A Translation of State Machines to Temporal Fault Trees

Nidhal Mahmud, Yiannis Papadopoulos, Martin Walker
Department of Computer Science, University of Hull, Hull, United Kingdom
{N.Mahmud@2006, Y.I.Papadopoulos@, Martin.Walker}@hull.ac.uk

Abstract

State Machines (SMs) are increasingly being used to gain a better understanding of the failure behaviour of safety-critical systems. In dependability analysis, SMs are translated to other models, such as Generalized Stochastic Petri Nets (GSPNs) or combinatorial fault trees. The former does not enable qualitative analysis, whereas the second allows it but can lead to inaccurate or erroneous results, because combinatorial fault trees do not capture the temporal semantics expressed by SMs.

In this paper, we discuss the problem and propose a translation of SMs to temporal fault trees using Pandora, a recent technique for introducing temporal logic to fault trees, thus preserving the significance of the temporal sequencing of faults and allowing full qualitative analysis. Since dependability models inform the design of condition monitoring and failure prevention measures, improving the representation and analysis of dynamic effects in such models can have a positive impact on proactive failure avoidance.

1. Introduction

State Machines (SMs) have become one of the prevalent paradigms for modelling dynamic systems. This includes modelling of nominal behaviour but also extends to modelling errors and failures, e.g. as in the HiP-HOPS extension described in [1], in Altarica [2] and the AADL error model [3]. However, although there have been attempts to perform dependability analysis of systems modelled with SMs, there are a number of difficulties. For example, AADL error models – which are effectively state automata showing transitions from normal to degraded and failed states [3] – can be converted to GSPNs and analysed for dependability [4]. Yet in this approach, it is not possible to perform qualitative analysis, i.e. establishment of direct relationships between causes and effects of failure, as in Failure Mode and Effects Analysis (FMEA). Qualitative analysis is particularly important when probabilistic data are not available, e.g. at early stages of design.

An alternative approach involves conversion of state machines to fault trees. This type of approach has been proposed for use with AADL [5]. Fault trees are logical networks of events that show how combinations of failures can cause a given system failure, and are ideally suited for qualitative analysis. But there are difficulties with this type of conversion; in particular, the temporal semantics of the state machine are lost in the translation to fault trees, which have no concept of event order or sequence, and this can potentially cause serious errors. In [2], this type of problem was noted when a conjunction of two mutually exclusive paths through the SM for a k-out-of-n system both became results of the fault tree analysis. This was solved by incorporating NOT gates into the conversion to introduce a "Partial Order", so that different paths through the SM could be distinguished by indicating that some events did not occur. However, although this will work in certain cases (e.g. to prevent the result (A AND B) OR (A AND C) OR (B AND C) when A and B are mutually exclusive), it still cannot distinguish SM paths that differ only in sequence. If, for example, the state machine defines that different sequences of faults, e.g. A before B and B before A, lead the system into two mutually exclusive failure states, the fault trees for these two states will show the combination of A and B as a common cause of both states. This result is logically and probabilistically incorrect, and will thus lead to incorrect conclusions regarding the dependability of the system.

Conversion of SMs to combinatorial fault trees for dependability analysis is being used in conjunction with influential modelling approaches including Altarica and AADL. The potential for erroneous results arising from application of this approach in a subset of SMs must therefore be addressed. To correct the conceptual flaw discussed above and allow true qualitative temporal dependability analysis, in this paper, we outline a technique to enable conversion of
SMs to temporal fault trees, thereby preserving the significance of the sequencing of faults. The fault trees can then be analysed using Pandora, which is a recently proposed temporal logic that enables qualitative analysis of temporal fault trees. The approach is generally applicable to error models expressed as SMs, including the AADL error model. It can also in principle be combined with a compositional analysis technique like HiP-HOPS [1], which would enable automatic synthesis and analysis of temporal fault trees from complex models where the error behaviour of the system has been described by SMs. In section 2, we begin with an explanation of the problems with converting SMs to fault trees and then explain how temporal fault trees can be used to solve the problems. We also introduce a primary-standby system example to help illustrate the significance of the temporal sequencing of faults. In section 3 we demonstrate the conversion process on the example system and then reduce the results obtained to their minimal form using Pandora, making it possible to see the benefits of the process when analysing SMs. Finally, in section 4 we present our conclusions and thoughts on future work.

2. Converting SMs to fault trees

2.1. Combinatorial fault trees

State machines are an expressive, high-level form of modelling the behaviour of systems. They readily express the different effects of events on a system in different states, making them well-suited to modelling the effect of failure and fault events on a system. For example, consider the state machine in Figure 1.a. This SM shows how two different sets of events – A and B on one hand and C and D on the other – can cause different degrees of system failure.

![Figure 1. a. Simple SM, b. More complex SM](image)

This type of SM can be converted into a fault tree for further analysis. A fault tree is a combinatorial model based on Boolean logic, showing the relationships between combinations of basic events (typically component failures) and the system-level failures they cause. In this example, we can see that there are two different system failure states – "Degraded", e.g. a state in which the system can only provide limited functionality, and "Failed", e.g. in which the system can provide no functionality. Each of these failures would become the head of their own fault tree, with the root causes beneath. Here, the causes are simple: the system entering the "Degraded" state is caused by A.B (i.e. A AND B) and the system failing entirely is caused by C.D. These combinations of events are known as the minimal cut sets of the fault tree – the smallest combinations of failures capable of causing the top event of the fault tree (i.e. the system failure) to occur.

However, consider the situation in Figure 1.b. It shows a SM containing two different sequences of the same two failures, each leading to different system failure states. If A happens before B, then the system enters the degraded state with reduced functionality. However, if B happens before A, then the system fails altogether. This is not an uncommon pattern of events; many safety critical systems feature the ability to continue operating with a subset of functionality, e.g. some aeroplanes can continue to fly even after an engine failure. In this case, A may be an engine fire and B the fire extinguishing system; if B fails first, the fire cannot be extinguished and may spread, whereas if A fails first, the fire can be extinguished and the aeroplane can continue to fly, albeit in a degraded state.

By transforming the SM to a combinatorial fault tree, we would obtain a conjunction of the same failures as a common cause for both degraded and failed states: Degraded = A.B and Failed = B.A, and since A.B <=> B.A according to the commutative law of Boolean logic, they are equivalent. The fault tree analysis suggests that the same two failures can lead to two different failure states. This, however, is logically impossible and invalidates application of this approach to this and other cases where the temporal ordering of failures is significant. Note that any consequent quantitative analysis would also be incorrect. Thus, this kind of transformation can lead to serious errors in the analysis: the results are not an accurate representation of the behaviour shown in the SM. The reason for this difficulty is that fault trees have no concept of time or event sequencing; they are purely combinatorial. However, if there was some way of representing event sequences within fault trees, then it would be possible to convert SMs into fault trees without sacrificing the temporal semantics and event sequencing they contain.

2.2. Temporal fault trees and Pandora

There have been a number of approaches that have attempted to extend fault trees with dynamic or
temporal information [6]. Typically these attempts follow one of two approaches: some add additional temporal information to the events (e.g. instead of "failure of X", it becomes "failure of X within 5 seconds of Y"), whereas others attempt to add some form of temporal logic to the structure of the fault tree, typically by adding new types of gates. One of the latter approaches is Pandora, a recently proposed technique for the incorporation of temporal information in fault trees and enabling qualitative analysis of that information [7].

Pandora introduces three new "temporal gates" to overcome the limitations of fault trees regarding the sequencing of events. These are: 1) The Priority-AND (PAND) gate (symbol "<"). The PAND is true if all of its inputs occur in sequence, e.g. A PAND B is true only if A and B both occur and A occurs before B. 2) The Simultaneous-AND (SAND) gate (symbol "&"), which is true if all input events occur at the same time. 3) The Priority-OR (POR) gate (symbol "|"), which is true if either only the first event occurs or if all events occur in sequence, e.g. A POR B is true if A occurs and B does not, or if both A and B occur and A occurs first. It is false if B occurs first or at the same time as A.

The normal precedence of the five gates in Pandora is as follows: OR, AND, POR, PAND, SAND, meaning OR has the lowest precedence and SAND the highest; thus X+Y|Z is equivalent to X+(Y|Z), not (X+Y)|Z. The temporal operators all have higher precedence than the non-temporal ones.

In the same way that Boolean logic has logical laws and truth tables, Pandora has temporal laws and Temporal Truth Tables (TTTs). Some of Pandora's temporal laws are derived from Boolean logic laws, whilst others are new, and all can be proved by means of TTTs in the same way Boolean laws can be proved with normal truth tables. A TTT resembles an ordinary truth table, except instead of true and false, it makes use of integers: zero represents occurrence (i.e. true), and non-zero represents non-occurrence (i.e. false), with the value of the integer representing the order in which the events occur. Thus if A = 1 and B = 2, then both occurred and A occurred first. If two columns of a TTT are the same, then the expressions those columns represent are equivalent.

Temporal expressions are generally complex, particularly when compared with Boolean ones; hence the temporal laws are very useful in order to minimise the expressions and thereby reduce the complexity involved. They also make it possible to perform qualitative analysis: by identifying and removing redundant sequences of events, Pandora can be used to determine the "minimal cut sequences" (MCSQs) of the fault tree: analogous to minimal cut sets, these represent the smallest sequences of events necessary to cause the top event of the fault tree to occur.

2.3. Example system

Figure 2 shows a simple primary-standby (PS) system. Components A and B are abstract representations of any kind of input, control or actuating device, arranged in a redundant series, with A as the primary component and B as the standby. A monitoring sensor, S, detects omission of output from A and activates the standby. "I" represents the input to the system, which feeds each of the two redundant components. "Out" is simply an abstraction of the output of the system. It must receive input from at either A or B for the system as a whole to function and has no failure modes of its own. Performing standard FTA on the system will result in the list of MCS below, suggesting that a failure of the PS system is caused by the occurrence of any one of the following: 1) "Omission of input at I", 2) "Both A and B fail" and 3) "Both A and S fail (as B will not be activated)".

![Figure 2. Primary Standby system](image)

At first glance, all appear to be correct. If there is no input to the system, then it cannot operate; similarly, if the output component of the system fails, then the system cannot function. If both primary and standby components (A and B) fail, then the system will likewise fail as well. #3 is more complex and describes the situation where the sensor fails, meaning it is unable to detect a failure of the monitored component and activate the backup, leading to premature system failure. However, the PS system exhibits dynamic behaviour: its true failure behaviour depends on the chronology of events. The system can function in either of two modes – with A active or B active – and the transition from the former to the latter mode is triggered by an omission failure detected by the monitor S. Different sequences of failure events can lead to the system failing in different ways, and not all of them are correctly represented by the results.

For example, assume that S fails first, then A fails second; in this case, component B will not be activated upon omission of output from A, since the monitor is
not operational and thus unable to detect the omission. However, if S fails after A, then it has no effect on the system: it has already served its purpose and activated B, so unless there are further failures elsewhere, the system will continue to operate. Thus MCS #3 can be seen as unnecessarily pessimistic in this case.

Clearly, standard combinatorial fault trees do not always produce accurate results for dynamic systems like this one. In this example, we have seen that although FTA suggests a failure of both of A and S will cause system failure, a failure of S after A will not cause system failure.

3. Translating SMs to Pandora

3.1. Representing SMs using Pandora logic

Consider again the example state machine in Figure 1.b. The results of a conversion to a combinatorial fault tree would be: Degraded = A.B and Failed = B.A, where A.B \(\iff\) B.A. However, the state machine indicates that the events occur in two different sequences. With Pandora, we can represent these sequences explicitly using the PAND gate: Degraded = A\(\land\)B and Failed = B\(\land\)A. The PAND gate is not commutative, i.e. A\(\land\)B \(\neq\) B\(\land\)A, and thus the importance of the ordering of the events is preserved, each leading to a different system failure.

However, not all state machines require the use of temporal gates when transformed to fault trees. In many cases, standard Boolean logic is adequate to represent the behaviour and perform a correct analysis; this is true of the SM in Figure 1.b, for example. In this case, we are concerned with separate combinations of events, i.e. the two branches have no events in common and each event contributes only to a single end state (and ultimately, contributes to only one top event). Here, a change in the sequence of the events (e.g. D before C instead of C before D) will not lead to a different failure. In the case that D happens first, the system simply stays in the initial state and when C occurs it performs two instantaneous transitions to reach the final failed state; thus the failure behaviour is not sequence-dependent as in the case of the SM in Figure 1.b. Therefore, in this scenario, Boolean logic is sufficient to model the situation unambiguously (Degraded = A.B and Failed = C.D) and the simpler transformation of SMs into fault trees described in [5] is sufficient to obtain an accurate analysis.

The choice between these two approaches is dependent on whether or not the SM has at least one event appearing in more than one path. Typically, if there is at least one event that contributes to the occurrence of more than one system failure, then conversion to temporal fault trees may be needed. It may also be true even if there is an event that is a contributory factor to the occurrence of only one system failure, but as a result of more than one sequence of events. In such cases, an accurate analysis depends upon the correct preservation of the temporal semantics, as different sequences of those shared events – or other events relative to those shared events – may lead to different final states (and thus different system failures).

3.2. Converting the PS example

Figure 3 depicts a state machine presenting a more detailed view of the failure behaviour of the PS example from Figure 2. Note that two paths for a combination A and B have been shown here for clarity, though as will be seen this does not affect the results. Likewise, the state machine shows that the final state "System Failure" is dependent on different sequences of events. For example, the sequence in which S failing before A can lead to system failure is shown, but the failure of S after A does not lead to system failure (as it has no effect). Other paths that are clearly redundant, such as failures of A then S then B, are omitted.

Figure 3. SM for the Primary Standby system

This type of sequence-dependent failure behaviour is clearly visible from the state machine, but cannot easily be analysed in this form. We could convert it to a combinatorial fault tree, but as explained earlier, the temporal semantics would be lost. Instead, we must convert it into a Pandora temporal fault tree. The first step in the process is to identify shared events, i.e. events that are present in more than one path through the state machine. For example, the event "A fails" contributes to multiple paths through this SM: "A fails \(\Rightarrow\) B fails", "B fails \(\Rightarrow\) A fails" and "S fails \(\Rightarrow\) A fails". These sequences of events are all different causes of the same system failure, and since fault trees do not represent states explicitly, only events, it is necessary
to use a temporal fault tree to represent this type of failure behaviour.

To see how the sequence of events affects the conversion process, let us consider three paths, each sufficient on its own to cause the system failure: Path 1 “A fails → B fails”, Path 2 “B fails → A fails” and Path 3 “S fails → A fails”. The event “A fails” (which we shall abbreviate to just “A”, and similarly “B” = “B fails”, “S” = “S fails”, and “I” = “O-I”, i.e. omission of system input) influences the state of the system in all three paths. However, which path is taken from the initial state depends on whether A, B, or S occurs first, assuming no omission of input occurs before. For example, in the case that A occurs first, we are already in path 1 and S does not influence the state of the system; only a subsequent failure of B (or O-I) leads to system failure. However, if S occurs first (leading us into path 3) then a subsequent failure of A is sufficient to cause system failure.

The only way to accurately model this in a fault tree is to use temporal operators. In particular, we want to ensure that the relative ordering of each event that leaves the initial state (e.g. in this case, A, B, and S each leads to a different path depending on which occurs first) is explicitly represented. The main sequence operator in Pandora is the PAND gate, and we can use this to unambiguously distinguish between a path in which A occurs first (i.e. path 1) and a path in which S occurs first (i.e. path 3): Path 1 “A < B” and Path 3 “S < A”. However, the situation is more complex than it first appears, as we also want to express priority – the idea that one event must occur first – without also implying that another event must occur. For example, to distinguish between path 2 and path 3, we need to know whether S or B occurs first, but S and B are not shared between these paths – only A is. Thus to distinguish path 3 from path 2, we need to say that S occurred before B, but only if B occurs; B does not have to occur for path 3 to be completed leading to system failure.

To do this, we can take advantage of Pandora’s POR gate. The POR gate represents this concept of priority – that one event should take priority over others and must occur first, but without specifying that the other events must also occur. Thus we can represent the three paths as follows: Path 1 “(A < B) . (A | S)”, Path 2 “(B < A) . (B | S)” and Path 3 “(S < A) . (S | B)”. Thus path 1 specifies two constraints: firstly that both A and B must occur, with A occurring first, and secondly that if S also occurs at all, it must occur after A. The second constraint represents a situation where S has already served its purpose and activated the backup, in which case its subsequent failure has no effect. Thus, in path 1, both A and B must occur for the system to fail as a whole. The PAND gate alone would not work in this type of situation as it would mean that all three events must occur. Only the POR gate correctly represents the temporal semantics and allows us to preserve the sequences of events in the state machine.

Using two operators like this enables us to accurately represent the temporal semantics of the SM, but it does lead to more complex expressions and thus a more complex translation process. By using Pandora’s temporal laws, it is possible to simplify the generation of these expressions so that they require only one operator – the POR. The law A<B => (A|B)B means that any expression containing a PAND can be converted to use a POR instead, e.g. for Path 1:

\[(A < B) . (A | S) \iff (A | B) . B . (A | S) \iff (A | B | S) . B \iff (A | B | S) . B\]

A second law was also applied here: A|B . A|C <=> A|B|C. Note that A|B|C is not equivalent to A|B|C; after the first event, the others can occur in any order, so A|B|C <=> A|C|B. Thus we can represent the three paths as follows: Path 1 “A | B | S | B”, Path 2 “B | A | S | A” and Path 3 “S | A | B | A”. This form allows us to see both the temporal constraints (represented by the PORs) and the purely combinatorial constraints (represented by ordinary conjunctions).

3.3. Conversion algorithm

A full formal presentation of the proposed algorithm for automatic generation of temporal fault trees from AADL error models or SMs is impossible in the space provided and beyond the scope of this paper which is rather focused on highlighting an important problem and just on outlining a potential solution. A summary of the conversion process with an illustration of how it would work on the PS system follows.

In general, for every final state of the SM, the algorithm generates a fault tree (possibly a temporal one) with that final state as the top event. A backwards traversal is performed, starting with each final state and ending at the starting state, and thus every path between those states becomes a new branch in the fault tree. Initially, only AND gates are used, so that all events in each path are represented as a conjunction. Next, for each visited state that is common to more than one path (known as a “join state”), the algorithm tries to detect whether a temporal operator is required to unambiguously distinguish one path from another. For example, the final state of the PS state machine is “System Failure”, and the join states encountered during backwards traversal are all those visited states with more than one output path for each – thus every
state in our example SM is a join state except the final one.

At this stage, the algorithm examines the events forming the branches so far and checks to see if any are shared. In the example SM, it would find that A is a shared event for join state "A active". Thus to distinguish e.g. path 1 from path 3, the algorithm would determine that the order of A and S is important and would add a temporal constraint to each branch (A\$ for path 1, S\$A for path 3). Similarly, since path 2 and path 3 both emerge from the same join state, it would examine the events that cause transitions away from the join state (i.e. B and S in this case) and add the appropriate constraints to distinguish them, i.e. "S\$B" for path 3 and "B\$S" for path 2. Since the temporal operators are only added later, SMs with no shared events, like the one in Figure 1.a, will have no temporal constraints and will consist only of logical conjunctions, i.e., "Degraded = B.A" and "Failed = D.C". No further refinements are necessary in that case. But in a situation like the one in Figure 1.b, A and B are shared events and the join state is the initial one. Therefore "Degraded = B.A|B = A\$B" and "Failed = A.B|A = B\$A".

The result of the algorithm is a set of (possibly temporal) fault trees, one per final error state, each containing one branch for each path through the SM to the given final state. Each sequence or combination of events in the fault trees will be unambiguous, i.e. any shared events will be part of sequences to ensure that the same sequence cannot lead to the top event of more than one fault tree.

Application of the above algorithm on the SM of the PS yields the following cut sequences (represented in traversed order), which provide an unambiguous description of the different event sequences that can cause the system to fail: 1) “B . A | S | B”, 2) “A . B | S | A”, 3) “A . S | A | B”, 4) “I . A | I | S | B”, 5) “I . B | I | S | A”, 6) “I | S | A | B” and 7) “I . S | I | A | B”. These results show how the sequence of the sensor failure relative to the failure of A, the failure of both primary and standby, and the omission of input alone or relative to other failures are all important in causing system failures.

3.4. Minimisation using Pandora

Although the seven results above are sufficient to unambiguously describe the sequences of events that can lead to system failure, they are not minimal. Just as with the initial results of an ordinary fault tree analysis, they can be reduced to a simplified form to give a clearer view of exactly which events are necessary (and, in this case, in which sequence) to cause system failure.

We can apply Pandora's temporal laws to perform this minimisation. However, before doing so, it is important to note that there is a key difference between standard Pandora and its application in this situation: the absence of simultaneity. State machines do not generally model the simultaneous occurrence of two or more events; indeed, should such a situation arise, it could lead to non-determinism in the state machine. Pandora, by contrast, does include simultaneity, even devoting an operator to modelling it: the SAND (Simultaneous-AND) gate. Many of Pandora's temporal laws use the SAND gate.

Therefore, to apply Pandora to the type of situation modelled by SMs, in which simultaneity is omitted, those laws which include the SAND operator need to be modified to remove it. In particular, the three main laws of Pandora – the Completion Laws, which relate temporal operators to Boolean ones – need to have their SAND terms removed. The three modified Completion Laws are therefore the Conjunctive Law “X . Y <= X/Y + Y/X”, the Disjunctive Law “X + Y <= X[Y + Y[X]” and the Reductive Law “X <= Y<X + X[Y]”. These laws are particularly useful for minimising the results from the conversion algorithm. By applying the Reductive Law, we can both add extra terms to an expression and also reduce multiple cut sequences into one. In this case, cut sequences #4 to #7 can be reduced to just "I" on its own. The other three cut sequences, #1 - #3, can likewise be reduced. Each can be simplified and expanded by the law X<Y . XZ <= X<Z<Y + X<Y[Z] to obtain the following cut sequences: A<$$B, A$$B\$S, B<$$A, B$$A\$S, S<$$B\$A and S$$A\$B. These in turn describe constituents of the 3-event Reductive Law, X<Y <= X<Z<Y + Z<X<Y + X<Y[Z], and can be reduced to even simpler forms. For example, S<$$B\$A + B<$$A reduces to just B<$$A, and similarly for S<$$A. Another law, X[I <= X<Y + X[I, means we can obtain S<$$A<$$B from S<$$A\$B and thus also reduce S$$A\$B + A$$B$$S to just A$$B. One further minimisation is then possible here: we can apply the Conjunctive Law to A$$B and B$$A to obtain A$$B. Thus the final minimal cut sequences are as follows: A$$B, S$$A and I. These minimised cut sequences give an instant understanding of the failure behaviour of the system: it will fail if both A and B fail, if the sensor S fails before A, or if there is no input. By converting the SM to a temporal fault tree and minimising the expressions produced, we have obtained a concise set of results that provides an more accurate view of the system failure behaviour than the results of the standard
4. Conclusion

State machines are an expressive, high-level form of representation well-suited for modelling the nominal and error behaviour of complex dynamic systems. Conversion of SMs to combinatorial fault trees is one of the established methods for qualitative and quantitative dependability prediction of systems, and has been applied to modelling languages like Altarica and AADL. The first contribution of this paper was to highlight an important problem in this method, namely that conversion from SMs to combinatorial fault trees can lead to erroneous results for a sub-set of SMs where the temporal ordering of failure events is significant. The second contribution of the paper is that it has outlined a potential solution by suggesting the conversion of SMs to Pandora temporal fault trees instead, which preserves the temporal semantics of the SM and enables a true qualitative analysis of a dynamic system. Pandora is a recently proposed temporal fault tree analysis technique that is well suited for this purpose as it enables both the representation of sequences of events in a fault tree and also allows the concept of priority to be correctly modelled, i.e. the notion that one event must occur before another, but that the other event does not necessarily have to occur. This is important for distinguishing different paths through the state machine and producing unambiguous fault trees.

The proposed translation algorithm has been implemented and tested on many examples including the above PS system. It detects which event transitions in the state machine must occur in a certain order by analysing the output paths from every join state and determining how each output path can be represented unambiguously using a particular temporal sequence of events. It then produces a series of fault trees – one per final failure state of the SM – and ensures that no sequence or combination of events can cause the top event (i.e. system failure) of more than one fault tree.

We believe that translating SM models to temporal fault trees offers a significant advantage over other approaches that target only combinatorial fault trees for analysis, as it allows improved fault modelling and failure prediction capabilities over non-dynamic approaches. Thorough and detailed dependability models should be used as an input to the design of condition monitoring and failure prevention mechanisms. The proposed approach can improve the representation and analysis of the dynamic effects of the sequencing of faults and propagated errors in such models, and can therefore have a positive impact on the design of proactive failure avoidance measures.

Nevertheless, there remains scope for improvement and further research work. In particular, we aim to investigate how to apply the algorithm to state machines with cyclic loops in them, and we also hope to investigate the possibility of integrating the algorithm as part of a safety analysis tool like HiP-HOPS, which will enable compositional synthesis and analysis of dynamic systems modelled with state machines.

Finally, we are currently examining the potential of incorporating relevant modelling and analysis concepts within the Electronic Architecture and Software Technology Architecture Description Language (EAST-ADL2) – a domain language for the design of embedded automotive systems.

5. Acknowledgements

This work was partly supported by the EU Project ATESTT-2 (Grant 224442).

6. References