A Representation of Action Structures

Erik Sandewall and Ralph Rönnquist
Department of Computer and Information Science
Linköping University
Linköping, Sweden

Abstract: We consider structures of actions which are partially ordered for time, which may occur in parallel, and which have lasting effects on the state of the world. Such action structures are of interest for problem–solving with multiple actors, and for understanding narrative texts where several things are going on at the same time. They are also of interest for other branches of computer science besides A.I.

Actions in the action structure are characterized in terms of preconditions, postconditions, and prevail conditions, where the prevail condition is a requirement on what must hold for the duration of the action. All three conditions are partial states of the world, and therefore elements of a lattice. We develop the formalism, give an example, and specify formally the criterion for admissible action' structures, where postconditions of earlier actions serve as prevail– or preconditions of later actions in a coherent way, and there are no conflicting attempts to change ("update") a feature in the world.

1. Introduction.

Our topic is the formal analysis of actions, i.e. things that happen in the world. The phenomenon described by the phrase 'John gives the ball to Mary', and the operation where an industrial robot moves a workpiece from a conveyor to an NC–machine, are examples of actions. Actions have a duration in time, and several actions may occur in parallel; these are essential properties of actions. It is not essential that there should be an identifiable actor who 'does' the action, so 'thunder' or 'a thunderstorm' could also qualify as an action. We shall use the term 'action structure' for a set of actions together with information about them, in particular, information about their relative order in time.

A formal analysis of action structures must be very much concerned with their effects: will (or may) a given action structure have a certain effect on the world; are two given action structures equivalent with respect to their effects on the world, etc.

Action structures have been studied in several branches of computer science (and also of course in several other disciplines), but with different and sometimes complementary assumptions or constraints. Usually action structures have been seen as 'plans' or 'programs', i.e. *pre*–scriptions for intended behavior in a machine. It is however also possible to see an action structure as a *de*–scription of what has happened, an account of history. In A.I., research on 'planning and problem–solving' has traditionally focused on the analysis of effects of sequences of actions, and has only recently begun to address the complicating issue of parallell actions. (A discussion of related work in this field is in section 11 of this paper).

On the other hand, the Operating systems and Data base fields have for a long time included work on concurrent programming, where the main issue is "how two or more sequential programs may be executed concurrently as parallel processes" [Andr83]. In principle there should not be any particular difference between structures of actions in the real world, and actions inside computers. In practice, however, there is a difference which is also indicated by the very term "concurrent program": one deals with a number of programs, one for each processor, and uses special constructs for synchronization. This works well if each of the programs is relatively complex, and synchronization can be perceived as a relatively marginal annotation.

For action structures in the real world, it is less natural to use the "concurrent program" viewpoint. One would prefer to make statements about what actions happen, and in which order they happen. That is therefore the approach that is taken in the present paper. – Section 11 also contains a more extensive discussion of how results from concurrent programming research relates to our topic.

2. Key ideas and results.

The key ideas in this paper are:

– An action structure is viewed as a set of actions, each of which has a start–point and an end–point. The set of such points is partially ordered for time.

– Partial state descriptions are used, where each 'feature' or proposition in a partial state description can have a definite value (e.g. a truth–value), or be undefined. The relation of 'containing more information than' defines a lattice over the space of partial state descriptions.

– Each action is associated with a ***pre-condition***, a ***post-condition***, and a ***prevail-condition***. All three conditions are partial states. The precondition and postcondition characterize what must hold at the beginning and the end of the action, respectively. The effect of the action on the world is therefore characterized (explicitly or implicitly) by the pre– and postconditions. The prevail–condition characterizes what must hold for the whole duration of the action.

For a concrete example, consider a small car rental company with only one outlet. The action of 'customer renting a car' has as its precondition and as its post–condition that the car is in the company premises, but it does not have to be in the premises during the rental period. On the other hand, the action of 'mechanic on duty fixing the car' requires the car to be on the premises for the duration of an action, which we would express as a prevail–condition.

This research was supported by the Swedish Board of Technical Development.

Thus in the case of pre–/ postcondition, it is possible but not necessary for the postcondition to be different from the precondition, in which case the action has an effect on those aspects or features of the world that are recorded in the conditions. In the case of prevail– condition, the condition must necessarily be equal at the beginning and the end of the action, since it should hold and be constant for the duration of the action.

In a sense that will become clear below, the pre– and postconditions correspond to the concept of 'non–shared resource' or 'read and write access' in operating systems terms. Prevail–conditions correspond to the concept of 'shared resource' or 'read only access'.

Also, the pre– and post–conditions correspond to the 'delete' and 'add' clauses in Strips–like problem solving systems. The prevail–conditions do not have a direct counterpart in systems like STRIPS, since the problem of whether another, parallel action can violate a prevail–condition of an action, does not arise when all actions are assumed to happen in sequence.

In programming languages such as Occam [May83], parallel programs are constructed using the ***seq***, ***par***, and Kleene star (repetition) operators. Those operators are however not sufficient for constructing all possible (and useful) action structures; Figure 1 shows an example of an action structure that can not be constructed from them. Our approach does not have those restrictions, and is in that sense similar to the Petri net approach [Pete82].

In this paper, we formulate the model and discuss its motivation. We also propose how the admissibility criterion for action structures can be expressed formally, and work out an example.

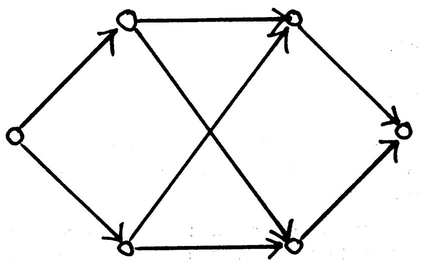

Figure 1

3. Partial models

We described above how, in our approach, the set of start–points and end–points of actions is viewed as a partially time–ordered set. But in every ***actual*** train of occuring actions, the set of time–points is of course totally ordered (as long as we can assume common–sense, Newtonian time). The action structure is therefore a way of characterizing a set of ***similar*** trains of actions. As such, it is an alternative to other ways of characterizing a set of 'admissible worlds'. One other, well–known method would be a logical system, where formulas and their truth–values are defined, and the admissible worlds are characterized using formulas that have the value ***true*** exactly in the admissible worlds [Krip63, Resch71].

When we characterize the momentary state of the world where actions take place, we can similarly choose to use ***partial*** states. In a very simple example, we might characterize the world only in terms of the position of a number of electrical switches, which can be in position '1' or '0'. A partial state in that world would be a function which assigns to each switch, either of the values '1', '0', or 'undefined'.

Partial states go well together with partially time–ordered action structures, for the following reason: if the 'state' (i.e. those aspects of the world that we consider in the formal characterization) consists of a number of components, and two actions happen in parallel or 'at the same time', but they affect distinct and unrelated aspects of the world, then we can analyze their effects without needing to know which of them actually occurred first, and without needing to understand their interactions if they actually occurred at the same time. In such a case, it is reasonable to assign a partial state to the start–point and end–point of each of those actions.

After this introduction, we can now proceed to the formal part of the presentation.

4. The lattice of partial states

We assume that we have a domain S of partial states of the world, and a partial order $\sqsubseteq$ on S such that $<S, \sqsubseteq>$ is a lattice. The lattice operations are written $\sqcup$, $\sqcap$, $\sqsubseteq$ and the top and bottom elements are written $\top$, $\bot$ as usual.

We pay particular attention to domains which are constructed as the Cartesian product of a finite number of feature domains. For example, the world which is characterized by the position of four different switches would be seen as

$F1 \times F2 \times F3 \times F4$

where each of the Fi is the feature domain consisting of the four elements u, 1, 0, and k, with the following order:

$u \sqsubseteq 1 \sqsubseteq k$

$u \sqsubseteq 0 \sqsubseteq k$

as shown also in the Hasse diagram in figure 2. One world–state vector is defined to be $\sqsubseteq$ another world–state iff corresponding elements are $\sqsubseteq$, as usual. We shall generally use u ('undefined') and k ('contradiction') for the bottom and top element in feature domains.

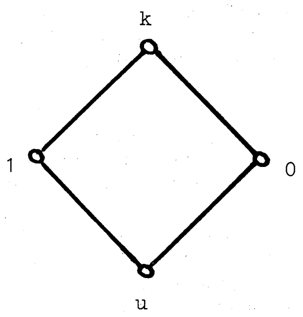

Figure 2

Let s be an element in a domain $F1 \times F2 \times \ldots \times Fn$. The element of Fi which is used for forming s, will be called the ***projection*** of s into the dimension i, and will be written s[i]. The element s will be said to ***have the i:th feature*** iff s[i] is different from u. We write

dim(s)

for the set of all i such that s has the i:th feature.

Two elements s and s' are said to be ***co-dimensional*** iff
dim(s) = dim(s')
and they are said to be ***anti-dimensional*** iff dim(s) and dim(s') are disjoint sets.

The element s is said to be ***consistent*** iff none of its projections equals k.

It will be desirable to generalize these concepts to be used also for those domains that are not formed as Cartesian products, and in particular, for domains that are formed by constraining a Cartesian product using propositions expressed in logic.

5. The domains of operations and actions.

We now introduce a domain V of ***operations***. For example, "to turn on switch number 3" might be one operation. In many cases it will be natural to form operations using a "verb" in some sense, combined with a number of "case slots". In the present treatment we however make no assumptions about the structure of operations.

The domain of ***actions*** is next defined as: an action is a fourtuple
[f, b, v, e]
where f, b, and e are states, and v is an operation. In the sense that was described in section 2 above, f,b, and e are the prevail condition, the pre–condition, and the post–condition, respectively.

From the domain of actions, we distinguish a subset A of ***valid*** actions. Intuitively, valid actions are those fourtuples [f,b,v,e] where f characterizes the state of the world for the duration of the action, b characterizes the state of the world immediately before the operation v takes place, and e characterizes the state of the world at its conclusion.

For example, suppose the states of the world are fourtuples which indicate the position of each of four switches, such as
[1, 0, u, 1]
If TurnOn2 is the operation of turning on switch 2, in a state where it is off, then the following action should be in the set A of valid actions:
[$\perp$, [u,0,u,u], TurnOn2, [u,1,u,u]]
where of course $\perp$ = [u,u,u,u]. In this example the prevail–condition is the bottom element of the partial state lattice because there are no constraints in the prevail–condition.

We always require from valid actions [f,b,v,e] that b and e must be co–dimensional with each other, and anti–dimensional with f.

We introduce an identity operation Noop which leaves every state unchanged, i.e.
[s, $\perp$, Noop, $\perp$]
is a member of A for every s in S.

We are now ready to introduce the action structures themselves, first intuitively/graphically and then formally. We will draw an action structure as in figure 3, where full arrows represent actions. If two actions begin at the same time, they start in the same point; if one immediately succeeds another then the endpoint of one arrow is the beginning–point of the next arrow. If a delay is allowed between one action and the next, then a dotted arrow is drawn from the end–point of one to the beginning– point of the next. In this way, we can also express e.g. that two actions (must) begin at the same time, or that the termination of one (must) preceed the termination of another.

This very natural structure is formally expressed as follows. We use a set T of ***time-points,*** corresponding to the beginning–points and end–points of the arrows in the figure. A partial order $\leq$ is defined on T, representing the order of temporal precedence.

ital

An action structure over a set A of valid actions is now a triple [T, $\leq$, p], where p (for plan) is a set of triples
[t, a, t']
where again t and t' are time–points in T, a is in A, and t $\leq$ t' for every triple in the set. Each member triple
[t, a, t']
or in expanded form,
[t, [f, b, v, e], t']
will be called an ***action occurrence***.

In order to draw a given action structure graphically, we should in principle make one dot for each time–point; represent the temporal order on time–points using dotted arrows, and then indicate the action occurrences using solid arrows, with the understanding that a solid arrow may be drawn on top of the dotted arrow since the action anyway implies that its beginning– point precedes its end–point.

6. Coherent action structures.

The preconditions, postconditions, and prevail–conditions impose a number of constraints on an action structure. Some of the relatively obvious constraints are:

– at the beginning of an action, all its preconditions must be present, either because they were present from the beginning of the action structure, or because they were the result of previous action(s)

– several actions which affect the same 'feature' of a state but in different ways, must not be allowed to occur in parallell. In other words, the temporal order must guarantee that one of them comes before the other

– several actions which require the same prevail–condition may occur in parallel. However, there must not be other, also parallel actions that have a prevail–condition feature in their pre– or postconditions.

The purpose of the present section is to capture these intuitions through a formal definition, which we call for the action structure to be ***coherent***.

These intuitions actually represent a simplification relative to the real world. Consider for example the scenario of parking a car parallel to the curb, between two other cars, starting from the point where 'our' car is positioned to the left of the car in front of the parking slot (in right–hand traffic) (figure 4). We consider three actions:
a1: keep the car moving in the reverse direction, at suitable speed
a2: keep the car's front wheels at an angle pointing right
a3: keep the car's front wheels at an angle pointing left.
The action plan of course is to do a2 and a3 in sequence, and a1 in parallel to both of them. These actions affect the same 'resource' or 'feature' of the state, namely the position of the car. Still, it is admissible and in fact necessary to perform them in parallel – first turning the wheels right and left, and only then moving backwards, would not have the intended effect.

In the present paper we do not account for such coordinated actions. Here we only wish to capture the intuition of actions which can occur in parallel because they do not interfere with each other. In a wider perspective and in future work, it will however be necessary to deal with the case of coordinated actions.

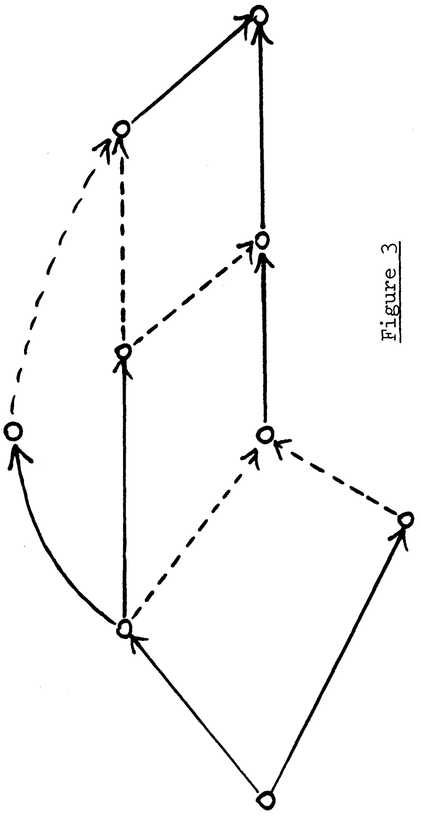

Figure 3

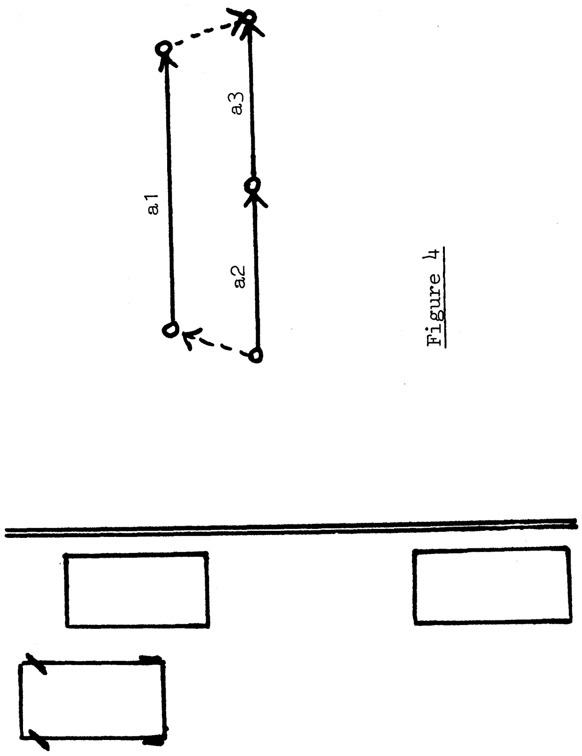

Figure 4

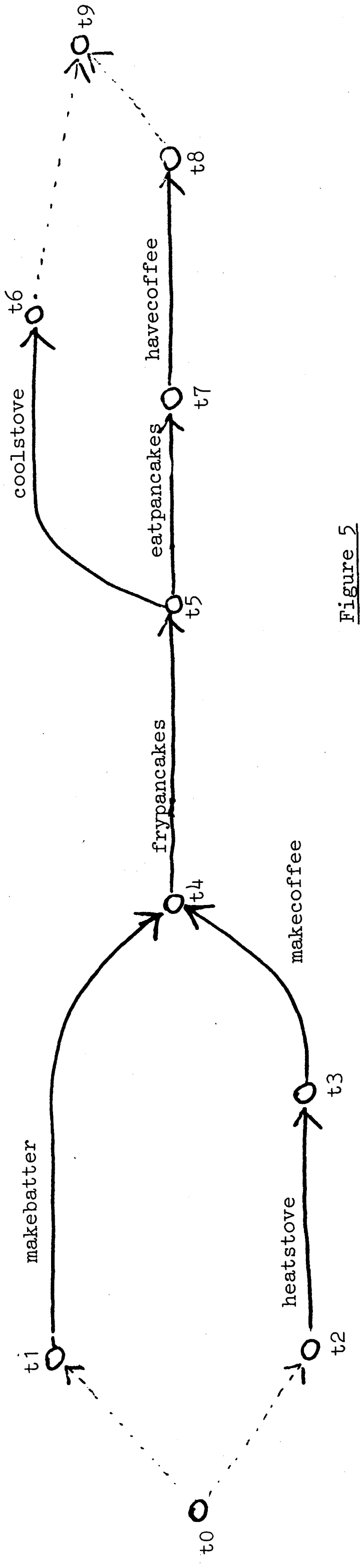

Figure 5

✓ Let [T, ≤, p] be an action structure. For each member t of T, we define the incoming action occurrences in p to be those of the form

[t', a, t]

i.e. having the given t as their last element. Graphically, if each action occurrence is represented as an arrow, the incoming action occurrences are those whose arrows end in the given time—point.

The ***incoming states*** for a time—point t are defined as follows:
— the post—condition of each incoming action occurrence, is an incoming state;
— the join of the prevail—conditions of all the incoming action occurrences, is also an incoming state.

Similarly, the ***outgoing action*** occurrences are those of the form

[t, a, t']

having the given t as their first element, and the ***outgoing*** states are the pre—conditions of the outgoing action occurrences, plus the join of the prevail—conditions of all the outgoing action occurrences.

We are now ready to formulate the coherence criterion.

An action structure [T, ≤, p] is defined to be ***coherent*** if, for every time—point t in T,

1. the incoming states are consistent and anti—dimensional,

2. the outgoing states are consistent and anti—dimensional,

3. the join of the incoming states equals the join of the outgoing states, if the time—point has both incoming and outgoing states.

The join of states mentioned in point 3 will be called the ***current state*** of the time—point t.

Furthermore, an action structure [T, ≤, p] is also coherent if one can add to p some number of action occurrences of the form

[t, [s, ⊥, Noop, ⊥], t']

✓ ~~and the resulting~~ action structure ~~is coherent~~. and obtain a coherent a.s.,
~~where t ≤ t'~~

The way an incoming state cold be inconsistent is if incoming action occurrences have incompatible prevail conditions, and similarly for outgoing states.

This definition captures most of the intuitions, but it leaves out some constraints. We shall first motivate this definition with a concrete example, and then proceed to the additional requirements and the formally derived properties of the concept.

7. An example.

Suppose we are to prepare and consume a meal consisting of pancakes followed by a cup of coffee. The coffee is to be cooked on the stove, and since there is only room for one pot at a time on the stove, and we do not want to interrupt the eating in order to cook, we decide to make the coffee before the pancakes. (Thus hot pancakes have higher priority than fresh cooked coffee). Figure 5 shows the action structure, including the operations of making the batter, heating the stove, and allowing the stove to cool.

In order to analyze the action structure, we use partial states with four truth—value components, namely the answers to the following questions:

is the stove hot?
is there batter?
is there pancakes?
is there coffee?

As before, each component of the partial state is either of u, 0, 1, or k. We write the states without punctuation, so 10uu is for example the partial state where the stove is hot, there is no batter, and otherwise we do not know.

The actions whose operations occur in figure 5 can now be defined as follows:

[⊥, u0uu, MakeBatter, u1uu]
[⊥, 0uuu, HeatStove, 1uuu]
[1uuu, uuu0, MakeCoffee, uuu1]
[1uuu, u10u, FryPancakes, u01u]
[⊥, uu1u, EatPancakes, uu0u]
[⊥, uuu1, HaveCoffee, uuu0]
[⊥, 1uuu, CoolStove, 0uuu]

If we use these actions in the structure of figure 5, and check out the coherence criterion above, we obtain a violation. The key problem is that the result of making the coffee, i.e. the fact that coffee exists, must be 'made known' to the action of having coffee, which of course has coffee existence as a precondition. This is accomplished by adding an action of the form

[uuu1, ⊥, Noop, ⊥]

from node t4 to node t7 in the figure.

One may ask why we do not instead augment the existing arcs from t4 to t5 and from t5 to t7 so as to contain also the information that coffee exists. The reason is that in a more general case, there could have been two (or more) parallell paths from t4 to t7, and then there would be no reason why one or the other should 'carry' the coffee existence information.

There is a similar problem concerning those nodes which are the first ones to have a feature (i.e. no earlier time—point has a current state that has the feature). We shall call such nodes the ***first use*** node(s) for the feature. In order to satisfy the coherence criterion, we have to add Noop actions from the initial time—point t0 to the first use nodes for each feature (at least if the first use node has some predecessor at all). The last nodes have to be similarly connected to the final time— point t9. (This is somewhat inelegant, but we outline below how one can avoid the need to formally introduce those Noop actions). The resulting action structure is shown in figure 6. For simplicity, a Noop action such as

[uuu1, ⊥, Noop, ⊥]

is written just as [uuu1] in the figure, and is drawn as a —·—·—·→ arrow.

It is now trivial to check off that the action structure in figure 6 is coherent. The current state of the respective time— points, i.e. the join of their incoming or outgoing states, is as follows:

t0	0000
t1	u0uu
t2	0uuu
t3	1uu0
t4	1101
t5	101u
t6	0uuu
t7	uu01
t8	uuu0
t9	0000

When action structures are repeated cyclically (for example, in robotics applications, for the programs of manufacturing cells), it is often undesirable to have a single startpoint and endpoint for the cycle. We would like a cycle to have several, parallel first actions, each of which can start as soon as all its prerequisites have been made available. Our model can easily be adapted for that purpose: instead of having the extra Noop actions that go to the first use node and from the last use node for each state feature, we would form a vector of first use nodes and another vector of last use nodes, across the feature space. The definitions of incoming and outgoing states in action structures must of course be modified

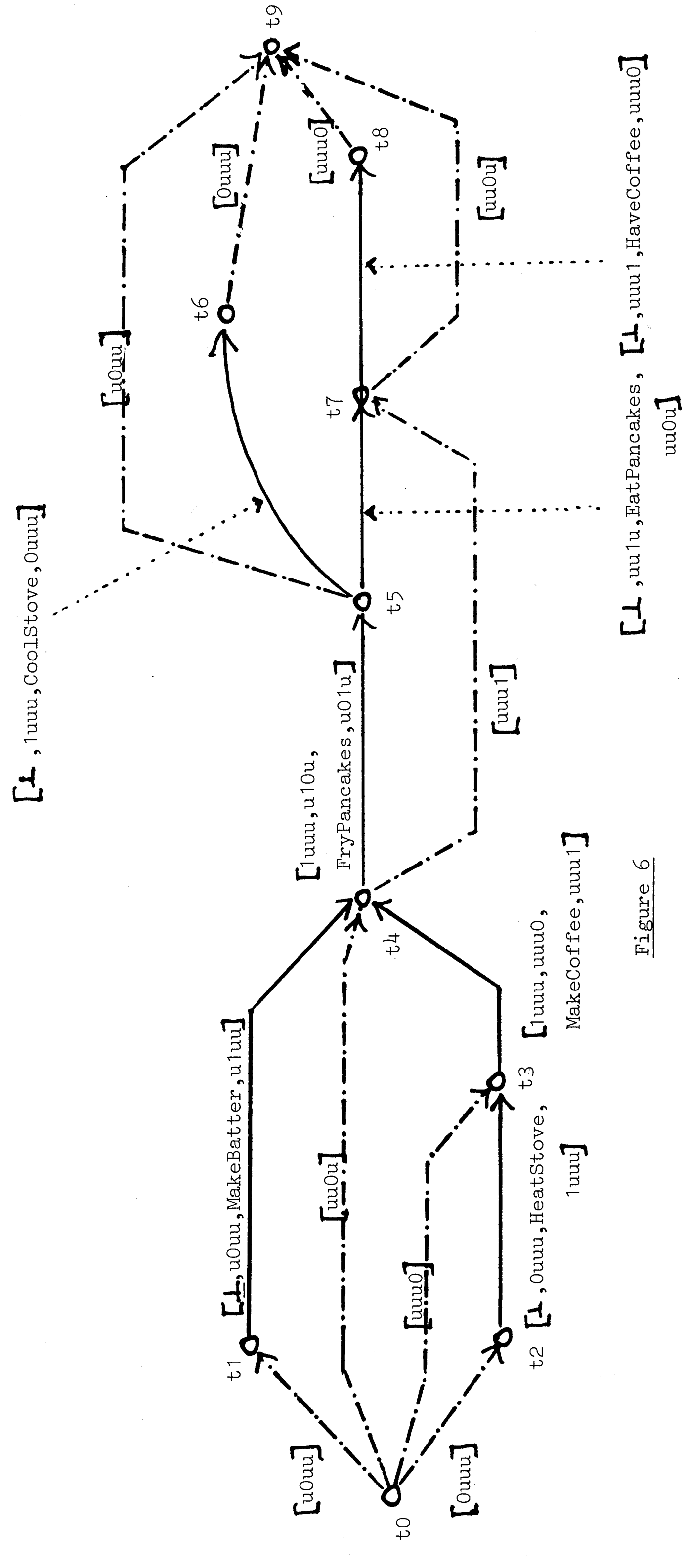

Figure 6

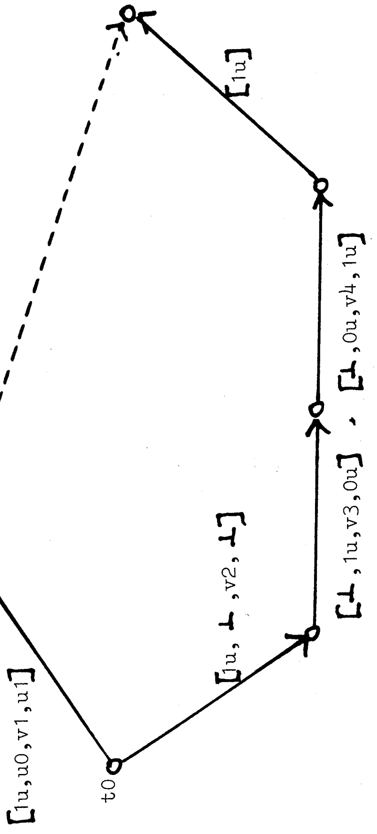

Figure 7

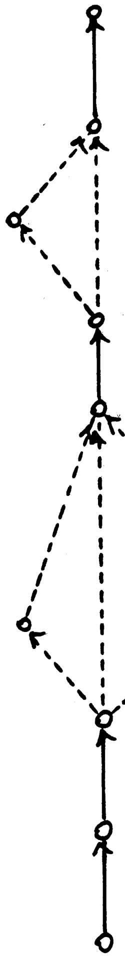

Figure 8

accordingly. The operation of combining two successive cycles is then be performed by introducing an appropriate Noop action from the last use node of each feature in one cycle, to the first use node of the same feature in the next cycle.

8. Additional requirements.

Consider the action structure described in figure 7. It is a prevail—condition of operations v1 and v2 that the first dimension feature shall be 1. The operation v3 changes that value from 1 to 0, and v4 changes it back to 1. The action structure in the figure is coherent, according to the definition in section 6. Yet we see that the action structure may possibly not be correctly executable, namely if operation v3 takes effect before v1 has concluded.

The example illustrates a side—effect problem: the problem which arises if another action, maybe in a remote part of the action structure, locally violates a condition of an action, or at least (with unfortunate timing) threatens to violate it.

The following is a possible way of characterizing that constraint formally:

Let $[T,\leq,p]$ be an action structure. A sequence of action occurrences in p is called a chain iff it has the form

$[t0,a1,t1], [t1,a2,t2],[t2,a3,t3],...$

An action occurrence

$[t, [f,b,v,e], u]$

is said to ***subsume*** another action occurrence

$[t', [f',b',v',e'], u']$

in the i:th feature, iff

$t \sqsubseteq t' \sqsubseteq u' \sqsubseteq u$

and

$f'[i] \sqsubseteq f[i]$

An action structure $[T,\leq,p]$ is now said to be ***aligned*** for the i:th feature iff there is some subset p' of p which is a chain, and where every action occurrence whose f,b, or e has the i:th feature, is either a member of p' or is subsumed by some member of p'.

It is easily seen that in an action structure that is aligned for the i:th feature, those actions whose b and e have the i:th feature (active actions, drawn ⟶) and those whose f have the i:th feature (passive actions, drawn ——————>) together form a structure of the type shown in figure 8. Substructures of passive, possibly parallell actions with a single start—point and end—point, are sequentially combined with active actions.

Our intuitions for admissible action structures can now be formulated as follows: an action structure $[T,\leq,p]$ is ***admissible*** iff there exists some p' which is a superset of p, where all the action occurrences i p'—p ~~are formed using the operation Noop,~~ and where $[T,\leq,p']$ is a coherent ~~and~~ action structure which is aligned for all features.

The following 'model existence' property is stated here without proof:

If $[T,\leq,p]$ is an admissible action structure, and $\preceq$ is a total order over T such that

$t \leq t' \rightarrow t \preceq t'$

then one can assign a consistent state s(t) to each time—point t in T, in such a way that the following holds for every action occurrence $[t', [f,b,v,e], t'']$ in p:

$b \sqsubseteq s(t')\ e \sqsubseteq s(t'')$

and for every t such that $t' < t < t''$,

$s(t)[i] = u$ for each i in dim(b),

and for every t such that $t' \leq t \leq t''$,

$f \sqsubseteq s(t)$

and finally ("frame property"), if $u > t$ is the immediate successor of t, $s(u)[i] = s(t)[i]$ unless the b or e condition of an action forces them to be different according to the above.

9. Verbs or verb phrases that express post- and prevail-conditions.

We have not said anything about the intended structure of operations. From a software engineering background, it may be natural to view operations as essentially procedure calls, i.e. names of procedures with their proper parameters. Pre— conditions, post—conditions, and prevail—conditions are then a part of the specification and/or the description of the procedures, but one would not expect to derive those conditions from the name or the definition (the 'body') of the operation.

If the operations are instead thought of as verb phrases in natural language, this picture changes somewhat. A verb phrase like '(to) open the door' directly suggests what is the postcondition, and also (taking for granted that one can not open a door that is already open) the corresponding precondition.

In common sense reasoning, we also have access to a reportoire of 'methods' for how to achieve a goal. The method for achieving a goal is often used in place of the mere attainment of the goal. For example, if we say: 'as John was driving home that afternoon, he was hit by the lightning and died', we refer to the action which, if properly completed, would have the postcondition 'John is at home', but which in this particular occurrence was tragically interrupted.

Similarly, if we watch a movie where the hero is asked to 'please leave the room', and he does so by crashing through the window, the possible entertainment effect is derived from the non—standard way in which the hero achieved the requested result.

These examples will suffice here as indications of how we would like to analyze some natural—language verb phrases in terms of intended world—states, and standard methods for achieving them. But there are also plenty of examples of how verb phrases refer to prevail conditions, namely phrases of the form

'keep' + condition

For example, 'keep the car on the road', 'keep the car at the regulated speed', 'keep the pot slowly boiling', 'keep the audience interested', 'keep all the rooms clean', all show how a lot of common sense phenomena may be understood in terms of qualitative regulators or feed back loops. The formal characterization of such actions would of course refer to the state that is intended to be kept, as a prevail—condition. In such cases the prevail—condition is not merely a prerequisite for doing the essential action, but it ***defines*** the essential action.

It is interesting to notice that this could be an entry point to "naive control theory", which would seem to have a potential for being of high industrial relevance. Also, we should maybe now return to ~~von Neumann~~'s early insight that feedback systems are of outmost importance for intelligent behavior, and blow new life into the term 'cybernetics' that he coined.

Norbert Wiener's

✓

10. Non-flat feature domains.

Above we have introduced feature—values as domains, but all examples have been chosen from the trivial case of flat, finite domains. It is easy to see how the more general case can be useful especially for prevail conditions. For example, suppose we have actions for painting a wall with color X, for different specific X, and we also have an action of photographing a white statue with that wall as background, which (in the prevail condition of the action) requires the wall to be non—white. If now the feature domain is organized so that

u ⊑ non−white ⊑ red ⊑ k

then we can organize our action structure so that the paint−red action is succeeded by the photographing action. At the time−point between those two actions, the incoming post−condition (red) is matched against an outgoing prevail condition (non−white). In order to satisfy the last requirement on coherent action structures, we must add an outgoing Noop action whose prevail condition says that the wall is red. The two outgoing prevail conditions 'red' and 'non−white', are co−dimensional but not equal, but that does not matter − the important thing is that their join ('red') is consistent.

11. Related work.

The theories and languages for concurrent programming address the issue of specifying 'two or more sequential programs that may be executed concurrently as parallell processes' (quoted from the survey article of Andrews and Schneider, [Andr83]). Their goal is therefore different from the goal of the present work, which is to characterize parallell processes in the world outside the computer, but evidently the techniques may sometimes be interchangable.

One of the approaches to concurrent programming is to consider ***cooperating sequential processes*** [Dijk68], i.e. to use a set of sequential programs, equipped with special synchronization operations. That approach may make good sense for concurrent programming, especially in machine−oriented programs, but is not as attractive for describing real−world action structures since there is usually not a good set of 'processors' to write programs for and to synchronize.

Another approach, ***path expressions***, separates the specification of operations from the constraints on execution order [Camp74]. In that respect they can be considered as similar to the approach taken in the present paper, since the action structure does not specify the 'procedure' for performing an operation, but only the allowable orderings. Also, the alignment criterium that was introduced in section 8, can be thought of as a set of path constraints, one for each feature. But the path operators that are used for writing path expressions, such as "," for concurrency and ";" for sequencing, do not easily lend themselves to expressing structures like the one in figure 1. Also, path expressions have not (to our knowledge) developed the counterpart of the precondition/ postcondition /prevail− condition characterization of operations.

A large amount of work has been based on ***modal logic***, both as a tool for concurrent programming, and in A.I. for characterizing structures of actions or events (which is exactly the goal of the present paper). Basically, the 'accessibility relation' that characterizes the Kripke semantics for modal logic [Krip63] is then used as the relation between a world−state and a (or the) succeeding world−state. ***Dynamic logic*** ([Prat76]) allows one to use a collection of such accessibility relations. Each elementary operation (from world−state to world−state) may be one such relation, and relations may be composed algebraically, using operators such as ";" for sequential composition, "union" for parallel composition, and the Kleene star for infinite sequential repetition.

The big problem with that approach, from our point of view, is that world−states are not explicitly named and talked about. The language only allows you to say things like "in the resulting state after first doing a, and then doing b and c in parallell, the proposition P will hold". Consequently, the language can not characterize structures like the one shown in figure 1. Also, it becomes quite difficult (probably impossible) to express the constraint of prevail−conditions, namely that no other, parallell branch is allowed to violate the prevail−condition, basically because one can not characterize what is a parallell branch.

Temporal logic ([Resch71]) uses the temporal ordering of points in time, as the accessibility relation, with modal operators such as:

FA A is true at some future time
GA A will be true at all future times

etc. As far as we can see, temporal logic (as used e.g. by Halpern, Manna and Moszkowski [Halp83]) leads to the same problem as were just discussed for dynamic logic.

Manna and Pnueli [Mann81b] apply temporal logic to the specification of concurrent programs, using the approach of "cooperating sequential processes" which is not well adapted to our goal, for the reasons quoted above.

Yet another approach, which is also frequently called "temporal logic", is to use a many−sorted first−order logic where e.g. 'times', 'intervals', 'states', and 'events' are distinct sorts, and where there are the obvious relations and functions such as

During(i1,i2)
Holds(p,i)

and so on. This approach, which we can call "explicit temporal logic" (to distinguish from "modal temporal logic") has been repeatedly used in A.I. Along with (one interpretation of) the logic programming paradigm, work with this approach is done by defining an ontology, first intuitively and then formally by writing down a large number of axioms in first−order logic. The axioms must of course characterize those sorts and relations. McDermott has done this for one particular ontology, which uses states, times, chronicles, etc. ([McDerm82]). Allen has done a similar work for a different ontology, which treats intervals of time as the basic concept ([Alle81]). A critique of these works, which seems to extend to the approach in general, has been written by Turner ([Turn84]).

Yet another approach, particularly in A.I., has been to extend a temporal logic, of some kind, with additonal constructs which turn it into a ***programming language***. The procedural logic of Georgeff et al ([Geor85]) is a case in point.

Outside the framework of formal logic, early A.I. research on ***planning and problem−solving*** developed methods that have inspired the results in this paper. The handling of preconditions and postconditions builds directly on STRIPS, as has already been discussed. Its successor, the NOAH system ([Sace75]) used a partial order on the actions in the plan, in order not to over−commit itself during the plan−making process.

Also, many "***semantic net***" type enterprises (in the broad sense of the attempts to develop adequate knowledge representations to be used for language understanding, scene recognition, question answering, etc., based on common sense and ad hoc notions) have introduced "nodes", "arcs", etc. for actions or events, and are able to express temporal relations, preconditions, and/or effects of those actions. Too often, of course, the expressiveness of such representations is so great that a formal analysis of what it is they express, is not possible.

In relation to these various approaches, ours can be characterized as an explicit temporal logic, and in that respect it is similar to the approach of McDermott and of Allen. However, we do not tread the usual path of logic, i.e. to define the language, write out axioms, define a semantics, and so on. The structures described above are the ones which would have been used for the semantics, if we had followed the standard path. But we do not see the need for language and axioms, at least not at this point. The purpose of the present paper has been to nail down a minimal set of necessary concepts (a simple ontology for action structures, if you wish), and to characterize the logically admissible action structures.

References.

[Alle81] Allen, J.F. "An interval based representation of temporal knowledge". Proc. 7th IJCAI, 1981, pp. 221–226.

[Andr83] Andrews, Gregory R., and Schneider, Fred B. "Concepts and Notations for Concurrent Programming". Computing Surveys, Vol. 15, No. 1, March 1983.

[Camp74] Campbell, R.H., and Habermann, A.N. "The specification of process synchronization by path expressions". Lecture notes in Computer Science, vol. 16. Springer Verlag, 1974, pp. 89–102.

[Dijk68] Dijkstra, E.W. "Cooperating sequential processes". In F. Genuys (ed), Programming Languages. Academic Press, New York, 1968.

[Geor85] Georgeff, Michael P., Lansky, Amy L., and Bessiere, Pierre "A Procedural Logic". Proc. 9th IJCAI, 1985, pp. 516–523.

[Halp83] Halpern, J., Manna, Z., and Moszkowski, B. "A hardware semantics based on temporal intervals". Proc. 19th ICALP. Springer Lecture Notes in Computer Science, Vol. 54, pp. 278– 292.

[Krip63] Kripke, S. "Semantical considerations on modal logic". Acta Philosophica Fennica, Vol. 16, pp. 83–94.

[Mann81] Manna, Z., and Wolper, P. "Synthesis of Communicating Processes from Temporal Logic Specifications". Proc. of the Workshop on Logics of Programs, Yorktown Heights, NY. Lecture notes in Computer Science, Springer Verlag, 1981.

[Mann81b] Manna, Z. and Pnueli, A. "Verification of concurrent programs: the temporal framework". In: Boyer, R.S. and Moore, J.S. (eds) The correctness problem in computer science, pp. 215–273. Academic Press, New York, 1981.

[May83] May, D., Inmos Ltd., Bristol, U.K. "Occam". SIGPLAN Notices, April 1983. (Occam is a trademark of Inmos Ltd.)

[McDerm82] McDermott, D. "A temporal logic for reasoning about actions and plans". Cognitive Science, Vol. 6, pp. 101–155.

[Pete82] Peterson, J.L. "Petri Net theory and the modeling of systems". Prentice–Hall, Inc., 1982.

[Prat76] Pratt, V.R. "Semantical considerations on Floyd–Hoare logic" Proc. 17th IEEE Symp. on Foundations of Computer Science, pp. 108–121.

[Resch71] Rescher, J. and Urquhart, A. "Temporal logic". Springer Verlag, 1971.

[Sace75] Sacerdoti, E.D. "A structure for plans and behavior". Ph.D. thesis, reprinted by Elsevier North Holland Publishing Co., New York, 1977.

[Tate76] Tate, A. "Project planning using a hierarchic non– linear planner". Univ. of Edinburgh, Dept. of A.I. Research, Report 25.

[Turn84] Turner, Raymond "Logics for artificial intelligence". Ellis Horwood, Ltd., 1984.