Dependencies between functions

Jon Bell

Doc. ref. SD/TR/FR/09: July 29, 2004

1 Introduction

Earlier work on functional hierarchy has concentrated on cases where a top level system function is composed of sub-functions that can be combined using logical operators or, additionally, temporal operators to allow representation of intermittent or sequential goal behaviours. Discussions with Neal have suggested that there might be a case for different classes of dependencies between functions, where rather than the various functions outputs being necessary to fulfil one shared purpose (such as both headlamps being dipped to fulfil the purpose of lighting the road ahead without dazzling) there are functions whose outputs fulfil different but related purposes, where the subsidiary function is triggered by achievement or failure of some other function. There are (at least) two classes of such function:-

- Telltale functions whose purpose is to increase the detectability of the achievement or failure of a prime function.
- Fault mitigation functions whose purpose is to allow operation of the system to continue despite the (possibly partial) failure of a prime function.
- Functions that recharge the system ready for a repeat of the main function.

This report will discuss each of these in turn. These will be followed by a brief discussion of any other classes of function that depend on some other function and a brief consideration of causality.

It is intended to stimulate discussion of this aspect of functional modelling to which end, some questions to consider are listed at the end, in lieu of a conclusion.

2 Function as purpose or goal state?

A starting point for this discussion is whether a system function should be thought of in terms of (one of) the purpose(s) of the system or a goal state of the system. At its simplest this need mean little more than a system function being given a different name, such as "road ahead lit fully" instead of "headlamps main beam", where the intended output is similar. However there is a difference when we come to consider secondary functions that are associated with the main ("prime") function. To reuse the same example, if we are to include the modelling of the dashboard telltales in our car lighting system, then the goal state for "headlamps main beam" is left headlamp main beam on, right headlamp main beam on and blue main beam telltale lit. It is, however, surely the case that the telltale does not contribute to the (main) purpose of this function, the road ahead is no less lit because the telltale has failed. In the context of FMEA, the failure of the telltale will hardly warrant the same value for severity as failure of a headlamp.

If functions are thought of in terms of purpose, this distinction becomes clearer. I appreciate that in this case, it is possible to treat the telltale as a separate system as all that is shared is the input. However, this is not the case for other classes of telltale (or warning) function to be discussed in Section 3.

One approach seems to be to regard a function as a mapping between purpose and goal state (expressed in terms of input and output or more strictly as required pre- and post-conditions). For example, the purpose of lighting the road ahead is being correctly fulfilled when the lamp switch is set to headlamps, the dip switch to main beam (I ignore the toggle switch problem from earlier reports) and the two headlamps' full beam filaments are lit. In other words, a function hierarchy can be though of as starting at a specific purpose.

This mapping between purpose and goal state is not without disadvantages, however. One is the increased complexity of functional labelling, though in the sorts of cases to be discussed below this complication is unavoidable and in other cases the complexity issue does not arise. At a system level, however, some might well remain. A car exterior lighting system has several purposes, so will have several functions, when modelled in this way. We might list some of them as in table 1.

Purpose	**Precondition**	**Postcondition**
Car visible	lamp switch side or head	left side on and right side on and left tail on and right tail on
Light road without dazzling	lamp switch head and dip switch dipped	left head dipped and right head dipped
Light road ahead	lamp switch head and dip switch main	left head main and right head main

Table 1: Some functions of a car lighting system

It will be seen that implicit in these functions is the idea that when the road ahead is being lit, so should the car be visible as the two functions share an input. In other words it is not necessary to explicitly state that the sidelights and tail lamps be on when the headlamps are, which is necessary if the goal state is used. This fits in with the idea that the sidelights and tail lamps do not contribute to lighting the road ahead, though it ignores the point that headlamps do contribute to making the car visible. This might reduce the complexity, as these functions can be reused. Instead of having to restate that sidelights and tail lamps are lit for three separate functions, we state that once and the input preconditions are sufficient to show that these lights are expected whenever the headlamps are lit.

These functions are equivalent to a corresponding set expressed in terms of pre- and post-conditions (that is required inputs and goal state). For example the goal sate for the lamp switch in the heads position might read "Car visible AND (Light road without dazzling OR Light road ahead)". This leads to the point that how the functional hierarchy is used is (arguably) affected by the terms in which a function is considered. There is also the related question of whether decomposing a system function in terms of sub functions is redundant. A strict interpretation of the idea of function as a mapping between input and output and / or behaviour and purpose arguably militates against the idea that we assemble, say, a "headlamps main" function from "right headlamp main" and "left headlamp main" as these subfunctions only relate to the purpose of the top level function. There is no additional information in the explanation “Headlamps dipped not achieved because right headlamp dipped is not achieved” that in “Headlamps dipped not achieved because right headlamp not lit”. There is some thought on this question in [2]. This is more complicated in cases when a function depends on more complex behaviour, such as those discussed in [4]. This might be felt to be somewhat pedantic, but there does seem to be a benefit in establishing the ends of a functional decomposition, arguably the more so once other functional dependencies are taken into account.

This is impossible to demonstrate, but I imagine that specifying system function in terms of purpose is more intuitive than in terms of system inputs and outputs. For now I am suggesting as a starting point for functional modelling the idea that a function is expressed as how the system fulfils some specific purpose, represented by required pre- and post-conditions, either of which might be composed of logical and / or temporal relations. This seems consistent with the representation suggested in [3].

3 Telltale functions

The section above has included an example where two functions can be said to be related, if only in so far as they share input, so whenever one is achieved so should the other be. One area where related functions have more complex interdependencies is that of telltale (or warning) functions. The purpose of such functions is to increase the detectability of the state of the prime function with which they are associated. To this end they will typically have some output (post-condition) that is audible or visible to the operator. Possible relationships between such functions and a prime function are listed below. Given that a prime function is achieved when its precondition Pi, its post-condition Po and (possibly) its behaviour Pb are all true, then we might classify telltale functions by which of the prime function’s conditions are used in its preconditions. They are listed in Table 2. Note that it may not be possible to determine the truth of the behaviour

Class	**Precondition**
input telltale	Pi
no input telltale	$\neg Pi$
output telltale	Po
no output telltale	$\neg Po$
behaviour warning	$\neg Pb$
confirmation	$Pi \wedge Po$
failure warning	$Pi \wedge \neg Po$
unexpected warning	$\neg Pi \wedge Po$
off confirmation	$\neg Pi \wedge \neg Po$
limp home warning	$Pi \wedge Po \wedge \neg Pb$
fault tolerant confirm	$Pi \wedge Po \wedge Pb$

Table 2: Telltale and warning functions

except with reference to pre- or post-conditions. By way of further illustration, here are textual description of these telltale functions, with examples, where I can think of one...

Input telltale is triggered whenever the prime function is triggered and should be achieved regardless of the post-condition of the prime function. An example is the main beam telltale on a car dashboard that lights whenever the light switches are set for main beam.

No input telltale is triggered whenever the prime function is not triggered regardless of its post-conditions. I seem to remember we once had a kettle with a light that came on whenever it was plugged in but not switched on, as a poor example.

Output telltale is triggered by the post-condition of the prime function, regardless of its pre-condition, so the prime function might be achieved unexpectedly.

No output telltale is triggered by the failure of the post-condition regardless of its precondition, so the prime function might be failed or simply off.

Behaviour warning is triggered by the failure of the behaviour (presumably according to some detectable internal state) of the prime function, so either misbehaviour is detected or the function is not active.

Confirmation This is triggered by the truth of both the pre- and post-conditions of the prime function, so it confirms that it is active and achieved correctly.

Failure warning is triggered if the preconditions of the prime function are true but the post-conditions are false, so the function has failed.

Unexpected warning is triggered if the prime function's preconditions are false and the post-conditions are true, so that function is achieved unexpectedly.

Off confirmation is triggered if both the pre-and post-conditions of the prime function are false, so the prime function is off.

Limp home warning is triggered if the pre- and post-conditions of the prime function are true but there is a fault in its internal behaviour. An example would be a warning lamp showing that a fault tolerant system was running without some expected sensor data, so there is a loss of redundancy.

Fault tolerant confirm is triggered if preconditions, post-conditions and behaviour are all active as expected. It therefore confirms that a fault tolerant system is working normally, not in a fault tolerant mode. It might be used in safety critical areas in preference to a "limp home warning" as in that case a failure in the warning system might mean loss of safety in the fault tolerant system going undetected whereas a failure of a fault tolerant confirm system will at least make it apparent that there is a failure (either in the main system or the telltale).

There is no need for any other classes of telltale function associated with failure of the prime function's behaviour provided we are willing to accept that the function operating with pre- and post-conditions satisfied implies correct behaviour and conversely that the absence of either pre- or post-conditions implies incorrect behaviour. Given these assumptions, a telltale that the correct behaviour is taking place is identical to the confirmation function (pre- and post-conditions both present) and adding incorrect behaviour to the failure and unexpected warning functions creates conditions that will be satisfied at the same times as the ones without behaviour. A function that has correct pre-condition, correct behaviour and incorrect postcondition will, of course, never be triggered, given these assumptions. Note that the use of behaviour (as opposed to pre- or post-conditions) as a trigger implies that there is some mechanism that detects the loss of some (internal) behaviour. A fault tolerant system detecting the absence of sensor data is a possible case in point.

Of course, no claims can be made for the completeness of this set of telltale functions where the conditions are not atomic. It might be the case that they are triggered either by a specific output state, such as the oven light coming on whenever the heater is on and going off when the heater is off (the oven has reached the right temperature) or because a subset of the prime function's post-conditions is sufficient to trigger the telltale function. Consider, for example, the idea of the car full beam warning being a confirmation function, so it comes on when the headlamps really are on main beam. If its purpose is to warn the driver that oncoming traffic will be dazzled, it might well be intended that it lights whenever one or both headlamps are on main beam, as one headlamp is enough to dazzle oncoming drivers even though the headlamps' prime function is not achieved. This does mean that this collection of telltale functions cannot be regarded as a formally complete set, though they are in the special case that the pre- and post-conditions of the prime function are atomic. They really only serve to illustrate areas where a function is triggered by the achievement of some other function and, of course, in practice even such an informal classification might conceivably be of assistance in defining the specification of these kinds of function in design.

The significant difference between these functions and the prime function, in general, is that they are not simply triggered by a user input to the system. This raises the more general point that the telltale function cannot be evaluated independently of the prime function, except in the case of the simple input triggered ones. It is useless to model the output of the prime function as an external "input property" as the aim of testing these functions is to show that failures of the prime function will be detected correctly. They depend on some mechanism for detecting the state of the prime function and if that function is reduced to a switch then that mechanism will not be simulated.

It should be pointed out that some of these classes of function are included for the sake of completeness. I certainly cannot call to mind examples of all of them.

The representation of function suggested in [3] offers the possibility of using the post conditions ("effects") in the preconditions of a telltale function. There is more an the proposed approach in Section 8. One aspect of these telltale functions that complicates their relationship with the prime function is that their purpose is to increase the detectability of the state of the prime function. This suggests a need for some way of defining this relationship between the functions. Where an RPN value is used, this will naturally affect the value for detection, arguably the prime function requires two alternative values for detection, one for when the telltale is working correctly and another (higher) one for when it is not. This needs more thought.

4 Fault mitigation functions

Another area where functions might relate closely is where the failure of a prime function is mitigated by fault tolerant behaviour. Possible cases are listed in the table in Table 3. These relate to a prime function with preconditions Pi, post-conditions Po, normal behaviour Pb and purpose Pp. In all cases, of course, the purpose of the fault mitigation function is to enable the continued fulfilment of the purpose of the prime function, at least to a limited extent — possibly at a reduced efficiency or for a limited time. This lists three alternative approaches to fault tolerance, have I missed any? It seems worth adding textual descriptions of these classes of fault tolerant function:-

Backup In this case the purpose is (at least) partly fulfilled by some separate system that

Class	**Precondition**	**Postcondition**	**Purpose**
backup output	$Pi \wedge \neg Po$	backup	Pp
fault tolerant	$Pi \wedge \neg Pb$	Po	Pp
limp home	$Pi \wedge \neg Pb$	$\subset Po$	$\subset Pp$

Table 3: Fault tolerant functions

automatically switches in on failure of the main system. An emergency lighting system that lights routes to fire escapes after failure of the building's main lights is an example.

Fault tolerant Here the output of the system is identical to the prime function's output, but there is a loss of behaviour, covered by redundancy within the system. A brake by wire system continuing to work despite loss of a sensor's data is a possible example. It is likely that this will be regarded as a "limp home" mode, because the loss of redundancy means that other system failures might cause dangerous loss of functionality.

Limp home The output of the fault tolerant function is not identical to that of the main system, but is sufficient to allow continued operation of the device. An engine management system failing in such a way that a default value for ignition timing is used instead of a calculated optimal value is an example, as is an anti-lock braking system falling back on braking conventionally following loss of sensor data.

It will be seen that the "limp home warning" function in Table 2 is identical to a confirmation function for the fault tolerant function (its precondition is that of the fault tolerant function combined with that function's postcondition using logical AND). It is likely that this telltale will be the only sign that the system is not working normally, it being needed to warn the operator that the system requires attention before some other component failure causes the system to fail. This is, of course, the problem with that class name.

I have omitted a manual backup system as this can be regarded as independent of the main system. An example might be a reserve parachute with its own rip cord which the parachutist will activate on finding a problem with the main canopy. This can be considered an independent system (though it clearly has a closely related purpose) except in any cases where it is desired to simulate some interlock mechanism that prevents both main and backup systems being active at once. Arguably, the enabling of the backup mechanism is then a backup function of the main system though it does, of course, not itself have any output unless some telltale is associated with it, to both warn the operator that the prime function has failed and to show that the backup can be used.

An interesting case that Neal and I have discussed is a backup power supply (UPS) where on the principal power supply failing a battery takes over for long enough for power to be restored or for the system concerned to be closed down. If this is being analysed on its own then one approach would simply be to model the failure of the main power supply as an input failure (much as though it were a switch being turned off). This is not a good solution as this does not fully model the system within the the UPS that detects the loss of power. In AutoSteve, for example, the main power supply must be represented in such a way that it can be simulated in CIRQ, so that this interaction between the main power supply and the backup is examined at a behavioural level. There seems to me to be room for thought as to how well a design analysis approach that relies on functional modelling for simulation copes with this. If you are going to specify that the function of the detection system is to switch over when power is lost, of course it will work. I imagine that this is avoided by not treating it as an atomic component so its correct functioning is demonstrated by the modelling of the underlying components. As I say, there is room for more thought (and reading!) here.

Another possible example of a backup system that raised a point of some interest was a backup lighting system where emergency lighting comes on if the main power supply fails. There are two possible ways of specifying such a system — either the emergency lights should come on when a failure of the main power supply is detected or (less likely in practice) the emergency lights should come on when the lights are switched on but no lights are lit. These behaviours are different, of course, as in the latter case, the emergency lighting should come on if the lamps in the main lighting circuit all blow. This raised the idea that the functional language might be used to formalise capture of requirements — after it is certainly possible that the requirement for the backup system was originally specified in terms of the switch being on but no light, which might be read as being ambiguous.

5 Recharging functions

One other possibly interesting class of dependent functions is what might be called recharging functions. The rôle of this class of function is to prepare the system for a repeat of the core function. A simple example is a flush toilet. This has two functions that share a trigger, from the user's point of view, that is pulling the chain. One is the flush function, whose purpose is to clear the bowl, the other is the refill function, whose purpose is to refill the tank ready for the next flush. These are not part of the same function as the consequences of failure are different. In other words there is no need for a "pulled chain" function whose goal state is a clean bowl and a full tank. This adds no information to the idea of the two separate functions. Is "Function pulled chain not achieved because flush not achieved" any more informative than simply "Function flush not achieved"? As the function is properly associated with the pulling of the chain as its trigger, then that can readily be added to the explanation.

There is, perhaps, room for discussion as to the relationship between the two functions (prime and recharging). It would be quite possible to combine them and have a flush function with the goal state "bowl cleaned AND water available". It could be argued that in many cases the prime function is the more significant — the recharging function's purpose is to enable a repeat of the prime function. In the case of the toilet, the whole point is that it can be flushed (but surely that implies that after the flush, the whole point is that it can be flushed again!). Equally it seems wrong to suggest that the recharging function is a logical component of the prime function (in the way that the left headlamp being lit is). To illustrate, if some idiot designed a car with separate switches for each headlight, for the road ahead to be lit the driver would have to switch both on. In contrast, if the user of a toilet had to operate a cock to refill the cistern, there would be little difficulty in calling that a distinct function as the two functions' purposes are distinct. One advantage of separating the function, apart from that approach's consistency with the idea of function mapping to a single purpose is that it can guide selection of a test scenario. As the failure of the recharging function is not apparent until a repeat of the prime function, then the distinction between these functions shows the need for the scenario to include two triggers of the prime function.

6 Other consequential functions

One related area is where a function depends (or seems to depend) on the preceding function. An example is the dip switch on my car. Whenever the headlamps are first switched on, they are dipped and then pulling and releasing the dip switch toggles between dipped and main beam. Therefore the effect of the trigger of a new function depends on the previous function. From the functional modelling point of view this is com[plicated by the fact that the achievement or otherwise of the function is irrelevant. It is clearly inappropriate to define the trigger of, say, the main beam function in terms of dipped beam being achieved as then the intended behaviour of the dip switch is (strictly) undefined if the function fails (perhaps because one of the headlamp filaments is blown).

It seems to me to be at least arguable that these cases are really more concerned with the behaviour rather than function of the system. One could finesse the problem in the functional model by having a dip and a main trigger, both of which are attached to the pulling and releasing of the dip switch. This does, of course, fail to capture the switching between goal states.

An alternative approach might be to define the intended behaviour in terms of a sequence of functions, even though this occupies several simulation steps. A state chart would be one way of doing so but it could also be done using the existing language along the lines of

```
(switch.position=HEAD TRIGGERS
   (RIGHT_DIPPED AND LFET_DIPPED)) CYCLE SEQ
(dip_switch PULL SEQ dip_switch RELEASE
   TRIGGERS (RIGHT_MAIN AND LEFT_MAIN)) SEQ
(dip_switch PULL SEQ dip_switch RELEASE
   TRIGGERS (RIGHT_DIPPED AND LEFT_DIPPED))
END_CYCLE
```

The cycle is, of course, broken by switching the headlamps off again. I'm uneasy about this, as clearly this description covers several functions. This could be eased, perhaps, by using incompletely specified subfunctions instead of the effects listed here. This whole area needs more thought. It depends on how the functional description is to be used. This might just be worth doing to describe an intended behaviour as a part of requirements capture. Arguably, for that task a state chart notation would be preferable, however, even if it was actually stored using the functional description language.

7 Causality and function

The idea that function be triggered by achievement or failure of some other function leads towards questions of causality in a more general sense. One point is the possibility that in functional modelling for interpretation of behavioural simulation, there might be no need for any notion of causality. For example, take a simple tank that is to be filled with the inward flow stopping before the tank overflows. Is there any need to specify such a function in any more detail than the idea that the tank is full ("water available" if expressed in terms of purpose, "tank full" in terms of goal state). This is, incidentally, one case where goal state seems more expressive than purpose. There are various ways in which this might be implemented. A float valve is the obvious way, of course, but it would be possible to implement the function using a timer and a pressure sensor on the inlet system. Arguably if a float valve is used, the stopping of the inflow is caused by the filling of the tank while if a timer is used this causal link is somewhat tenuous. Note that if there is a hole in the tank then the float valve version will flow indefinitely while the timer version will stop, but neither will fill the tank. On the other hand, if both emptying and refilling of the tank share a trigger (as in the flush toilet example) then the timed implementation might cause an overflow if the trigger fails to empty the tank. Is this an interesting distinction? Perhaps not. But I suggest that we need not draw such a distinction, anyway. This is because the causal path is specified (implicitly) in the behavioural model of the system, it need not be in the functional (interpretive) model. Therefore the same functional model can be used for either implementation of the tank system.

8 A possible approach

One aspect that most of these functions have in common is greater complexity of input (or triggering input event or precondition). One other approach to modelling function that uses logical relations is in [5], where an axiomatic approach to functional representation is proposed. That paper uses a right pointing arrow (logical "implies"?) to differentiate between the input and output parts of a function. This is (arguably) incorrect, as a strict reading of this means the function's conditions resolve to true whenever the output side resolves to true (whenever the effects are found). This is problematic as in many (most? all?) engineered systems the function will be controlled (i.e. it should be possible to turn it off!). It therefore seems necessary to resolve the overall function (expressed in terms of both input and output) in four ways:-

- Correctly not achieved — the inputs not set and the outputs not found. In logical terms both sides of the function resolve to false.
- Correctly achieved — inputs and outputs all present. So both sides of the logical expression resolve to true.
- Failed — Inputs correct but outputs not found. Input resolves to true but output to false.
- Unexpected — Inputs not correct but outputs found. Input resolves to false and output to true.

Logical AND is possibly usable here but it fails to distinguish between the three different cases where the function is not correctly achieved. Implies only distinguishes between failed and all other cases, so function achieved unexpectedly is lost. I notice that NOT XOR distinguishes between the two failure cases and the two (possibly) correct cases but even here it fails to distinguish between the different natures of the failures.

As a first cut I suggest a new keyword TRIGGERS that divides the input from the output. This allows unambiguous combination of inputs as well as outputs, for cases where either several inputs are combined (such as "light road ahead without dazzling" being "lamp switch.position=HEAD AND dip switch.position=DIP TRIGGERS left headlamp dipped AND right headlamp dipped") and for cases where the input to one function is the output of another (such as the telltale functions discussed above). A possible example might be the ignition telltale staying on if the battery is not being charged. "ignition.position=run AND NOT battery charging TRIGGERS ignition telltale on". It is probably inappropriate to use a truth table for TRIGGERS as the two Boolean values are insufficient to distinguish between the systems functional states, but it might look like Table 4.

It should still be allowable to have the inputs derived from the correct simulation for failure analysis, unless there are telltale functions that are not triggered by that simulation so the failure mode simulation will not be able to distinguish between correct and incorrect behaviour of such functions. Note that many of the telltale functions will need both pre- and post-conditions

Precondition	**Postcondition**	**Function**
false	false	not intended
true	false	failed
false	true	unexpected
true	true	achieved

Table 4: “Truth table” for TRIGGERS

specifying even for failure analysis as the system will not otherwise know when they are correctly achieved. If a telltale is active during the correct analysis and the prime system and telltale both fail during a failure mode simulation, there is no way of telling without explicit knowledge of the telltale's trigger whether it is an “input telltale” (that has itself failed) or a confirmation function working correctly. I wonder if the system could not establish a probable selection of the type of telltale function by analysis of all failure modes in an FMEA but probably not a certain one, as some failures might result in both failing together and some in either one failing.

Another possible aspect of this is the idea of using TRIGGERS to signal the start of a time step for testing timely achievement of functions. At present this needs more thought. There is nothing intrinsically different here from the idea of having an implicit time slot starting with the trigger of the simulation step, ending once the simulation step finishes (with the system in a steady state) described in [4].

I note that the use of triggers allows the representation of sequences of inputs, which suggests that timed inputs could readily be represented as well. For example, the case study in [5], a hand dryer, could have its input represented be “button pressed SEQ button released”, to indicate that the blower should not start until after release of the button. Timing constraints could be added in the same way as is done with outputs. So the belt minder temporary deactivation can be represented by “driver buckled SEQ driver unbuckled (before 3sec)”. This does not, of course, solve the problem of how this scenario is modelled in a simulation that treats each input as an independent event.

Another possible difficulty is modelling the behaviour when one input causes an output to start but maintaining the input for a certain period is required to trigger the timed output. For example, a hand dryer might start on pressing the button but if the button is released immediately the hand dryer stops, only continuing for its timed period of output if the button is held down for a period of time. One rather cumbersome possibility is to model these cases as two separate function, of course. Another is to ignore the immediate release case on the grounds that it does not really fulfil a function (in the sense of purposeful behaviour). This is fine if it really doesn't matter what happens in this case.

9 Questions arising

Rather than a proper conclusion, I shall finish by suggesting possible questions this gibberish suggests.

- Firstly - is it interesting? It suggests a requirement to specify triggers for system functions based on the achievement of some other function, which does look worthwhile.
- Have I missed any classes of function? I believe the set of telltale functions is complete,

but am less sure of the fault tolerant set, which seems to be (even) less formally defined.

- Is there a need to distinguish between function as a relation between behaviour and purpose and function as a relation between input and output?
- How sensible is the idea of the TRIGGERS relation being used both to distinguish between input and output (or, strictly, pre- and post-conditions) and to specify timing constraints? This needs more thought.

9.1 Future work

The main aim of future work arising from this is to arrive at a way of specifying the trigger for system functions that arise from some other function. The TRIGGERS approach looks promising here but this might need more thought. "Prime" functions are specified as being triggered by an input event (such as throwing a switch), either implicitly for failure analyses, where the input is derived from the "correct" simulation or explicitly for design verification. There is material on the problems arising from this in [1].

Another area of future work is the specification of timing, there is earlier work on this, but that predates the SEQ relation, so needs updating.

References

[1] Jon Bell. Functional modelling for SoftFMEA. SoftFMEA document ref. SD/TR/FR/01, 2002.

[2] Jon Bell. Functional decomposition and SoftFMEA. SoftFMEA document ref. SD/TR/FR/04, 2003.

[3] Jon Bell. Representation of function. SoftFMEA doc. ref SD/TR/FR/14, 2004.

[4] Jonathan Bell and Neal A. Snooke. Describing system functions that depend on intermittent and sequential behavior. In *Proceedings 18th International Workshop on Qualitative Reasoning, QR2004*, 2004.

[5] Rasia Loganantharaja. Representation of functional knowledge. In *Reasoning about Function: Workshop Notes of AAAI-93*, pages 102–107. AAAI, 1993.