Hierarchy and function

Jon Bell

Doc. ref. SD/TR/FR/12: July 30, 2004

Abstract

A hierarchical approach to describing function has already been implemented. However, this simple hierarchical modelling gives rise to problems such as representation of partial achievement of function and the possibility of missing unexpected achievement of function in the design analysis.

This report proposes a more fully realised functional hierarchy that is intended to address these problems and also provide a suitable starting point for extending the description language to support description of functions that depend on complex behaviour.

1 Introduction

This piece discusses a possible approach to a hierarchy of function. It is specifically intended to support interpretation of qualitative simulation of engineered systems but it might be worth considering how applicable it is to other reasoning tasks.

The aim is to describe the framework for a logical structure to enable description of system functions that can then be decomposed into individual system properties, such as inputs to or outputs of the system, expressed in terms of properties of specific components.

It is intended to inform the writing of the hierarchy chapter of my thesis (indeed, in an ideal world a good deal of this might be recycled therein!).

It should be read in conjunction with the report on *Representation of function*, (Bell 2004) which contains some material that fills the background to this report and what might otherwise be the introductory material.

1.1 Use of RPN

In FMEA a failure is allocated a Risk Priority Number (RPN) that is the product of values for the failure's severity (the higher the number the more severe) the detectability (the higher the number the more likely that the failure will go undetected) and occurrence (the higher the number the more likely the fault that causes the failure will occur). This report makes some use of the severity value to illustrate the relationship between the subsidiary functions and the main function. The relationship between detectability values is to be discussed and the occurrence value can be ignored here as it is a feature of the component, not the system. Strictly speaking, of course, the probability of a component failure is not independent of the system — the location of a wire will affect the probability of it shorting to power, for example, and the load a system puts a component under might also affect the probability of failure.

The use of severity is less because of its intrinsic importance to the proposed functional representation language (though it will need to be available) than because it seems to be a useful shorthand for illustrating the different possible relationships between a function and its

components. By component is meant a component of the functional model or description (such as the function's post-condition) and not of the system.

1.2 AutoSteve functional models

It seems worth briefly noting my understanding of the nature of the functional hierarchy in AutoSteve, partly to illustrate what I see as possible problems with that implementation and also in hope of establishing what is and is not novel in the proposed version.

A function is either composed of subfunctions or its effect (recogniser) is decomposed by associating it with several outputs in linking the functional model to the system.

In AutoSteve a function is principally decomposed in terms of its effects (outputs). This is inevitable to some extent as for FMEA the triggers are not explicitly represented in the model but are derived from the 'correct' simulation. A typical example might be function main bean composed of left main and right main. The subfunctions (so-called) need not have consequences of their own (either for failure or unexpected achievement) but they cam have. Triggers are only added on mapping the functional model to a specific system and can again (I think) be added at any level in the functional hierarchy.

Effects (subfunctions) are combined using the conventional logical operators AND, OR, XOR and NOT, though there is (arguably) some room for thought as to the need for all of these — AND is by far the most common. One difficulty here is that the language is (strictly) unable to record partial achievement of a function.

Consequences of failure or unexpected achievement of a function are added to any function. This leads to the danger that unexpected outputs are lost as they do not themselves constitute a function.

2 Requirements

This section briefly discusses requirements and characteristics of description of functional decomposition. These include:-

- Possibility of partial achievement of function. There are (arguably) four cases here:-
  - All outputs are required for function to be considered to be achieved. Function is not achieved if any of the outputs fails. An example might be car headlamps, as it is illegal to drive with one headlamp. This is, of course, consistent with using logical AND to combine the outputs.
  - A function is achieved if any of its outputs are present. A possible example might be a room considered to be lit by one or more lamps. This is consistent with logical OR. This is arguably not a useful relationship as partial failures of the function (one of the several room lamps fails) will be missed as the function will still be achieved.
  - A function fails, but only partially, if not all the outputs are present. An example might be a warning system which should both sound a horn and light a lamp. While both are required, it will be appreciated that the partial warning given by one or other without both is better than nothing.
  - A function is achieved, but only partially, provided at least one of its required outputs is achieved. This is clearly similar to OR above, but in this case it is expected that the partial failure will be noted.

It might well be argued that the last two cases are similar, but I suggest that they are not identical. There is also the question of where the boundary between this cases is to be found. For example, it might well be the case that a room can be considered to be adequately lit if one lamp fails but no more.

- Asymmetry between non achievement and unexpected achievement of a function. Given that one headlamp being lit does not achieve a function (as in the first case above) there is a danger that the effect of its being lit unexpectedly is lost (if only as a result of careless model building). It is still the case that its being lit unexpectedly is an effect - it might still dazzle oncoming drivers and might still drain the battery. Therefore the treatment of failure of expected functions and unexpected achievement of function need different approaches.

- In the case of failure analysis, it is possible to omit explicit representation of a function's precondition as this can be derived from the "correct" simulation. However, this appears not to be the case where a function is triggered by achievement (or failure) of some other function. Can we establish a rule for cases where the use of the correct simulation is sufficient?

- Cases where a function is achieved by exactly one of two outputs, that is by XOR. (I leave discussion of ternary XOR!) There are two cases of failure here, the consequences of which are likely to be different, so they need distinguishing. These, of course, where neither output occurs and where both occur together. I suppose there could be cases where two outputs cancel each other out. These cases are hindered by my inability to come up with any case studies that might use XOR for the effect.

- Dependencies between functions. The most typical example will be where a function is triggered by the achievement or failure of another function, but a more subtle case is where a function's consequence affects another function. A case in point is where the detectability of a function is affected by the failure of a warning function.

Each of these requirements will be discussed in its own following section. These will be followed by a section discussing the representation of purpose.

3 An approach to functional decomposition

There seem to be situations where a complete function (consisting of trigger, effect and consequences) might be decomposed in terms of one, some or all of its attributes. At the simplest, a function might be composed of subsidiary functions (each with their own purpose) or simply several required effects that do not themselves achieve any purpose.

A top level function will have a mapping to purpose (expressed in terms of consequences of failure of the function), a trigger (though this might be derived from the correct simulation in failure analysis) and an effect. It is possible that the triggers and effects are actually 'inherited' from subfunctions (though I suggest not the purpose). It might well be the case that a function has several effects (outputs), each of which fulfil a sufficiently distinct part of the function that they can usefully be thought of having their own purposes. However, if they only occur as part of a top level function, they will only be associated with the top level function's trigger. This leads to a need for an 'incomplete function' that lacks its own trigger but has its own effect and purpose. I suggest that anything less than that does not constitute a subfunction. There might also be some use in a different kind of incomplete function that has a trigger, an effect but

no distinct purpose. We could distinguish between these cases by referring to 'purposive' and 'operational' incomplete functions. I shall discuss each of these classes of incomplete function in turn.

3.1 Operational incomplete functions

An operational incomplete function is represented by an identifier (name) and a complete recogniser (consisting of both trigger and effect). It does not have a formal mapping to purpose, this being provided by a function elsewhere in the decomposition. It therefore provides a way of associating individual triggers and effects in a hierarchical function. There is a theoretical requirement for such a function as breaking the triggers and effects of a function down does not give the same result.

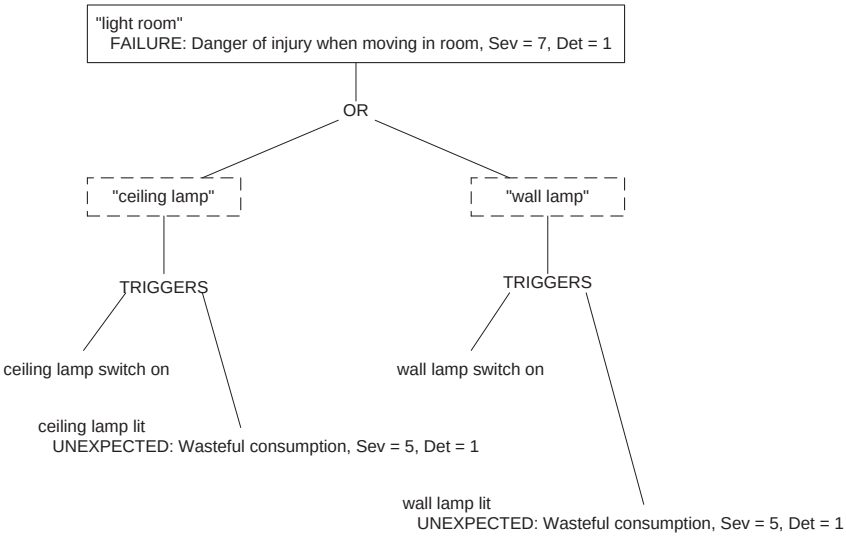
I am unsure how useful the operational incomplete function really is. The best example I have come up with so far is the locking of a car (or house) where the purpose is to secure the car and this is only achieved by locking all the doors. In this case it can be argued that there is no purpose to locking one door without ensuring the others are also locked. From the designer's point of view this is a pretty useless example as the doors are all independent systems (no central locking or the trigger will be shared, of course). All such a functional model does is tell the user that s/he must lock all the doors. This functional model would perhaps suggest to a designer that devising a central locking system where one trigger ensures all the doors are locked is a good idea.

I am reasonably persuaded that whenever an output amounts to an effect, it should be associated with a purpose, even if that purpose is only a subset of a higher level function's purpose. I suspect there are no cases where a function depends on two pairs of triggers and effects, either of which on its own can sensibly be deemed to achieve nothing. Note this is not the same as the case where, say a lock must be released before the effect of the other trigger can occur.

A possible example of the use of operational incomplete functions is illustrated in Figure 1. Here, a room has two lamps (each with its own switch) either of which will allow someone to find their way around the room without colliding with the furniture. However, each trigger is associated with its own effect. I note that one could capture this in one recogniser, along the lines of "ceiling lamps switch on TRIGGERS ceiling lamp lit OR wall lamp switch on TRIGGERS wall lamp lit". This loses the idea that a function is a (set of) triggers triggering a (set of) effects (so TRIGGERS appears once in a function). This restriction on the use of TRIGGERS avoids careless specification of these complex triggerings and means that the four way "truth table" associated with TRIGGERS is kept clear of the conventional logical operators. It also seems possible that later in the design process, these operational incomplete functions can be refined and acquire their own purposes, so they are "promoted" to complete functions. For example, the wall lamp on has the purpose light desk, which becomes clear as the layout of the room is known, but not from the wiring diagram.

3.2 Purposive incomplete functions

An example of a purposive incomplete function might be a warning system where there is to be an audible and a visual warning signal. Each of these is triggered by the same event and while each has its own effect, they clearly both contribute to the top level warning function. However some warning is still given if one fails, and the effect of each is sufficiently distinct to suggest that each has its own clearly defined subset of the top level function's effect and purpose. There is a link here between the idea of such subsidiary functions and partial achievement of the main



POSSIBLE TEXTUAL FORM

FUNCTION "room lit"
FAILURE: Danger of injury when moving in room, Sev = 7, Det = 1
RECOGNISER: "ceiling lamp" OR "wall lamp"

OIF "ceiling lamp"
RECOGNISER: ceiling lamp switch on TRIGGERS ceiling lamp lit [UNEXPECTED: Wasteful consumption, Sev = 5, D = 1]

OIF "wall lamp"
RECOGNISER: wall lamp switch on TRIGGERS wall lamp lit [UNEXPECTED: Wasteful consumption, Sev = 5, D = 1]

Figure 1: Operational incomplete functions

function. In this case if the audible warning occurs without the visual one that is better than nothing. The proposed approach is that such subsidiary functions are used when one being achieved does mitigate the failure of the main function.

One aspect of these incomplete functions is the possibility that such a function might contribute to several higher level functions. One case might be some sort of industrial instrument panel in which one of several plant failures is indicated by a warning hooter but each has its own warning lamp on the monitor panel. The purpose of the hooter is therefore essentially to tell the operator to look at the instrument panel to find which failure has occurred. There are two (equivalent) ways of modelling this. Either the hooter function could be separate with its own complete function (draw attention) and trigger (which would be each failure, combined using logical OR) or it could be a reused purposive incomplete function with no trigger, but instead added to the various warning functions. This latter arrangement seems more correct, as a warning for each failure should correctly combine the two elements - hooter and lamp.

3.3 Different decompositions

The use of these two types of incomplete function suggests the possible decompositions illustrated in Figure 2. Note that wherever triggers or effects appear in the diagram, there might be several combined using logical operators. Also, of course, there is no reason why the decomposition is limited to one layer of subfunction.

It is hoped that the figure is reasonably self explanatory. It is perhaps worth briefly describing the alternative decompositions.

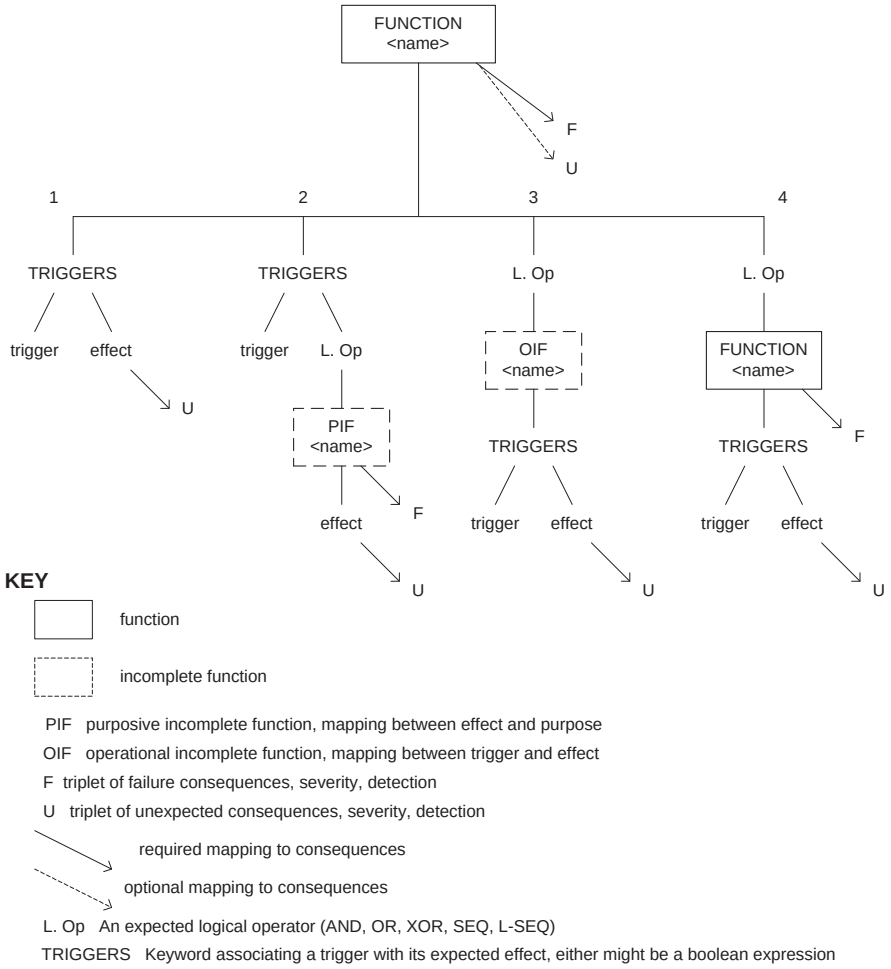


Figure 2: Alternative ways of decomposing a top level function

- The first case is where there is no functional decomposition, a function is composed of a trigger and an effect (either of which may have several parts, combined using logical operators). The dipped beam function is an example.
- In the second case a trigger triggers more than one sets of effects, each with its own distinct purpose (which might mitigate the consequences of the top level function). A simple example might be a "car visible" function whose subfunctions are sidelights on (so car is visible from the front) and tail lamps on (so car is visible from behind). The audible and visual signals of the belt minder system os another example.
- Here a function depends on more than one combination of trigger and effect, but these have no distinct purpose. The room lighting example in Figure 1 is an example.
- In the final case, a top level function is composed of several lower level functions, each with its own trigger effect and purpose. A possible example is a hob where each of the four rings has its own function combined using OR into a top level "cook on hob" function that captures the idea that some failure that stop all the rings from working is more severe than one that only stop one from working.

4 Partial achievement of functions

As noted earlier there does seem to be a difference between cases where if any expected effect is absent the function can be regarded as not achieved and cases where the absence of an effect means the function, while still not achieved, has its failure mitigated by the presence of some of the required effects. In other words we might want to recognise cases where the function is partly achieved. In Figure 3 there are six cases identified, each associated with a different textual interpretation in a resulting design analysis report. It will be appreciated that in each case, were both effects (or subfunctions) missing the top level function would simply not be achieved.

This raises the question of whether all six variations are really necessary. I note especially that the difference between "partial AND" and "partial OR" is very slight, amounting to little more than an optimist saying a glass is half full while a pessimist insists it is half empty. I had the idea that One could distinguish between AND and partial AND (and OR and partial OR) simply by whether or not the operators were associated with purposive functions or with effects. This does, of course, restrict the decompositions to four alternatives. It also is problematic with OR as OR will not result in any notification of a missing effect (the function being achieved). Of course, for failure analysis this difference will be known from comparison of the correct and failure mode simulations, so could be noted in terms of effects. I am tempted to suggest that the best approach might be to include partial OR as being sufficiently distinct from OR with subfunctions but not bother with partial AND. This is partly because of the similarity between partial AND and partial OR and partly because the if the consequences and severity are those of the subfunction, this seems to imply that the top level function is regarded as achieved rather than not. Having said that, if we are to allow for the idea that a function requires two effects (subfunctions) but one is at least a mitigation of the function's non-achievement, it does perhaps make sense to use the lower level function's value for severity. At least the textual interpretation ("function top level not achieved because...") is reasonably consistent with logical AND.

An alternative is to suggest that the full language might include both partial operators, even though partial AND will be rarely used, perhaps specific users of the language might configure the tool support to disable the use of one or both of the partial operators.

One point that arises here is that it seems likely that OR will be rarely used to combine effects, can think of no functions where it really does not matter if one or more or all of the expected effects occur. A corridor with say three lights might have each light being lit combined using OR, but the system should still show that in some circumstances not all the lights come on as expected. There might be a presumption that OR is not used with effects. A tool might support this by giving a message asking for confirmation that this is intended.

4.1 Representation of trigger

One point to be made here is that the problems of partial truth of the effects of a function do not seem to apply to the trigger — it is sensible to regard the trigger as simply true (so the function should be achieved) or false (so it shouldn't be). It seems to be the case that where a function depends on more than one trigger, partial fulfilment of a function's trigger will not result in partial achievement of that function (though it could result in achievement of some other function).

5 Unexpected achievement of function

Part of the problem is that while not every effect associated with a function results in achievement of any function, every effect (inevitably) does have some outcome if achieved unexpectedly. If the output has no effect outside the system, it should not be included as an effect of the function.

There are two possible approaches to this problem. Either the consequences of unexpected achievement of the effect have to be included for every individual effect, that is at the leaves of the decomposition or where unexpected achievement of a function is concerned, all decompositions could possibly be related using OR, so any effect results in (admittedly partial) unexpected achievement of the function.

It is worth noting that in order to avoid unexpected individual effects (outputs) being missed, it is also necessary to avoid the use of AND when mapping between the bottom of the functional decomposition and the actual effect (the output of the behavioural model, output properties as AutoSteve would have it). It is wrong, for example to map a function "headlamps dipped" to the output properties of the two headlamps as if one lamp is lit unexpectedly, the function will not be achieved, so the functional model will fail to detect the effect. Instead, it is necessary to have lower level functions, "left dipped" and "right dipped" if all unexpected effects are to be detected. Again, there is no problem using OR in this way, as either output will result in unexpected achievement of the function, so it will be detected. One possible approach to this might be to attach unexpected consequences to the output property, avoiding the need for the bottom level function. This is what is suggested in (Bell 2004), so each effect has its own consequences of unexpected achievement.

One complication that does arise, however, is the possibility that the consequences of unexpected achievement of several effects are more severe than those of each individual effect. The combination of audible and visual warning signals in (arguably) a case in point in that if both occur unexpectedly, this gives a clear (incorrect) warning. It is therefore necessary that a function can have its own unexpected consequences, which apply if both (all) of its effects occur unexpectedly.

6 Function depends on one of two effects

This is where a function is decomposed using XOR. This case is complicated by the obvious fact that there are now two ways the function can fail — either both effects can occur or neither. I ignore ternary XOR, which I suggest is scarcely relevant, though I suppose it should be considered.

A further difficulty is that while these problems apply to XOR on the effect side it is useful on the trigger side. An obvious example is a domestic landing light with switches at the top and bottom of the stairs. Its functional model will look something like "top switch position on XOR bottom switch position on triggers stairwell lit".

One interesting case where XOR might be used is where there is a backup system to fill (part of) the purpose of the primary system. A possible example is emergency lighting system that comes on if the main lights should be lit but aren't. I do not feel this is a good example, as the emergency lighting trigger will presumably include the failure of the main lights.

XOR does need to be available as it is useful in describing triggers. A room light switched by two two way switches is a case in point.

7 Dependencies between functions

Functional dependencies is one of the areas where new work on functional modelling is proposed. What is meant in this case is principally where a function is triggered by the state of some other function. The simple approach to this is, of course where a function's effects trigger some other function, possibly a warning function or the functioning of a backup system. It is not the intention to discuss this area in great detail, merely to attempt a review of how the idea of one function depending on another (in other than an simple hierarchical sense) fits in with the hierarchical model.

8 Behaviour and function

One area which maybe still needs some thought is how behaviour is mapped to function, how significant behaviours are identified. AutoSteve uses "output properties" for this. Have these been formally written up and if so, where? They sometimes seem to amount to labelling of a goal state (such as a remote locking system being in the state "locked") and sometimes to a way of highlighting some aspect of behaviour that is beyond the scope of the simulation, such as using the idea of a lamp being lit whenever there is current in the filament.

There is some similarity between this second sense of an output property and the "abstract states" that (Keuneke and Allemang 1988) see a need for. In that case, an abstract state is used to describe a system state dependent (principally) on some complex underlying behaviour, such as a cycle. In both this case and the second sense of output property, it could be argued that all that is being done is to side step limitations of the simulator. For example, an "ideal" simulator, with knowledge of temperature, properties of filaments, etc could tell that a lamp will glow when current flows through, so that the goal state (lamp filament being active) needs no further explanation. Similarly in the case of the electric buzzer used as an example in (Keuneke and Allemang 1988) if the simulator had sufficient knowledge of the behaviour of springs and the effects of electromagnetism, and causes of acoustic vibration it could establish that the "reed" will repeatedly close the circuit, and not only do so, but make a noise.

Limitations of simulators' knowledge do seem to suggest a need for such explicit labelling of significant states, either because the effect is in some other domain from the simulator's and / or because of the need to abstract complex underlying behaviour.

Another (possibly rather academic) question that surrounds this is where function (in the sense of purpose expressed in terms of system properties) is separated from behaviour. This seems to depend on an arbitrary choice of the boundaries of the system. If the output or goal state is sufficient to describe the system's purpose (the purpose of a buzzer viewed in isolation is to make a buzzing noise when the button is pressed) then it seems reasonable to think of such a goal state in terms of function, while this becomes more difficult once the behaviour is that of a component within a larger system, so that in the case of the belt minder system, for example, buzzing is merely a requisite post condition for correct achievement of the warning function. This leads on to the idea that a system might be analysed in isolation and the relationships between pre and post conditions be used to model that system as a component in a larger system. Hardly new, I know!

All this does seem to illustrate the idea that the split between function and behaviour is hard to clearly define. Indeed, the split between structure and behaviour (or function where behaviour is not represented) is much the clearest of the divisions between the four classes of knowledge listed in (Chittaro and Kumar 1998) and used in my "grid" diagrams. However, I am reasonably persuaded that there remains a rôle for each of the four classes of knowledge.

References

Bell, J. (2004). Representation of function. SoftFMEA doc. ref SD/TR/FR/14.

Chittaro, L. and A. N. Kumar (1998). Reasoning about function and its applications to engineering. *Artificial Intelligence in Engineering 12*(4), 331.

Keuneke, A. M. and D. Allemang (1988). Understanding devices: Representing dynamic states. Technical Report 88-AK-DYNASTATES, Ohio State University. From the Laboratory for Artificial Intelligence Research, Department of Computer Science.

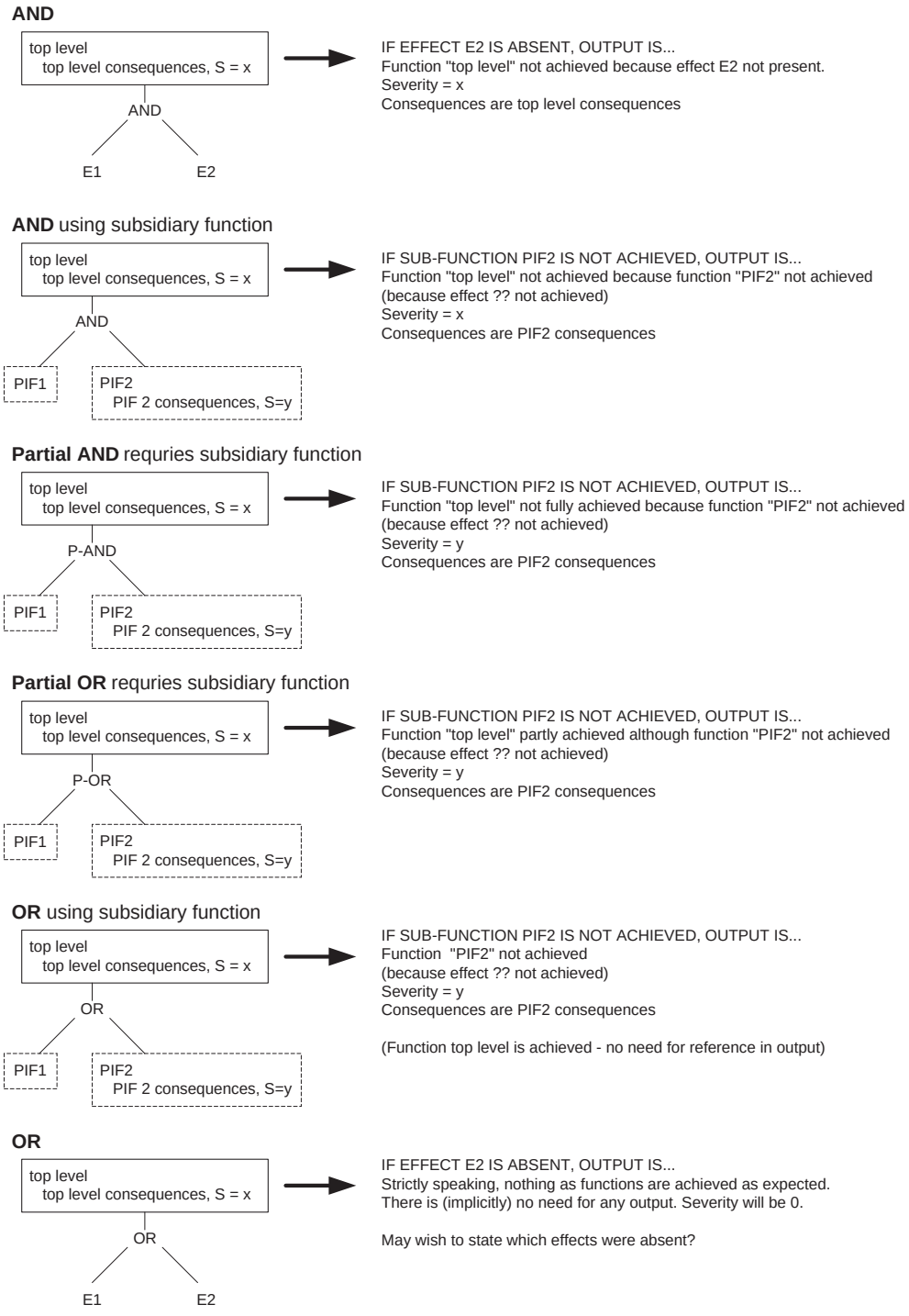


Figure 3: Six degrees of fulfilment of a function