say no more about them.

3. Automata and programs

There are automata of many different kinds, from sophisticated computing machinery to washing machines to simple gadgets ordinarily not thought of as automata at all (like ball point pens and electric switches).

As logicians we are interested only in the most general properties of automata. Let us therefore think of them as black boxes (BB's) in the usual way. The main assumptions we make about them are the following. (1) At every instant of time, the BB is in some welldefined state (= total state). (2) There is a definite set of programs for the BB. (3) The BB can be programmed, the particular program, a given initial state (and perhaps a randomizing device) determining the action of BB. (4) If a program is chosen and the BB is started, the BB acts by proceeding from state to state in a discrete fashion, the transitions taking no time at all. (5) When the BB is through with a program, it stops. (6) One can always tell whether the BB has already stopped; but if the BB has not yet stopped, there may be no way of telling whether it ever will stop. (7) It is possible for a BB to fail; this occurs when the BB remains in a certain state with no further action forthcoming, and yet has not stopped. (8) The BB is perfect in the sense that it does not break down — there is no malfunction, nor any interference from the outside. (The last assumption is sometimes negotiable.)

Think of a particular BB as given. It may not be necessary to postulate that the set of programs associated with it be "wellfounded" in the sense of containing a small subset of "primitive" or "basic" programs in terms of which all others are definable. But here we do: we shall assume that there are some primitive programs $\pi_0, \pi_1, \ldots, \pi_m (, \ldots)$ finitely or infinitely many. We also assume that the set of programs is closed under what is called the three regular operations, the binary $+$ and $\cdot$ (actually the dot is often omitted in formulas!) and the unary *. They may be given a heuristic account as follows.

It is helpful to have some intuitive picture or pictures of programs. Two ways of understanding them are as *actions* and as *imperatives*. On the former understanding one thinks of a program α as an action (action type rather than particular action): "the action α" or "the action consisting of doing α". On the latter understanding one thinks of α as an imperative such as "Do α!" or "Carry out the α-routine!"

These readings allow us to explain $\alpha+\beta$, $\alpha\beta$, and α^*, given that α and β are programs. If α and β are thought of as "the action α" and "the action β", respectively, these three are thought of as

"the action consisting of doing α or β",
"the action consisting of doing first α and then β",
"the action consisting of doing α some number of times".

If α and β are thought of as "Carry out the α-routine!" and "Carry out the β-routine!", respectively, the three instead become,

"Carry out the α-routine or the β-routine!",
"Carry out first the α-routine and then the β-routine!",
"Carry out the α-routine some number of time!".

Yet another way of visualizing a program is as a *list of instructions* of this format:

BEGIN
φ_0
.
.
.
φ_{k-1}
STOP

where $\varphi_0, \ldots, \varphi_{k-1}$ are themselves programs. (It is a conception of this kind that often makes authors of textbooks refer to the advice given to the White Rabbit in *Alice in Wonderland*: '"Begin at the beginning", the King said, very gravely, "and go on till you come to the end: then stop."') Viewing programs this way may be bewildering when it comes to composite programs of type $\alpha+\beta$ or α^*, but sometimes it can be helpful. For example, the list interpretation elucidates the distinction between the *impossible program*, which cannot be executed, and the *identity program*, which executes nothing. For the impossible program may be thought of as the list

BEGIN
Verify that $0 = 1$!
STOP

whereas the identity program may be thought of as the list

BEGIN
STOP

Thus the difference is that if you embark on the impossible program you never get to STOP, while if you embark on the identity program you get there instantly. The impossible program is denoted by **0** and the identity program by **1**. They may or may not be available as primitive programs; if they are, they may or may not be among the π_i's.

All this is of course rather loosely speaking. But even here it is clear that the question of determinism enters. Note that the readings of $\alpha+\beta$ and α^* become unintelligible in connection with deterministic BB's. For in the case of $+$ it is left open which alternative, α or β, is to be chosen.

Similarly in the case of * the precise number of iterations of α, 0 or 1 or ..., is left open. The idea here is that it is the BB itself that decides on these questions (with the help of the randomizing device postulated under (3) above).

4. Languages and models for automata

Continue to think of a particular BB as given. We shall now try to discuss two notions, the notion of *modal logical language suitable for* the BB in question, and the notion of *model based on* the BB. Our discussion is still informal, and we do not rigorously define the notions just mentioned. Formal definitions of related formal concepts are given later — see Section 5 for languages and Section 6 for models.

First the languages. There will always be some primitive or basic propositions; perhaps $\top$ or $\bot$ is one. Exactly what they are will depend on the BB and on what we are interested in — we will not go into that question, but the discussion in Section 2 should be suggestive. Then there are the more complex propositions that can be obtained by the use of propositional operators. The Boolean ones are wellknown: if **A** and **B** are propositions, then so are $\neg\mathbf{A}$, $\mathbf{A}\wedge\mathbf{B}$, $\mathbf{A}\vee\mathbf{B}$, $\mathbf{A}\rightarrow\mathbf{B}$, $\mathbf{A}\leftrightarrow\mathbf{B}$, and perhaps still others. But we also need propositional operators of a novel kind, as indicated by the washing machine example. The following ones seem to be among the simplest and most natural ones (notation to the left, import to the right):

$[\alpha]$ after every computation according to α,
$\langle\alpha\rangle$ after some computation according to α,
$[\![\alpha]\!]$ always during every computation according to α,
$\langle\!\langle\alpha\rangle\!\rangle$ sometimes during some computation according to α,
$\langle\!\langle\alpha]\!]$ always during some computation according to α,
$[\![\alpha\rangle\!\rangle$ sometimes during every computation according to α.

(Actually, the readings as given are ambiguous. The ambiguity will be removed below.)

As presented here, programs and modal logical languages can be described independently of one another. But this is not necessarily so; if so-called test programs are allowed, it is no longer the case. If **A** is a proposition, it makes sense to ask whether **A** obtains. So **A**?, codifying this question, can be introduced as a new program. On the action interpretation, think of **A**? as "the action consisting of verifying that **A**" (rather than the misleading formulation "the action consisting in testing whether **A**"). On the imperative interpretation, think of **A**? as "Verify that **A**!". Of course, if **A** is not the case, then the program **A**? cannot be carried out.

Let us now see what we can do for semantics. By our assumptions, there is a welldefined set of all the states of our BB. Such a state should

be thought of as a total state: a momentary state-of-affairs inside the BB or, if you prefer, a cross-section of the BB as an object in space-time: the contents of its input, output, memories, ... (The program is not considered part of the state.)

Now suppose that the BB is in some state x. Let the BB be programmed with a program α. When started the BB will go into action, proceeding from state to state in a discrete fashion. In principle at least it is possible to plot its journey through state space. Let us call any sequence of states a *path*. Thus there is an obvious sense in which the BB, when started, will produce a path, viz., the sequence of states in the order the BB passes them through. Under certain conditions, such a path will be what we shall call a *computation according to* α or *α-computation*. There are four cases to consider.

Case 1. The BB ceases to act after finitely many steps, and the path produced is an α-computation. This computation, then, is of type $\langle z_0, \ldots, z_n \rangle$, with $z_0 = x$ and $n \geqslant 0$. The length of the computation is n.

Case 2. The BB ceases to act after finitely many steps, but the path does not qualify as an α-computation. (This path is called a *failure of α at x* or an *α-failure*.)

Case 3. The action of the BB goes on for ever, and the path produced is an α-computation. This computation, then, is of type $\langle z_0, \ldots, z_n, \ldots \rangle$, with $z_0 = x$ but without a last element. The length of the computation is ω.

Case 4. The action of the BB goes on for ever, but the path produced does not qualify as an α-computation.

Case 4 may not be of any practical interest, but it has been included as a logical possibility. Of the other three, case 2 is particularly interesting. Two instances of it are worth mentioning here. One is obtained by letting α be the impossible program. In fact, the difference between the impossible program and the identity program now comes out clearly. The only computations according to the identity program are those of length 0, for nothing "happens" during such a computation. On the other hand, there can be no computation at all according to the impossible program — no computation in the world could verify that $0 = 1$.

The other example of Case 2 is provided by the ?-operator. Suppose that **A** does not obtain, and let α be the complex program $\mathbf{A}? \cdot \beta$ ("Carry out first a verification that **A** and then the β-routine!"). Under the circumstances this program is seen to function more like a *conditional imperative* than a categorical one: "If **A** is the case, then carry out the β-routine!"; what is to be done if **A** is not the case is not specified. The BB, unable to verify that **A**, is left suspended in mid-air, as it were.

With the notion of computation according to a program at hand, we may now articulate our intuitive semantics more precisely. From a semantic point of view, our object language is indexical: propositions are not true or false by themselves but true or false only with respect to a state. We write $\vDash_x \mathbf{A}$ to indicate that a proposition $\mathbf{A}$ is true at a state x (still referring to the given BB). We take the truth or falsity at states of primitive propositions for granted. The Boolean conditions work out as usual. It is the new propositional operators that attract interest.

For the two "after"-operators we would presumably have these conditions:

$\vDash_x [\alpha]\mathbf{A}$ iff, for every finite α-computation $\langle z_0, \ldots, z_n\rangle$ such that $z_0 = x$, $\vDash_{z_n} \mathbf{A}$.

$\vDash_x \langle\alpha\rangle \mathbf{A}$ iff, for some finite α-computation $\langle z_0, \ldots, z_n\rangle$ such that $z_0 = x$, $\vDash_{z_n} A$.

Similarly, for the "during"-operators we would have these conditions:

$\vDash_x [\![\alpha]\!] \mathbf{A}$ iff, for every (finite or infinite) α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$, for all i such that z_i is defined, $\vDash_{z_i} \mathbf{A}$.

$\vDash_x \langle\!\langle\alpha\rangle\!\rangle \mathbf{A}$ iff, for some (finite or infinite) α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$, there is some i such that z_i is defined and $\vDash_{z_i} \mathbf{A}$.

$\vDash_x \langle\!\langle\alpha]\!] \mathbf{A}$ iff, for some (finite or infinite) α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$, for all i such that z_i is defined, $\vDash_{z_i} \mathbf{A}$.

$\vDash_x [\![\alpha\rangle\!\rangle \mathbf{A}$ iff, for every (finite or infinite) α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$, there is some i such that z_i is defined and $\vDash_{z_i} \mathbf{A}$.

We have now developed a semantics of a sort. To make it quite explicit we might backtrack and introduce a notion of model as follows. Let U be the set of all total states of the BB. For each program α, let $C(\alpha)$ be the set of all (finite or infinite) computations according to α. Define

$$C = \{C(\alpha)\colon \alpha \text{ is a program}\}.$$

For each propositional letter $\mathbf{P}$, let $V(\mathbf{P})$ be the set of states in which $\mathbf{P}$ holds. Then all the information used in the preceding truth conditions can be retrieved from the triple $\langle U, C, V\rangle$, which therefore may be called a model based on the given BB.

The final step in making the semantics explicit is to define a proposition as *true for* the BB if it true at every state. The set of all formulas true for the BB may be called the *theory* of the BB. The logician can now formulate the usual questions about theories: whether it is axiomatizable, decidable, etc.

There are several other concepts that can be handled quite naturally within the present framework (cf. [1, 2, 4]). Equivalence between pro-

grams is one. Suppose that $\equiv$ is part of the object language and that $\alpha \equiv \beta$ is a formula whenever α and β are programs, asserting that α and β are "really the same program" or "come to the same thing". The vagueness of this suggestion is removed by the following truth condition:

$\vDash_x \alpha \equiv \beta$ iff every α-computation is a β-computation, and *vice versa* (that is, iff $C(\alpha) = C(\beta)$).

Another example is offerred by the concepts of convergence and divergence. Suppose that we have in our object language, for each program α, propositional constants $\boldsymbol{conv}_\alpha$ and $\boldsymbol{div}_\alpha$. Their truth conditions are the following:

$\vDash_x \boldsymbol{conv}_\alpha$ iff every α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$ is finite.
$\vDash_x \boldsymbol{div}_\alpha$ iff there is an infinite α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$.

This distinction could be built into a competing notion of "after":

$\vDash_x [\alpha]^\dagger \mathbf{A}$ iff every α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$ is finite, and, for every finite α-computation $\langle u_0, \ldots, u_n\rangle$ such that $u_0 = x$, $\vDash_{u_n} A$.

$\vDash_x \langle\alpha\rangle^\dagger \mathbf{A}$ iff either some α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$ is infinite, or there is some finite α-computation $\langle u_0, \ldots, u_n\rangle$ such that $u_0 = x$ and $\vDash_{u_n} \mathbf{A}$.

A more complicated example involves the notion of preservation. Let us say that a program α *preserves* a proposition **A** if, during every α-computation, if **A** is ever true, then **A** stays true. Assume that our object language contains, for each program α, a propositional operator α-***pres***. The corresponding truth condition would seem to be this one:

$\vDash_x \alpha\text{-}\boldsymbol{pres}\ \mathbf{A}$ iff for every (finite or infinite) α-computation $\langle z_0, \ldots\rangle$ such that $z_0 = x$, for all i, if $\vDash_{z_i} \mathbf{A}$ and z_{i+1} is defined, then $\vDash_{z_{i+1}} \mathbf{A}$.

Preservation is a complex concept, and it may be possible to analyse it in terms of simpler ones. For example, one might try to define a new propositional operator ***next*** so that α-***pres*** **A** is rendered by $[\alpha]$ (**A**$\rightarrow$***next*** **A**). Such an operator would be indexical not only with respect to state but also with respect to program; so the semantical apparatus would have to be revised.

One concept that does not seem amenable to treatment within to suggested semantics is that of failure. Suppose that the object language contains, for each program α, a propositional constant $\boldsymbol{fail}_\alpha$. No truth condition in terms of U, C and V would seem to do justice to the intuitive import of these new constants. One way to proceed would be the follo-

wing. For each α, let $F(\alpha)$ be the set of all states at which there is at least one α-failure. Define

$$F = \{F(\alpha)\colon \alpha \text{ is a program}\}.$$

Expand the notion of model based on the given BB from the triple $\langle U, C, V\rangle$ to the quadruple $\langle U, C, F, V\rangle$. Then the previous truth conditions are unchanged, and the following one is added:

$\models_x \boldsymbol{fail}_\alpha$ iff $x \in F(\alpha)$.

A much greater revision would be the following. First, for each program α, define $P(\alpha)$ as the set of (finite or infinite) paths $\langle z_0, \ldots\rangle$ that can be produced by the BB under α if started while in state z_0; thus $C(\alpha) \subseteq P(\alpha)$. Then define model as the quadruple $\langle U, C, P, V\rangle$. The same notion of failure would now be captured by this truth-condition:

$\models_x \boldsymbol{fail}_\alpha$ iff there is a finite element $\langle z_0, \ldots, z_n\rangle$ in $P(\alpha)$ such that $z_0 = x$ which is not also an α-computation.

Other concepts of failure can be defined in a similar manner.

Thus we see that, just as the concept of modal logical language suitable for the given BB allows many instances, so the concept of model based on the given BB is not uniquely determined. Roughly speaking, there is always more information about the BB than can be built into any one tuple. Your choice of both language and model depends on what you are interested in.

It is time to end the intuitive discussion. So as to make sure that intuition and formal development are not confused, we start all over again when we now go to the next section, moving from applied logic to pure.

5. Abstract languages and logics

In this section we shall first give a definition of *language for* (*propositional*) *dynamic logic*, which will be seen to be an abstraction of the intuitive notion of modal logical language suitable for a BB, discussed in the preceding section. In laying down such a definition, it is convenient to use a wellknown idea due to Ajdukiewicz and recently revived by Montague.

A language of this kind is defined in three steps. First one defines the structure of syntactic categories. They are, by our definition, always the same: there are two basic syntactic categories, f (*propositional expressions*, or *formulas*) and p (*program expressions*, or just *programs*). Furthermore, whenever $a, b_0, \ldots, b_{n-1}$ are syntactic categories, basic or derived, then

$$a^{b_0,\ldots,b_{n-1}}$$

is a new derived category.

The second step of the definition of language consists in defining, for each syntactic category s, the set $B(s)$ of *basic expressions of category s*, any two of which are disjoint. (Usually all but finitely many of the categories will be empty.) We shall make the following assumptions: $B(\mathfrak{f})$, the set of basic formulas, will always contain a set of elements called *propositional letters*; the others are called *propositional constants*. Similarly, $B(\mathfrak{p})$, the set of basic programs, will always contain a set of elements called *program letters*; the others are called *program constants*.

The third and final step in the definition of a language consists in recursively defining, for each syntactic category s, the full sets $C(s)$ of *expressions of category s*. This step is the simplest since it can be taken care of once and for all: the sets $C(s)$ are to be the smallest sets such that (i) for each category s, $B(s) \subseteq C(s)$, and (ii) whenever

$$\mathbf{X} \in C(a),\ \mathbf{Y}_0 \in C(b_0), \ldots \mathbf{Y}_{n-1} \in C(b_{n-1}),$$

then

$$\mathbf{X}(\mathbf{Y}_0, \ldots, \mathbf{Y}_{n-1}) \in C(a^{b_0,\ldots,b_{n-1}}).$$

Montague assures us that a definition of this kind is correct.

With the first and third steps in the definition of language fixed, it is the second one that matters. Hence a language might be identified with the function B assigning to each syntactic category s the set $B(s)$.

In the light of this, let us now review some of the material of Section 4. First this example: suppose that we have a language such that whenever α is a program (expression of category $\mathfrak{p}$), then $[\alpha]$ is a propositional operator (expression of category $\mathfrak{f}^{\mathfrak{f}}$). The generality of the present approach would then make it natural to introduce an operator $[\ ]$ of category $(\mathfrak{f}^{\mathfrak{f}})^{\mathfrak{p}}$ and agree to write $[\alpha]$ instead of $[\ ](\alpha)$. Alternatively, it would be possible to introduce $[\ ]$ as an expression of one of the categories $\mathfrak{f}^{\mathfrak{f}\mathfrak{p}}$, $\mathfrak{f}^{\mathfrak{p}\mathfrak{f}}$, and $(\mathfrak{f}^{\mathfrak{p}})^{\mathfrak{f}}$ instead. In the former two cases, if $\mathbf{A}$ is a formula, we would agree to write $[\alpha]\mathbf{A}$ for $[\ ](\mathbf{A}, \alpha)$ or $[\ ](\alpha, \mathbf{A})$, respectively. In the last case we would agree to write $[\ ]\mathbf{A}$ for $[\ ](\mathbf{A})$ — which would be an expression of category $\mathfrak{f}^{\mathfrak{p}}$: a formula-making program operator — and $[\alpha]\mathbf{A}$ for $[\ ]\mathbf{A}(\alpha)$.

The following is a list of possible expressions suggested by the discussion in Section 4. The idea is that if the expression indicated on the left is in the language, then it is of the category listed on the right. The typographical shapes should encode enough information to make the list self-explanatory:

$\top, \bot$	$\mathfrak{f}$
$\neg$	$\mathfrak{f}^{\mathfrak{f}}$
$\wedge, \vee, \rightarrow, \leftrightarrow$	$\mathfrak{f}^{\mathfrak{f},\mathfrak{f}}$
$\mathbf{0}, \mathbf{1}$	$\mathfrak{p}$
$*$	$\mathfrak{p}^{\mathfrak{p}}$
$+, \cdot$	$\mathfrak{p}^{\mathfrak{p},\mathfrak{p}}$

?	$\mathfrak{p}^{\mathfrak{f}}$
$[\ ], \langle\rangle, [\![ ]\!], \langle\!\langle \rangle\!\rangle, \langle\!\langle]\!], [\![\rangle\!\rangle$, ***pres***	$(\mathfrak{f}^{\mathfrak{f}})^{\mathfrak{p}}$ or $(\mathfrak{f}^{\mathfrak{p}})^{\mathfrak{f}}$ or $\mathfrak{f}^{\mathfrak{f},\mathfrak{p}}$ or $\mathfrak{f}^{\mathfrak{p},\mathfrak{f}}$
$\equiv$	$\mathfrak{f}^{\mathfrak{p},\mathfrak{p}}$
conv, div, fail	$\mathfrak{f}^{\mathfrak{p}}$

We take this opportunity to list some conventions regarding notation. We use **A, B** as generic names of formulas; α, β as generic names of programs; **P** as a generic name of propositional letters; π as a generic name of program letters; i, j, k, l, m, n as generic names of the members of the set ω of natural numbers $0, 1, 2, \ldots$; s as a generic name of syntactic categories. Our use of parentheses is informal. Instead of $\neg(\mathbf{A})$, $\rightarrow(\mathbf{A}, \mathbf{B})$, etc., we write $\neg\mathbf{A}$, $\mathbf{A}\rightarrow\mathbf{B}$, etc. Instead of $+(\alpha, \beta)$, $\cdot(\alpha, \beta)$, $^*(\alpha)$ we write $\alpha+\beta$, $\alpha\beta$, α^*.

In the present context, classical modal logic may be characterized as the study of operators of category $\mathfrak{f}^{\mathfrak{f}}$. A good deal is known about such operators, and when one develops dynamic logic — which used to be called the modal logic of programs — it seems natural to try to draw on this knowledge. Let us quickly repeat some wellknown concepts in this area.

A *logic* (in a given language) is a set of formulas containing all truth-functional tautologies, closed under modus ponens and substitution (of formulas for propositional letters). If L is a logic we write $\vdash_L \mathbf{A}$ or even $\vdash \mathbf{A}$, when confusion does not arise, for $\mathbf{A} \in L$.

An operator $\bigstar$ is *congruential* in a logic L if $\vdash \mathbf{A}\leftrightarrow\mathbf{B}$ implies that $\vdash \bigstar\mathbf{A}\leftrightarrow\bigstar\mathbf{B}$. Furthermore, $\bigstar$ is *regular* in L if $\bigstar$ is congruential in L and distributes over conjunction; that is, $\vdash \bigstar(\mathbf{A}\wedge\mathbf{B})\leftrightarrow(\bigstar\mathbf{A}\wedge\bigstar\mathbf{B})$. Finally, $\bigstar$ is *normal* in L if $\bigstar$ is regular and $\vdash \mathbf{A}$ implies that $\vdash \bigstar\mathbf{A}$. Notice that the so-called Kripke schema is derivable for operators that are at least regular: if $\bigstar$ is regular then $\vdash \bigstar(\mathbf{A}\rightarrow\mathbf{B})\rightarrow(\bigstar\mathbf{A}\rightarrow\bigstar\mathbf{B})$.

Extending this terminology, let us say that an operator Δ of category $(\mathfrak{f}^{\mathfrak{f}})^{\mathfrak{p}}$ is congruential (regular, normal) if, for each α, $\Delta(\alpha)$ is a congruential (regular, normal) operator of category $\mathfrak{f}^{\mathfrak{f}}$.

The recent history of modal logic shows that many normal propositional operators can be given an extensive analysis within relational Kripke semantics, while regular operators can be handled in a slightly modified Kripke semantics. The study of congruential operators has attracted much less attention, but some work has been done with the neighborhood semantics usually ascribed to Montague and Scott. (The much older algebraic semantics due to Tarski and his followers is not considered here.)

6. Formalizing "after" and "during"

Of the intuitive operators listed in Section 4, at least $[\alpha]$ and $[\![\alpha]\!]$ seem to be normal. For this reason we single them out for study in this section

and will try to give them the usual Kripke type treatment. (Of the others, $[\alpha]^{\dagger}$ seems to be regular, $\langle\!\langle\alpha]\!]$ and $[\![\alpha\rangle\!\rangle$ congruential. Concerning the former, see [2, 5, 8]. One would conjecture that the latter two could be given some kind of neighborhood analysis.)

To be quite specific, let us list a particular object language to be used in this section. All basic categories are to be empty except the following:

$$B(\mathfrak{f}) = \{\boldsymbol{prop}_0, \boldsymbol{prop}_1, \ldots, \boldsymbol{prop}_n, \ldots\};$$
$$B(\mathfrak{p}) = \{\boldsymbol{progr}_0, \boldsymbol{progr}_1, \ldots, \boldsymbol{progr}_n, \ldots\};$$
$$B(\mathfrak{f}^{\mathfrak{f}}) = \{\neg\};$$
$$B(\mathfrak{f}^{\mathfrak{f},\mathfrak{f}}) = \{\wedge, \vee, \rightarrow, \leftrightarrow\};$$
$$B(\mathfrak{p}^{\mathfrak{p}}) = \{*\};$$
$$B(\mathfrak{p}^{\mathfrak{p},\mathfrak{p}}) = \{+, \cdot\};$$
$$B((\mathfrak{f}^{\mathfrak{f}})^{\mathfrak{p}}) = \{[\ ], [\![\ ]\!]\}.$$

Thus $\langle\ \rangle$ and $\langle\!\langle\ \rangle\!\rangle$, duals of $[\ ]$ and $[\![\ ]\!]$, respectively, are not primitive here. But they can be introduced by definition in the usual way:

$$\langle\alpha\rangle\,\mathbf{A} =_{\mathrm{df}} \neg[\alpha]\neg\mathbf{A},$$
$$\langle\!\langle\alpha\rangle\!\rangle\,\mathbf{A} =_{\mathrm{df}} \neg[\![\alpha]\!]\neg\mathbf{A}.$$

The next task is to develop a semantics for this language. Instead of directly building a concept of model along the lines of Section 4, we shall proceed more obliquely. The reason for this strategy will become apparent.

By a *model* let us mean a quadruple $\mathfrak{M} = \langle U, R, S, V\rangle$ such that the following conditions obtain:

(i) U is a set (the *domain*);

(ii) $\{R(\alpha)\colon \alpha \in C(\mathfrak{p})\}$ and $\{S(\alpha)\colon \alpha \in C(\mathfrak{p})\}$ are families of binary relations on U (the *R-alternative relations* and the *S-alternative relations*, respectively);

(iii) V is a function from $B(\mathfrak{f})$ to the power set of U (the *valuation*).

With our definition of language it can easily be shown that the formulas — the elements of $C(\mathfrak{f})$ — are exactly the basic ones — the elements of $B(\mathfrak{f})$ —, Boolean compounds of other formulas, and expressions of type $[\alpha]\,\mathbf{A}$ or $[\![\alpha]\!]\,\mathbf{A}$. The semantic definition of *truth in* $\mathfrak{M}$ *of* a formula $\mathbf{A}$ *at* a point $x \in U$ can therefore be given in the expected way. We write $\mathfrak{M} \vDash_x \mathbf{A}$ for this notion.

$\mathfrak{M} \vDash_x \mathbf{P}$ iff $x \in V(\mathbf{P})$, if $\mathbf{P} \in B(\mathfrak{f})$;
$\mathfrak{M} \vDash_x \mathbf{A}\wedge\mathbf{B}$ iff $\mathfrak{M} \vDash_x \mathbf{A}$ and $\mathfrak{M} \vDash_x \mathbf{B}$;
and similarly for the other Boolean operators;
$\mathfrak{M} \vDash_x [\alpha]\,\mathbf{A}$ iff $\forall y(xR(\alpha)y \Rightarrow \mathfrak{M} \vDash_y \mathbf{A})$;
$\mathfrak{M} \vDash_x [\![\alpha]\!]\,\mathbf{A}$ iff $\forall y(xS(\alpha)y \Rightarrow \mathfrak{M} \vDash_y \mathbf{A})$.

A formula is said to be *true in* $\mathfrak{M}$ if it is true at every point of the model.

So far it is not all clear what the preceding modelling has to do with the intuitive "after" and "during". In fact, as it stands the modelling just introduced is too general to be of other than technical interest. In order to define the kind of model we really want, we need some concepts from automata theory, and we shall now turn to them.

A *word* is any sequence, finite or infinite, of program letters (and is thus of order type $\leqslant \omega$). A special case is the empty sequence, λ, which is called *the empty word* (there is only one!). We shall use σ, τ as generic names of words. (The reader will note that even a finite word $\pi_0 \dots \pi_n$ is not a program expression, even though our informal mode of notation may suggest that it is. For example, a word $\pi_0\pi_1$ is a sequence and would be written $\langle \pi_0, \pi_1 \rangle$ or even $\{\langle 0, \pi_0\rangle, \langle 1, \pi_1\rangle\}$ in a careful exposition. The program expression $\pi_0\pi_1$, on the other hand, is actually a shape $\cdot(\pi_0, \pi_1)$, and thus something quite different. This is not to say that the two notions are not intimately related — they are, as will be seen presently.)

If σ and τ are words and σ is finite, then *the concatenation* $\sigma\tau$ *of* σ and τ — the sequence obtained by concatenating σ and τ, in that order — is also a word, finite if and only if τ is finite. When below we write $\sigma\tau$ or, in general, $\sigma_0 \dots \sigma_n$ with $n > 0$, we implicitly assume that σ respectively $\sigma_0, \dots, \sigma_{n-1}$ are finite.

The *language* of a program expression α, denoted by $|\alpha|$, is defined as follows:

$$|\pi| = \{\pi\}, \quad \text{if} \quad \pi \in B(\mathrm{p});$$

$$|\alpha + \beta| = |\alpha| \cup |\beta|;$$

$$|\alpha\beta| = \{\sigma\colon (\sigma \in |\alpha| \text{ and } \sigma \text{ is infinite}) \text{ or } \exists \tau_1 \in |\alpha|\ \exists \tau_2 \in |\beta|(\sigma = \tau_1\tau_2)\};$$

$$|\alpha^*| = \{\sigma\colon \exists n \forall \tau_0, \dots, \tau_{n-1} \in |\alpha|(\sigma = \tau_0 \dots \tau_{n-1}) \text{ or } \forall n\, \exists \tau_n \in |\alpha|(\sigma = \tau_0\tau_1 \dots \tau_n \dots)\}.$$

Notice that $\lambda \in |\alpha^*|$, for all α.

If $\sigma \in |\alpha|$, then we say that σ *instantiates* α. (The preceding definitions are related to, though slightly different from, those of Salomaa [6] (where words are finite sequences). The use of the word "language" in this technical sense may be somewhat awkward in the present context, but it has seemed desirable to follow standard usage.)

Let $\mathfrak{M} = \langle U, R, S, V\rangle$ be any model; actually, for the following definitions we only need U and R as defined on $B(\mathrm{p})$. We say that a nonempty, finite sequence $\mathbf{z} = \langle z_0, \dots, z_n\rangle$ of not necessarily distinct elements of U is a *finite* α*-computation* (in $\mathfrak{M}$) if there is a word $\pi_0 \dots \pi_{n-1} \in |\alpha|$ such that, for all $i < n$, $z_i R(\pi_i) z_{i+1}$; the *length* of $\mathbf{z}$ is n. Similarly we say that an infinite sequence $\mathbf{z} = \langle z_0, z_1, \dots, z_n, \dots\rangle$ of elements of U, again not necessarily distinct, is an *infinite* α*-computation* (in $\mathfrak{M}$) if there is an infinite word $\pi_0\pi_1 \dots \pi_n \dots \in |\alpha|$ such that, for all $i < \omega$, $z_i R(\pi_i) z_{i+1}$;

the *length* of $\mathbf{z}$ is of course ω. In both cases we say that the computation is *from* z_0 or that it *starts at* z_0; furthermore, that it is *through* each z_i in the computation. In the finite case we also say that the computation is *to* z_n or that it *terminates at* z_n. Finally, $\pi_0 \dots \pi_{n-1}$ respectively $\pi_0\pi_1 \dots$ $\dots \pi_n \dots$ is said to be a *word of* $\mathbf{z}$ (notice the indefinite article — a computation may have more than one word).

Notice that, for every $x \in U$, $\langle x \rangle$ is an α-computation of length 0, whenever $\lambda \in |\alpha|$. Furthermore, for any program letter π, if $\langle z_0, \dots, z_n \rangle$ is a π-computation, then $n = 1$ and $z_0 R(\pi) z_1$.

We remarked above that our notion of model is too general for our purposes. However, in possession of the formal concept of computation, we can now define a more relevant notion of model. Let us say that a model $\mathfrak{M} = \langle U, R, S, V \rangle$ is a *standard model* if and only if the following two conditions are satisfied, for each α:

$$xR(\alpha)y \text{ iff } \exists \mathbf{z}(\mathbf{z} \text{ is an } \alpha\text{-computation from } x \text{ to } y);$$
$$xS(\alpha)y \text{ iff } \exists \mathbf{z}(\mathbf{z} \text{ is an } \alpha\text{-computation from } x \text{ through } y).$$

It is an interesting problem whether standard models can be characterized by simple set theoretic conditions on R and S. For a large class of standard models an affirmative answer is given by Theorem 1. Let us say that a binary relation T is serial if $\forall x \exists y\ (xTy)$; that a model $\langle U, R, S, V \rangle$ is *serial* if $R(\alpha)$ is serial, for every α.

THEOREM 1. *A necessary and sufficient condition for a serial model* $\mathfrak{M} = \langle U, R, S, V \rangle$ *to be standard is that the following conditions are satisfied:*

$(R+)$	$R(\alpha+\beta) = R(\alpha) \cup R(\beta)$.
$(R\cdot)$	$R(\alpha\beta) = R(\alpha) \mid R(\beta)$.
(R^*)	$R(\alpha^*) = R^*(\alpha)$.
(RS)	$S(\pi) = R(\pi) \cup \{\langle x, x \rangle : x \in U\}$.
$(S+)$	$S(\alpha+\beta) = S(\alpha) \cup S(\beta)$.
$(S\cdot)$	$S(\alpha\beta) = S(\alpha) \cup R(\alpha) \mid S(\beta)$.
(S^*)	$S(\alpha^*) = R(\alpha^*) \mid S(\alpha)$.

The proof is straightforward but too long to be included here.

The theorem is interesting in its own right. But it is also a useful lemma for the proof of the following result:

THEOREM 2. *The set of formulas true in all serial standard models is the smallest logic in which both* $[\ ]$ *and* $[\![\ ]\!]$ *are normal and all instances of the following schemata appear as elements:*

$(\alpha+)$	$[\alpha+\beta]\mathbf{A} \leftrightarrow [\alpha]\mathbf{A} \wedge [\beta]\mathbf{A}$.
$(\alpha\cdot)$	$[\alpha\beta]\mathbf{A} \leftrightarrow [\alpha][\beta]\mathbf{A}$.
$(\alpha^* T)$	$[\alpha^*]\mathbf{A} \rightarrow \mathbf{A}$.
(α_1^*)	$[\alpha^*]\mathbf{A} \rightarrow [\alpha]\mathbf{A}$.

(a_2^*) $[\alpha^*]\mathbf{A}\to[\alpha^*][\alpha^*]\mathbf{A}$.

(a^*ind) $\mathbf{A}\to([\alpha^*](\mathbf{A}\to[\alpha]\mathbf{A})\to[\alpha^*]\mathbf{A})$.

(aD) $[\pi]\mathbf{A}\to\langle\pi\rangle\mathbf{A}$.

(dT) $(\llbracket\alpha\rrbracket\mathbf{A}\to\mathbf{A})$.

(ad_1) $\llbracket\alpha\rrbracket\mathbf{A}\to[\alpha]\mathbf{A}$.

(ad_2) $\mathbf{A}\to([\pi]\mathbf{A}\to\llbracket\pi\rrbracket\mathbf{A})$.

$(d+)$ $\llbracket\alpha+\beta\rrbracket\mathbf{A}\leftrightarrow\llbracket\alpha\rrbracket\mathbf{A}\wedge\llbracket\beta\rrbracket\mathbf{A}$.

$(d\cdot)$ $\llbracket\alpha\beta\rrbracket\mathbf{A}\leftrightarrow\llbracket\alpha\rrbracket\mathbf{A}\wedge[\alpha]\llbracket\beta\rrbracket\mathbf{A}$.

(d^*) $\llbracket\alpha^*\rrbracket\mathbf{A}\leftrightarrow[\alpha^*]\llbracket\alpha\rrbracket\mathbf{A}$.

Unfortunately, the proof is much too long to be reproduced here. Suffice it to say that it is of the canonical models/filtrations type, similar to that given in [7], and so it also yields as a by-product the f.m.p. with bounds that can be estimated. The author hopes to publish the proof elsewhere.

This marks the end of the formal work. It is submitted that the formal notion of standard model successfully formalizes an important part of the intuitions described in Sections 2-4. Theorem 2 also gives a feeling for what the logic of these notions is like. It is of some interest to note that it is not closed under substitution of program expressions for program letters. To be certain, every substitution of a program expression for the program letter in an instance of (aD) yields a derivable formula, as is readily checked. But in the case of (ad_2) this is not so: it is easy to find, for any given propositional letter $\mathbf{P}$ and program letters π_0 and π_1, a serial standard model rejecting the formula

$$\mathbf{P}\to([(\pi_0\pi_1)^*]\mathbf{P}\to\llbracket(\pi_0\pi_1)^*\rrbracket\mathbf{P}).$$

7. Philosophical relevance of dynamic logic

With formalities out of the way it is time to take stock of the situation. What has this paper got to do with the philosophical problems pertaining to applying modal logic? Even a hostile critic of applied modal logic will admit that dynamic logic offers an interpretation of modal logic that is beyond formal reproach—fruitful or not, it is precise and perfectly intelligible. But, such a critic might continue, apart from some possible methodological interest, what does it matter to philosophy whether dynamic logic is going to benefit computer science?

Not much, perhaps. Yet this paper has been written in the belief that Pratt's dynamic logic has something to offer philosophy. Its main virtue, according to this belief, is that it points to a new way of approaching the logic of action. The formalism of dynamic logic separates talk about actions from talk about states-of-affairs. That it does, and the way it does it, is what seems so interesting.

It is not possible to go deeply into this issue here. But to lend some

substance to the preceding paragraph we will briefly consider two examples, emphasizing that they are no more than sketches, and quick ones at that.

As the first example, take "Anyone who is killed dies". It is not a very interesting proposition, but it is true nevertheless, and in a peculiar way: not for empirical reasons but because of what the words mean. Whether you choose to call it an analytic truth or a logical truth, you will want to see to it that it becomes a truth in your formalization; linguists in effect do when they analyse "kill" as "cause to die", as they sometimes do. Disregarding various subtleties, we would formalize this proposition in dynamic logic by something like

$$[\textit{kill } x]\ (x \textit{ is dead}),$$

the brackets being the by now familiar ones. Or, to sidetrack for now the difficulties attaching to free variables and to individuals, consider the simpler instance

$$[\textit{kill Caesar}]\ (\textit{Caesar is dead}).$$

Here *kill Caesar* is what might be called an action program, standing for a certain type of action. On the other hand, *Caesar is dead* is a proposition, reporting a certain state-of-affairs. The propositional operator [*kill Caesar*] can be read "after Caesar is killed" or, more carefully, "after Caesar is killed, no matter how".

On the semantic side, propositions are identified with sets of possible worlds, as before, while actions are identified with sets of sequences of possible worlds. There would be a notion of model in general, as in Section 6, but the interest would focus on the standard models. The latter would have to be defined in such a way that, among other things, Caesar is left dead at the end of every possible run of the kill Caesar-action program. Other models are logically possible, but they would not be standard.

The second example is from deontic logic. Suppose Kim receives his father's permission to go to the beach or to the movies. If this is formalized in the usual way — writing ***perm*** in place of the more wellknown **P** — as

$$\boldsymbol{perm}\ (\textit{Kim is at the beach} \vee \textit{Kim is at the movies}),$$

then we also encounter the usual difficulties; for example, even though we can infer the disjunction of the two propositions

$$\boldsymbol{perm}\ (\textit{Kim is at the beach}),$$
$$\boldsymbol{perm}\ (\textit{Kim is at the movies}),$$

we cannot infer either one by itself. Many authors have pointed out that this does not seem intuitively satisfactory (the problem of free choice permission). Now, the operator ***perm*** is of category $\mathfrak{f}^{\mathfrak{f}}$. If instead we

introduced a new operator ***PERM*** of category $\mathfrak{f}^{\mathfrak{p}}$ it becomes natural to render Kim's permission as

$$\boldsymbol{PERM}\ (go\ to\ the\ beach + go\ to\ the\ movies),$$

and it is possible to argue that this proposition implies both of the following two:

$$\boldsymbol{PERM}\ (go\ to\ the\ beach),$$
$$\boldsymbol{PERM}\ (go\ to\ the\ movies).$$

That is to say, we have at least two concepts of permission, of different grammatical categories and with different logical properties:

($) $\boldsymbol{perm}\ (\mathbf{A} \vee \mathbf{B}) \leftrightarrow \boldsymbol{perm}\ \mathbf{A} \vee \boldsymbol{perm}\ \mathbf{B}$,
($$) $\boldsymbol{PERM}\ (\alpha + \beta) \leftrightarrow \boldsymbol{PERM}\alpha \wedge \boldsymbol{PERM}\beta$.

To support this analysis one might try to use an idea originally due to Alan Ross Anderson and Stig Kanger: that something is permitted if it can be realized without incurring any sanction or without making the world worse, in a certain sense — the world remains "deontically satisfactory", as it were. Let us use ***OK*** as a propositional constant expressing the proposition that the world is deontically satisfactory, in this sense. Then one way of rendering the Anderson/Kanger suggestion in dynamic logic — there may be other candidates — would be this:

$$\boldsymbol{PERM}\alpha \leftrightarrow [\alpha]\ \boldsymbol{OK}.$$

(Actually this definition must be improved in view of wellknown difficulties discussed at length in the literature — as it stands it can only be used in situations in which no obligation has already been violated.)

Using this simple-minded analysis we readily derive

$$\boldsymbol{PERM}\alpha \wedge \boldsymbol{PERM}\beta \leftrightarrow [\alpha]\ \boldsymbol{OK} \wedge [\beta]\ \boldsymbol{OK},$$
$$\boldsymbol{PERM}(\alpha + \beta) \leftrightarrow [\alpha + \beta]\ \boldsymbol{OK},$$

whence ($$) follows from schema (a+) in Theorem 2.

What is interesting in this example is not that we have hit upon a new permission operator of a certain intuitive plausibility — by now, several have been suggested — but that both our permission operators arise within the system in such a natural way.

8. Conclusion

Pioneers in applied modal logic were preoccupied with propositional operators (taking propositions into propositions). From them we may have inherited an exaggerated tendency to expect that intensional notions must be represented by such operators. If so, it is a tendency that needs tempering. Already Montague's work gives some perspective on

this issue. So does that of Pratt, but a different perspective. Philosophers outside modal logic have already argued that actions — and events — ought to be considered a basic category. Dynamic logic suggests one way to do so, and it deserves to be explored.

References

[1] M. J. Fischer and R. E. Ladner, *Propositional modal logic of programs*, extended abstract, presented at the Ninth A. C. M. Symposium on the Theory of Computing, Boulder, Colorado, May 2-4, 1977.

[2] D. Harel and V. R. Pratt, *Nondeterminism in logics of programs*, manuscript, dated November 10, 1977.

[3] V. R. Pratt, *Semantical considerations on Floyd-Hoare logic*, ***17th I.E.E.E. Symposium on Foundations of Computer Science*** (1977), pp. 109-121.

[4] —, *Logic of processes*: manuscript, dated November 29, 1977.

[5] —, *Applications of modal logic to programming*, this issue of ***Studia Logica***.

[6] A. Salomaa, ***Theory of Automata***, Pergamon Press, London, 1969.

[7] K. Segerberg, *A completeness theorem in the modal logic of programs*, to appear in the Stefan Banach International Mathematical Centre publications series.

[8] —, *A conjecture in dynamic logic*, in: ***Mini-essays in honor of Juhani Pietarinen***, Åbo, 1978, pp. 23-26.

Åbo Academy
Åbo, Finland

Received September 10, 1979.