Logical relations for describing intermittent and sequential behaviour

Jon Bell

Doc. ref. SD/TR/FR/07: November 18, 2003

1 Introduction

This report presents a possible ordered logic to allow description of intermittent and sequential behaviours for functional modelling of engineered systems for design analysis. The report is intended to fill the gap in the draft paper (Bell 2003b) where the logical relationships between functions (outputs) were not specified, pending further thought and discussion. Since then, Neal and I have met and discussed this and this report is based on our conclusions and thought since. It is intended to spark further discussion of this, before either incorporation into (Bell 2003b) or writing up on its own.

This report ignores some questions abut the language that have been raised before (such as the importance of NOT) and also modelling of time. All it is concerned with is modelling intermittent behaviours either in or out of some ordered sequence.

The following section discusses the background for the proposed approach, which is then introduced along with a possible alternative. Its use for describing intermittent behaviour is then illustrated and finally the report will discuss some questions that arise.

2 Background

The hierarchical functional language already includes AND, OR, XOR and NOT, though AutoSteve leaves out NOT. The relations used in AutoSteve are as follows:-

- AND — all subfunctions must be achieved.
- OR — at least one subfunction must be achieved.
- XOR — one subfunction must be achieved (this assumes binary XOR).
- equivalent ("temporal OR") — achieved if any subfunction is achieved at any point point during the simulation.
- strict sequence — subfunctions must be achieved in order defined.
- loose sequence — subfunctions must be achieved but in any order.

These are discussed in more detail in (Bell 2003a). The relationships between the temporal relations seem to be slightly unclear. One of the intentions of the language proposed herein is to define these relationships more closely.

The implementation was flawed (though this might have been corrected) and for FMEA (at least), the simulation was only checked at the end (once a steady state has been reached) so the sequence relations cannot be relied upon. This checking of significant states (or outputs) in terms of function at the end of a simulation step is, of course, adequate if constant behaviours are all that are regarded as significant. In other words, it is acceptable to consider the simulation step, between the input event changing the state of the system and the system settling into a new steady state, as occupying a single time slot. However, if it is required to specify intermittent behaviours as significant, then it is necessary to subdivide the simulation step into a sufficient number of time slots to ensure that each stage of the intermittent behaviour is achieved correctly.

In (Gerevini and Schubert 1995) there is a list of thirteen basic relations in their "interval algebra", shown in figure 1. I am a bit uneasy about the apparently causal nature of their

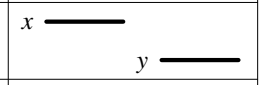
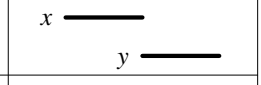
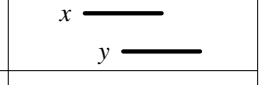
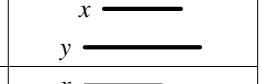
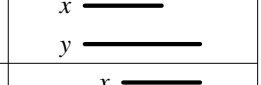
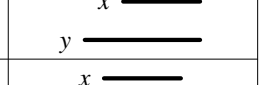
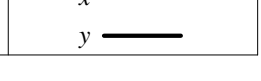
Relation	Meaning
x before *y* *y* after *x*	
x meets *y* *y* met-by *x*	
x overlaps *y* *y* overlapped-by *x*	
x during *y* *y* contains *x*	
x starts *y* *y* started-by *x*	
x finishes *y* *y* finished-by *x*	
x equal *y*	

Figure 1: Temporal relations in the interval algebra, from Gerevini and Schubert

relations. This will be discussed later in Section 4.

In addition to these quite clearly defined temporal relations, in which the relative times of both the start and end of the individual outputs are specified, there is also the possibility that it is unnecessary to define these relations, so that a function does not fail because of ordering of some of the transitions between outputs states. These relations are illustrated in figure 2. It seems sensible to suggest that an ordered logic should be able to model these cases. Are they sufficient?

3 The proposed approach

Any intermittent behaviour can (arguably) be viewed as a sequence of states of the system (characterised by presence or absence of some significant output) and transitions between these states. These transitions are how the simulation step time slot is subdivided into intermittent state based time slots. The operators are therefore also divided into state based ones (that have no temporal aspect) and temporal ones, concerned with ordering of transitions entering or leaving significant system states.

Relation	Meaning	Interpretation
x unordered-with *y*	*x* ? *y*	both *x* and *y* must occur at some stage in the simulation step
x must-overlap y	*x* ? ? *y*	There must be an overlap between *x* and *y* but either can start or end first
x starts-with *y*	*x* ? *y*	both *x* and *y* must start together but the ordering of the ends is unspecified
x ends-with *y*	*x* ? *y*	both *x* and *y* must end together but the ordering of the starts is unspecified
x starts-before *y*	*x* ? *y*	*x* must start before *y* but the ordering of the ends is unspecified
x ends-before *y*	*x* ? *y*	*x* must end before *y* but the ordering of the starts is unspecified
x starts-before *y* and *x* ends-before *y*	*x* ? *y*	*x* must start before *y* starts and end before *y* ends but can end before or after *y* starts

Figure 2: Interval relations that are only partly defined

The state based operators are the straightforward logical ones — AND, OR, XOR and NOT. If these operators are used without any temporal operators (to be introduced shortly) then it will be sufficient to examine the state of the system at the end of the simulation step (once a steady sate is reached) and seeing which outputs (or internal sates) are found and relating them the the functional model. This is as AutoSteve. The simulation can be thought of as having one time slot, which ends once the simulation step finishes. This is illustrated in figure 3. It

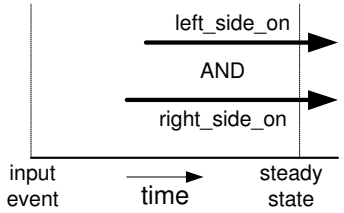


Figure 3: The simulation step as one time slot

will be noted that this is consistent with the current implementation of these relationships and also that as there is no temporal aspect, it does not matter which of two outputs combined by AND is achieved first, or whether they are achieved simultaneously. If it is desired to specify this ordering of transitions into the required output states, then a temporal operator must be used.

If there is an intermittent behaviour to be modelled, then it is necessary to divide the simulation step into several time slots and the state of the system checked in each slot, as illustrated in figure 4. In this case, then, we need to use operators to indicate the starts and ends of intermediate time slots. Note that these are concerned with state of the system, so that the intermittent behaviour in figure 4 can be thought of as "In response to the input, the system will start with the warning lamp off and the horn silent, then the lamp will light and the horn

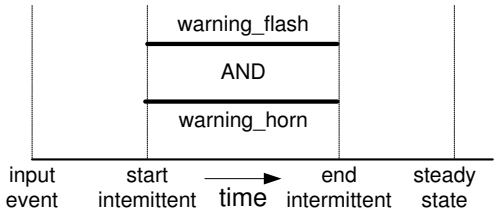


Figure 4: Dividing the simulation step into several time slots

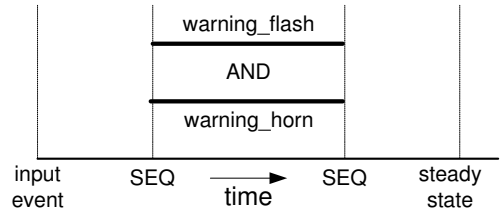
sound and finally the lamp will go off and horn stop, when the system settles until the next input".

We actually need two such operators – one for sequences that end with the system waiting in a steady state (as above) and one where the system enters a cycle of intermittent states which continues until some other input event causes it to stop.

I have used the names

- SEQ — True if preceding state evaluates to true and following state evaluates to true with no intermediate states. In practice it might be more useful for us to regard the condition as true if there are no intermediate states that take any time, so it will be true if the switch between the preceding and following states is instantaneous. As we will evaluate this purely in terms of function, such intermediate states will be lost anyway.
- CYCLE — As SEQ but after last state in sequence of CYCLE time slots, system will return to state preceding first CYCLE operator in sequence. This means that the cycle will continue indefinitely until some external input causes the behaviour to change.

Figure 5 illustrates the use of SEQ while figure 6 shows how CYCLE differs — there is no specified end to the sequence of outputs. Both of these relations have the later interval immediately

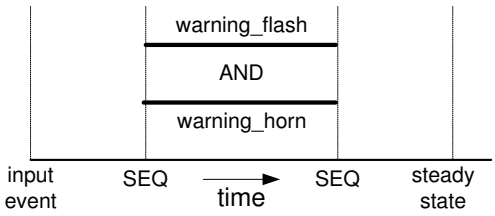


(NOT warning_flash AND NOT warning_horn) SEQ
(warning_flash AND warning_horn) SEQ
(NOT warning_flash AND NOT warning_horn)

Figure 5: SEQ is used to describe a finite sequence of outputs

following the earlier one, so in the case of the behaviour illustrated in figure 5 both the warning flash and horn should start simultaneously. It will be appreciated that cases where one output should precede the other can be described using the same relations. Indeed it is sufficient to allow all the cases illustrated in figure 1 to be described, as shown in figure 7.

However, as suggested in section 2 above, it might be the case that it is unnecessary to specify the precise relationship between the starting (or stopping) of different outputs, as illustrated in



(NOT warning_flash AND NOT warning_horn) SEQ
(warning_flash AND warning_horn) SEQ
(NOT warning_flash AND NOT warning_horn)

Figure 6: CYCLE describes an ongoing sequence of outputs

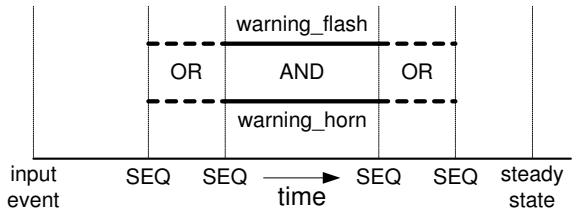
Ordering	**Logical description**
x *y*	(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (NOT *x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x *y*	(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x *y*	(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x *y*	(NOT *x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x *y*	(NOT *x* AND NOT *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x *y*	(NOT *x* AND NOT *y*) SEQ (NOT *x* AND *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND NOT *y*)
x *y*	(NOT *x* AND NOT *y*) SEQ (*x* AND *y*) SEQ (NOT *x* AND NOT *y*)

Figure 7: Using the suggested language for defined temporal relations

figure 2. Where there is to be more than one output but these need not start simultaneously logical OR can be used, as shown in figure 8. It will be appreciated that using OR to specify allowable intermediate states in cases where there are several outputs and several alternative outputs states (such as a motor running slow or fast) then the use of OR might become somewhat cumbersome if it is unnecessary to specify these intermediate states. For this reason, additional sequence and cycle operators are suggested. These specify that the state following the operator must follow the state preceding the operator but need not do so immediately, so the relation is true even if there are (undefined) intermediate states, such as one of the required outputs being achieved before the other. These operators have been named

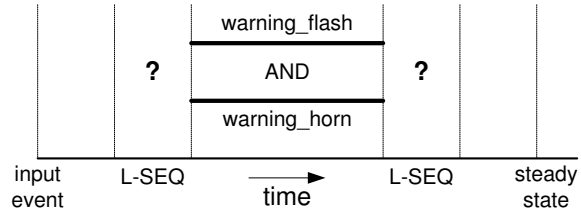
- L-SEQ — Second state to follow previous state but transition may take more than one time slot, so system may have (undefined) intermediate states.
- L-CYCLE — As L-SEQ but returning to start of sequence, as CYCLE.

These leave the intermediate states undefined,as shown in figure 9. This does, of course, differ



(NOT warning_flash AND NOT warning_horn) SEQ
(warning_flash OR warning_horn) SEQ
(warning_flash AND warning_horn) SEQ
(NOT warning_flash OR NOT warning_horn) SEQ
(NOT warning_flash AND NOTwarning_horn)

Figure 8: Using OR to specify that outputs need not start simultaneously



(NOT warning_flash AND NOT warning_horn) L-SEQ
(warning_flash AND warning_horn) L-SEQ
(NOT warning_flash AND NOT warning_horn)

Figure 9: Using L-SEQ to allow incompletely defined ordering

from the use of OR, especially in cases where there are other possible component states. If the allowable intermediate states are to be defined, then OR can be used, as suggested above. This has the advantage of making the model builder specify the desired intermediate states, so any that might not be tolerable can be disallowed. For example, if one lamp is to switch on and another switch off, but the change over need not be instantaneous but one or other must be on the whole time (no interval with no light output), this can be specified with A.lit AND NOT B.lit SEQ A.lit OR B.lit SEQ NOT A.lit AND B.lit. Note that the second time slot need not result in any change as the final state in this sequence is allowed in A.lit OR B.lit.

This means that L-SEQ and L-CYCLE are actually unnecessary except (possibly) for specifying cases where literally anything goes between the two specified states (so that a period with no light emitted is allowed). The relationship between specifying such transitions using OR and L-SEQ is arguably more complex once outputs have intermediate states. This is discussed later.

L-SEQ can be used for all the incompletely defined temporal relations illustrated in figure 2. The use of L-SEQ in these cases is as shown in figure 10. If it is required to define what intermediate states are allowable, OR can be combined with SEQ to describe all these temporal relationships, except the first.

L-SEQ is not used in the first case, where there are two intermittent outputs whose order is undefined. Instead AND is used to specify that both sequences are achieved but as all that this implies is that at some point during the simulation step each sequence's associated output

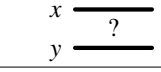
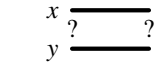
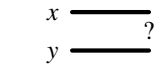
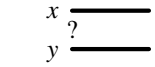
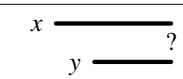
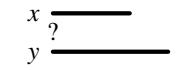
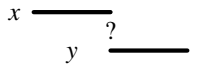
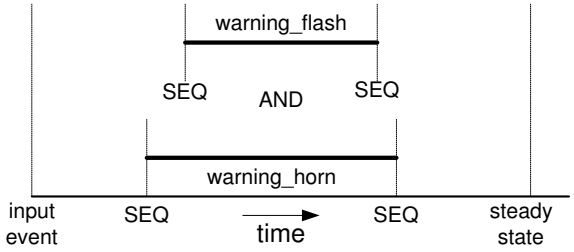
Ordering	Logical description
	(NOT *x* SEQ *x* SEQ NOT *x*) AND (NOT *y* SEQ *y* SEQ NOT *y*)
	(NOT *x* AND NOT *y*) L-SEQ (*x* AND *y*) L-SEQ (NOT *x* AND NOT *y*)
	(NOT *x* AND NOT *y*) SEQ (*x* AND *y*) L-SEQ (NOT *x* AND NOT *y*)
	(NOT *x* AND NOT *y*) L-SEQ (*x* AND *y*) SEQ (NOT *x* AND NOT *y*)
	(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) SEQ (*x* AND *y*) L-SEQ (NOT *x* AND NOT *y*)
	(NOT *x* AND NOT *y*) L-SEQ (*x* AND *y*) SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)
	(NOT *x* AND NOT *y*) SEQ (*x* AND NOT *y*) L-SEQ (NOT *x* AND *y*) SEQ (NOT *x* AND NOT *y*)

Figure 10: Describing interval relations that are only partly defined

be found there is no ordering of the two outputs, all AND implies is that at the end of the simulation step, each sequence will be in its final (completed state) As the time slots are local to a sequence, those associated with each sequence can be in any order relative to the other sequence, the results being that either output can occur first, with or without an overlap. This is shown in figure 11.



(NOT warning_flash SEQ warning_flash SEQ NOT warning_flash) AND
(NOT warning_horn SEQ warning_horn SEQ NOT warning_horn)

Figure 11: Two sequences mean no ordering of intermediate time slots

L-CYCLE has not been illustrated but it bears exactly the relationship to CYCLE as L-SEQ does to SEQ. Therefore the specified states need not follow each other immediately (as in L-SEQ) and the cycle will continue until some external stimulus causes the behaviour to change, as in CYCLE. This raises the question of whether the undefined intermediate states should be the same as the corresponding ones from the previous iteration through the cycle. As they are undefined, it seems better to suggest they need not.

3.1 Temporal logical relations

It seems worth briefly discussing the relationship between the approach suggested above with the use of temporal logical operators:-

- temporal AND — both subfunctions must occur simultaneously.
- temporal OR — one or other function must occur, or both, in which case they must be simultaneous.
- temporal XOR — One or the other subfunction must occur. It will be seen that binary temporal XOR is useless, but as ternary XOR is true if all three subfunctions are true it might be necessary to specify that they are simultaneous. It seems highly unlikely that this will occur in practice.

There seem to me to be one or two difficulties with this approach. If two subfunctions are combined using temporal AND then they are simultaneous. Does that mean the states are entered simultaneously, left simultaneously or both? If we wish to specify that two outputs start simultaneously but need not finish simultaneously, then it is necessary to define temporal AND as being concerned with the entry to the required states. This is possible, of course, but it seems neater to apply temporal constraints to transitions (events) than to states. If we define temporal AND, temporal OR and temporal XOR as being concerned with the timing of entry to the states, this approach can be used to model the same set of sequential behaviours as the use of both strict and loose sequences. In this approach, there is no need for a strict sequence as if two states are to be entered simultaneously, they are combined using temporal AND. Sequence can therefore allow intermediate states and time slots.

Using these temporal relations cannot replace SEQ and CYCLE for two reasons. The first is that these relations do not describe sequences of states (outputs), rather they impose a temporal aspect to the existing logical relations. This means that they cannot be used to describe behaviour consisting of one intermittent output, for example. The second difficulty is related to this. It is that there is no way of distinguishing between a sequence and a cycle. As the temporal relations fail to describe a sequence, this is inevitable.

It might be thought that the use of strict and loose sequences does not allow correct modelling of temporal OR as defined above, but it does, as no intermediate time slots (that is changes of system state) are allowed with SEQ (strict sequence) then if both outputs for OR are to be achieved they must be achieved at the same time.

4 Outstanding questions

This section will briefly discuss questions relating to the proposed language, so suggesting limitations and addressing some of them.

The examples all have functions that map to the presence or absence of a specified output. This is the simplest case, but the existence of more complex component output states (such as motor running fast or slow) should not cause any more problems than at present. The bottom of the functional hierarchy will be concerned with mapping function to output and will typically (but not necessarily) map to a specific output state, so a “fast wipe” function will depend on the wiper motor running quickly while a wiping function might map to the motor running quickly OR slowly. As the new features of the proposed language are orthogonal to the functional hierarchy its adoption should have no effect on these relationships.

A more interesting question is how well the language models continuous changes in output. There are two related issues here, how the language helps when modelling system functions that depend on continuous changes to the output, such as a siren signal depending on glissando changes in pitch, and the possibility of specifying physically impossible behaviours, such as instantaneous filling of a reservoir.

As the language is based on the idea of state and transition, it does not seem ideal for modelling behaviour that depends on continuous changes in output, such as the glissando siren. The difficulty could perhaps be worked around in some cases by attaching the complex output to a state, but this is not an adequate answer where there is more than one component involved. For example if the siren's pitch varies in response to change in voltage from a control ECU then the correct signal output will depend on both components (and the connections) and so this behaviour cannot be associated with a sate in either component. In practice, it seems possible that this problem will barely arise as there seems little call to model such behaviour in so great a detail (at least for FMEA).

The physically impossible sequences (such as instantaneous filling) arise, strictly speaking, from careless use of the language. If a reservoir has states empty part-full and full, then opening the valve cannot result in an instantaneous change from empty to full, so the sequence "empty SEQ full" is impossible, as there must be intermediate time slots associated with the starting and finishing of the filling process (the transitions between empty and part-full and between part-full and full). This is one area where the use of L-SEQ helps simplify modelling, of course, but it is a potential pitfall for users of the language for modelling physical systems.

All the examples used have all functions sharing a single external trigger (such as a switch being pushed) but there are cases where a confirmation or warning function will be triggered by the achievement or non-achievement of some other function. The flashing of the direction indicators to confirm remote locking springs to mind as an example. Arguably this is a distinct issue from that discussed herein, but it might be suggested that the addition of operators to specify such relations between functions is possible. Call them TRIGGERS (for a confirmation function) and FAIL-TRIGGERS for a warning.

There is perhaps some related question as to the use of apparently causal relations between outputs (or whatever) in the Interval Algebra of (Gerevini and Schubert 1995), specifically the relations "x starts y" and "x finishes y" and their counterparts for y seem to imply causality, though I suspect that the paper merely sought a brief description of a strictly time (or ordering) based relationship. I have certainly assumed no notion of causality. Indeed the idea that x triggers y seems to convey no unambiguous temporal relationship. It says nothing about when the two outputs should stop. All that is clear is that y cannot start before x.

Another interesting area is the relationship between the suggested language and the idea of capturing temporal constraints on the achievement of system functions. It will be appreciated that placing a time constraint on,say, the dipping of headlamps does not specify the ordering of individual transitions involved in entering this state. It is suggested that the use of SEQ to indicate the presence of intermediate temporal slots and to represent transitions might provide a suitable means of attaching such temporal constraints. It might be argued that the usefulness of this ordering of states depends on some knowledge of timing of the transitions. There is little point in an intermittent output occurring if it is for so short a time that will have no effect, for example. This stuff does therefore seem quite closely related to the timing stuff in (Bell 2003b).

For clarity, I have given each sequential example a known start state. However, in practice the start state will, of course, be the finishing state (or continuing cycle) of the previous simulation step. I do not believe that this will cause any problems. It could perhaps be argued that any problems that arise from this are more associated with our use of the ordered logic (is that a

good term!?) than the logic itself. This being the case there is no need for us to check the system state for the first time slot, as this will be the same sate as the end of the last simulation step. The sequence can therefore start with the first change of state consequent on the simulation step's input.

This leads on to the question of whether we can simplify the specification of successive steps in the sequence. The simplest case might be where two outputs start one before the other in a known order. There are two possible approaches to assuming the state of a sub function. It can be assumed to be in some default state (off, perhaps), though this seems unsafe. The alternative is to have the rule that where an output is unchanged at a transition, it can be omitted from the following condition. For example "side-on SEQ heads-on" would be the same as "sides-on SEQ (sides-on AND heads-on)" While this saves effort in many cases, there are cases where the approach will add complication. It seems sounder to avoid any such possible ambiguities and insist on relations being described in full. This relates closely to the question as to the use of NOT, which needs care and also (for our use) of whether it is necessary to specify outputs that are inimical to correct achievement of a function, or whether it is sufficient to have such outputs associated with unexpected achievement of some other function.

5 Conclusion

The proposed language seems to be sufficient for the cases illustrated, and I foresee no difficulties with the cases discussed in section 4.

While the language is itself (I claim) as simple as possible, it cannot be denied that some of the description for the behaviours in the cases studies are quite long. The use of L-SEQ or L-CYCLE does simplify the descriptions in some cases. It is suggested that if complex behaviour is to be adequately described, then the description must itself be complex.

The most important work related to this is arguably consideration of whether (and how) causality should be specified. This is awkward where a warning function is triggered by failure of an expected function. In the short term I shall search for references to similar temporal logic papers.

References

Bell, J. (2003a). Function and complex behaviour. SoftFMEA document ref. SD/TR/FR/03.

Bell, J. (2003b). Temporal modelling for failure modes effects analysis. Draft of intended paper.

Gerevini, A. and L. Schubert (1995). Efficient algorithms for qualitative reasoning about time. *Artificial Intelligence 74*(2), 207–248.