Model Based Importance Analysis for Minimal Cut Sets*

Eckard Böde[1], Thomas Peikenkamp[1], Jan Rakow[2], and Samuel Wischmeyer[2]

[1] OFFIS e.V., Oldenburg, Germany
[2] Carl von Ossietzky University, Oldenburg, Germany

Abstract. We show how fault injection together with recent advances in stochastic model checking can be combined to form a crucial ingredient for improving quantitative safety analysis. Based on standard design notations (Statecharts) annotated with fault occurrence distributions we compute to what extent certain fault configurations contribute to the probability of reaching a safety-critical state.

1 Introduction

Today's transportation systems become ever more complex and rely to a large extent on embedded systems. Typically such systems come equipped with sophisticated redundancy and monitoring concepts to achieve a high degree of fault-tolerance. Fault injection, that is the injection of particular failure behaviour into the nominal behaviour of a formal system model, has been proven to be a suitable method to investigate the effectiveness of these fault-tolerance recipes. Based on such an extended system model the ISAAC project [1] developed methods and tools to automatically compute fault trees [2] and extract *Minimal Cut Sets (MCSs)*. Intuitively, a MCS describes a minimal combination (i. e. no sub set of an MCS is still an MCS) of faults that can cause a safety critical situation to occur. In [3] the application of these tools to an actual system from the avionics domain and their integration into the established safety process are presented.

However, in practice such *qualitative* analyses are not sufficient and *quantitative* safety assessment, taking concrete fault occurrence rates (e. g. taken from technical specification) into account, becomes imperative. The key problem for assessing the quantitative impact of safety-critical fault configurations is to factor in the actually possible fault ordering as well as the transient nature of faults that are both deeply integrated in the system and far from being evident. Expensive manual analyses of the concrete system dynamics and fault interplay have to be performed in order to set up accurate stochastic models. But, since

* This work is supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

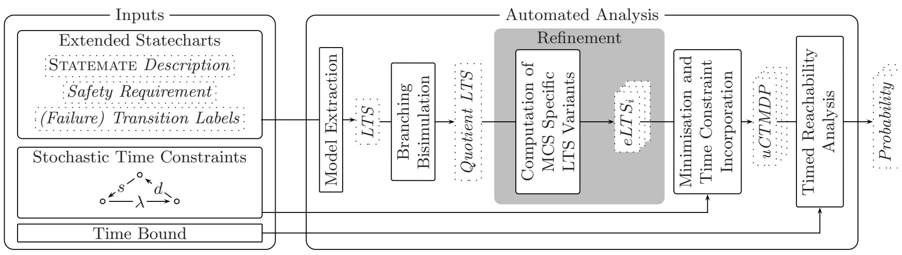

Fig. 1. Timed reachability analysis for STATEMATE - extended tool chain

those analyses are quite time consuming and can usually only be performed by experts, simplified, naive methods are often employed to approximate results. Making use of recent advances in stochastic model checking, in [4,5] a (plug-in) extension of the industrial design tool STATEMATE [6] is presented enabling the automated evaluation of timed reachability properties of the form:

"The probability to enter a safety critical system state within a mission time of 100 hours is at most 10^{-6}*."*

This paper presents a valuable extension of the work presented in [4,5]. Based on a STATEMATE model extended by fault injection and annotated with fault occurrence distributions the probability of reaching a safety critical state is analysed, taking into account *all* faults. Our extension makes it possible to determine the contribution of particular MCSs to the over-all probability and thus to identify those components, whose failing contributes most to reach a safety critical system state. These components should be replaced or improved first in order to improve the over-all system safety, for example if the requirements for certification cannot be met. The benefits of our model-based approach are as follows: The quantitative analysis is automated and can be performed directly on the formal system model. Thus, no extra efforts are required and, as only the actually possible failure sequences are taken into account, more accurate probability measures are derived compared to naive approaches.

We achieve this by encoding path information into the state space of a labelled transition system (LTS). The principal set-up of the tool chain presented in [4,5] and extended by this processing step is depicted in Fig. 1. We will refer to this figure later. A brief overview of the tool chain is given in the next paragraph.

Tool Chain Overview. The over-all approach presented in [4,5] is compositional: *Stochastic Time Constraints*, used to delay firing of particular (failure) transitions, are introduced to the non-stochastic system model in a *Minimisation and Time Constraint Incorporation* step, after the model has been minimised drastically by building the *Branching Bisimulation* quotient. In particular, unique *(Failure) Transition Labels* are used in the *Extended Statechart* formalism to expose relevant transitions by labels in a corresponding LTS. The same labels are used to enrich absorbing continuous Markov chains (CTMCs) with "synchronisation potential" yielding a *Stochastic Time Constraint* [7]. This way, for example, time to failure distributions can be integrated into the system model LTS. Using phase-type approximation [8] the incorporation of arbitrary stochastic delay distributions is possible [4,5]. The stochastic process algebra of Interactive Markov Chains (IMCs) [9] is the key enabler that makes it possible to handle the orthogonal combination of LTSs and CTMCs. The path encoding, we present in this paper, allows for the *Computation of MCS Specific LTS Variants*, retaining the contribution of the faults in a particular minimal cut set, while disregarding contributions of other faults. Note that the time constraint CTMCs are made uniform. This uniformity is preserved during the different processing steps finally yielding a uniform continuous time Markov decision process (CTMDP) [10]. Further note that the *Timed Reachability Analysis* copes with model inherent non-determinism (often used to represent under-specification or to specify an unpredictable environment) by computing the *worst-case probability* to reach a safety critical state within a given *Time Bound* among the existing choices.

Structure. In Sect. 2 we describe the modelling approach, how faults are specified and furthermore introduce an example model that we will extend in Sect. 4 to highlight benefits of our approach. Section 3 shows the construction of cut set specific LTS variants. Concluding remarks are given in Sect. 5.

Related Work. Fault trees (FTs) are used to represent the dependencies of failures and other system events. Based on a FT stochastic models can be developed to establish a quantitative analysis. Dynamic fault-trees (DFT), described in the latest revision of the Fault Tree Handbook [2], are an extension of traditional fault-trees and can be used to describe fault-tolerant systems and also relate the FT to Markov models. The main differences to conventional FTs are the introduction of the new, dynamic gate types *Priority-And (PAND)*, Functional Dependency (FDEP) and *Spare*. With these new gates it becomes possible to express also *sequences of events*. For example only if all sub-events to a *PAND* gate occur in the same order in which they are connected will the gate be activated. The *Spare* gate enables describing several standby configurations where in the case of a fault in one basic component the function will be taken over by another spare component. A spare gate allows sharing of these replacement components between the different branches of the FT and will only propagate the fault if no more spares are available that means all inputs to the spare gate have failed. With these additional gates fault-trees become more expressive and can be used to model systems that could previously not be described adequately. However there are also drawbacks most notably the increased complexity of the formalism for analysis as well as for the construction of the fault-tree although there are some approaches (cf. [11]) that strive to overcome this issue. But another problem of classical fault-tree analysis is still present with this extended approach. Generating a DFT is usually a non-automated approach that requires lots of manual steps and expert knowledge. This is an expensive step in terms of money as well as in terms of time required to build the fault-tree. Keeping the fault-tree in sync with an evolving system design requires even more effort.

2 Extended Statecharts

In this section, following the lines of [5], we introduce *extended Statecharts* as the user-visible formalism to specify behavioural models. We show how an extended Statechart is translated into an LTS and thereby present a concise definition of the formalism. The focus is set on the relevant core needed to clearly expose the syntactical and semantic extensions to the conventional STATEMATE formalism. We present a simple example to illustrate the translation and furthermore the tool chain's principles.

As indicated in Fig. 1, extended Statecharts rest upon two constituents, namely (i) a STATEMATE description of the system under study and (ii) a set of (failure) transition labels. These ingredients determine the semantics of extended Statecharts. Additionally (iii) a safety requirement is used to characterise a subset of system states to be safety critical. For example we identify all the states where *"sensor has failed without being detected"* holds as safety critical. These are the states we are interested in the timed reachability analysis, reaching one of them is also considered as top level event (TLE).

While conventional Statecharts [12] support non-determinism, but no stochastic time aspects, extended Statecharts allow one to refer to particular Statechart transitions by a distinguished set of labels A that are later used as reference anchors to synchronise with appropriately labelled stochastic time constraints. This enables the modelling of stochastic, non-deterministic systems.

Example 1. The extended Statechart of Fig. 2 describes a simple train odometer controller component. It comprises three orthogonal components. The *Monitor* component detects faults in the *Wheelsensor* component, so that, in case of a sensor fault, a brake manoeuvre can be initiated. A third component *Observer* is not part of the system but a means to identify safety critical states.

Depending on the value of a non-deterministic input variable SPEED_FAST, the *Wheelsensor* can reach the node WF representing "sensor failed". As soon as the state WF is entered, the *Monitor* component will detect this and fire the (thin) edge pointing to node BRAKE. Note that initiating this "emergency brake" is only

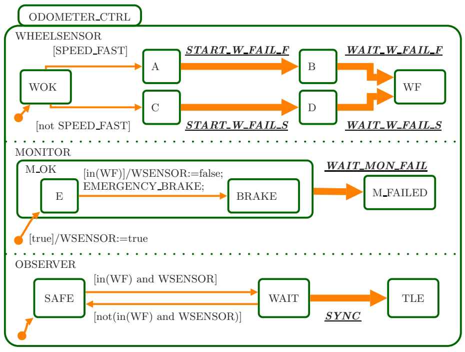

Fig. 2. An extended Statechart: a simple train odometer controller

possible, if the *Monitor* is still working (i. e. in node E). In particular after firing the (bold) edge pointing to node M_FAILED node E will be left and the *Monitor* is no longer sensitive to the guard in(WF) and thus can neither signal the *Wheelsensor* status to other train components (WSENSOR:=false) nor initiate a break manoeuvre (EMERGENCY_BRAKE). The over-all compositional modelling approach allows us to delay the occurrence of the WAIT_MON_FAIL edge using for instance a simple exponential distribution determined by a rate λ. Likewise, the firing of edge WAIT_W_FAIL_F can be delayed w. r. t. START_W_FAIL_F by rate μ. Thus, we can describe arbitrary failure behaviour of a component as part of the Statechart model and control the occurrence of the injected failure behaviour by means of stochastic delays. In addition, it is possible to incorporate delays to describe aspects of the nominal (i. e. non failure related) behaviour.

Statecharts have an intuitive graphical syntax. We vary the corresponding textual syntax as follows.

Definition 1 (Extended Statecharts). *An* extended Statechart $SC = (N, A, V, G, S, E, m, r, d, c)$ *is a 10-tuple, with*

- *N is a finite set of nodes,*
- *A a finite set of action labels,*
- *V a finite set of variables with a (possibly empty) subset I of input variables,*
- *G a finite set of boolean expressions on V,*
- *S a finite set of variable assignment statements,*
- *$E \subset N \times A \,\dot{\cup}\, \{\tau\} \times G \times 2^S \times N$ is a finite set of edges,*
- *$m : N \to \{Basic, Or, And\}$ is a type function, identifying nodes as Basic nodes, Or nodes, or And nodes.*
- *$r \in N$, $m(r) = Or$ is the root node of SC,*
- *$d : \{n : n \in N \wedge m(n) = Or\} \longrightarrow N$ assigns a default node to each node of type Or,*
- *$c : N \to 2^N$ a child relation introducing hierarchy on N.*

Example 2. The nodes WAIT and WHEELSENSOR are of type *Basic* and *Or*, respectively. *And* node ODOMETER_CTRL is the only child of the root node r. The hierarchy determined by c is shown by nesting of states. Here $d(\texttt{Monitor}) = \texttt{E}$. The underlined identifiers in the Statechart define the set A (e. g. $\{\texttt{SYNC}, \texttt{WAIT\_MON\_FAIL}\} \subset A$). Edge $e_{01} = (\texttt{E}, \tau, \texttt{in(WF)}, \{\texttt{WSENSOR:=false;EMERGENCY\_BRAKE;}\}, \texttt{BRAKE})$ is a τ-labelled Statechart edge which we draw as a thin line by convention, while edges $e_{02} = (\texttt{M\_OK}, \texttt{WAIT\_MON\_FAIL}, \texttt{true}, \{\}, \texttt{M\_FAILED})$ and $e_{03} = (\texttt{WAIT}, \texttt{SYNC}, \texttt{true}, \{\}, \texttt{TLE})$ are labelled by elements of A and hence drawn bold. We implicitly define the guard g of such bold edges to be always true.

For the sake of brevity, the above definition omits some well-formedness conditions (cf. [12]) that are unchanged with respect to STATEMATE Statecharts. The substantial extension to conventional Statecharts is the labelling of edges by elements of $A \,\dot{\cup}\, \{\tau\}$. We use the label τ for system internal behaviour, that is for ordinary Statechart edges. Labels in A will be used for synchronisation with "start" and "delay-expired" events of stochastic time-constraints. We emphasise that the behaviour of an extended Statechart is essentially in line with that of conventional STATEMATE Statecharts, except that extended Statecharts allow for a more refined control over which edges are allowed to be fired in orthogonal components within one step. We introduce the following usual notions to determine this semantics. The *scope* $sc(e)$ of an edge $e \in E$ is the most nested Or state that contains the edges nodes. We use $de(n)$ to denote the depth of node $n \in N$ in the node hierarchy c and define $de(SC) = max\big(\{de(n) : n \in N\}\big)$. The *priority* of an edge e is given by its scope distance from the root r. We define the priority relation $e \leq_p e'$, s. t. $e \leq_p e'$ iff $de(SC) - de\big(sc(e)\big) \leq de(SC) - de\big(sc(e')\big)$. Two edges $e, e' \in E$ are orthogonal, denoted $e \perp e'$, iff either $e = e'$ or their scopes are different children of some And node or their descendants. In the example Statechart, it holds $e_{01} \perp e_{03}$ and $e_{01} \not\perp e_{02}$, for example.

Definition 2 (Configurations). *Let $\mathcal{D}$ be the data domain of the variables V. A* configuration *of an extended Statechart SC is a pair $\mathfrak{c} = (M, \sigma) \in \mathfrak{C} \subset 2^N \times \Sigma$, where Σ is the set of all variable valuations $\sigma : V \backslash I \rightarrow \mathfrak{D}$ and M is a set satisfying*
1. *$r \in M$*
2. *$n \in M,\ m(n) = Or$ implies $\exists ! n' \in c(n) :\ n' \in M$*
3. *$n \in M,\ m(n) = And$ implies $\forall n' \in c(n) :\ n' \in M$*

Such a node set M is called a valid node configuration*. We denote $\mathfrak{c}_0$ for the unique initial configuration of Statechart SC, given by an initial valuation of the variables σ_0 and the node configuration determined by d. The set of all configurations of SC is denoted by $\mathfrak{C}$.*

With $dc(M \subset N)$ we refer to the *default completion*, as the smallest superset of node set M, so that $dc(M)$ is a valid node configuration. In particular $dc(M)$ comprises the default node $d(n)$, for all those Or nodes $n \in dc(M)$, that are not already represented by a child node in M. The scope completion $scc(e)$ of edge e is the maximal set of child nodes derived by recursive application of c to the edges scope node $sc(e)$.

Intuitively, configurations comprise all current $Basic$ nodes and their parent nodes (given by inverse of c) and a valuation of the variables V. The state space of the system LTS is defined over such configurations. We will now define the transition relation between configurations and thus define how labels in A affect the semantics of the edges they label. An extended Statechart can be considered as a labelled transition system using the following transition relation.

Definition 3 (Transition Relation). *For extended Statechart SC the transition relation $\longrightarrow \subseteq \mathfrak{C} \times A \,\dot{\cup}\, \tau \times \mathfrak{C}$ is composed of two types of transitions:*
Internal Step. $\mathfrak{c} = (M, \sigma) \xrightarrow{\tau} \mathfrak{c}' = (M', \sigma')$, *iff there exists a maximal set of edges $\mathcal{E} = \big\{e_i : e_{1 \leq i \leq k} = (n_i, a_i, g_i, s_i, n_i') \in E\big\}$ so that*

1. *$\mathcal{E} \subseteq \mathcal{E}_{en} = \big\{e = (n, a, g, s, n') \in E : n \in M$ and g evaluates to true in $\sigma\big\}$.*
2. *$\forall e_i \in \mathcal{E} : a_i = \tau$ and $\forall e_i, e_j \in \mathcal{E} : e_i \perp e_j$.*
3. *$\forall e \in \mathcal{E}_{en} \setminus \mathcal{E}\ \exists e' \in \mathcal{E}$, s. t. $e \not\perp e'$ and $e \leq_p e'$.*

and σ' is obtained from σ by applying the statement sets $s_{1 \leq i \leq k}$ in some permutation on σ and $M' = dc((M \setminus \bigcup_{i=1}^{k} scc(e_i)) \cup \{n_i'\}_{1 \leq i \leq k})$.
External Step. $\mathfrak{c} = (M, \sigma) \xrightarrow{a} \mathfrak{c}' = (M', \sigma')$, *iff*

1. $\nexists e = (n, \tau, g, s, n') \in E : n \in M$ *and* g *evaluates to true in* σ.
2. $\exists e = (n, a, g, s, n') \in E : a \in A$ *and* $n \in M$

and σ' *is obtained from* σ *by applying the statement set* s *on* σ *and* $M' = dc((M \setminus scc(e)) \cup \{n'_i\}_{1 \le i \le n})$.

In a nutshell, the *Internal Step* rule defines conventional Statechart configuration transitions that comprise firing of a maximal set of τ-labelled (thin) edges in orthogonal components and thus implements truly concurrent executions. Instead the *External Step* rule restricts the (bold) labelled edges to be fired in mutual isolation and only if no τ-labelled edge can be taken. Since bold edges relate to time-relevant events, this mutual isolation allows us to recover particular configuration transitions, relative to other transitions. The semantics also gives (non time consuming) internal steps precedence over external steps. This idea of timeless computation is typical for the super-step semantics of STATEMATE Statecharts [12]. We allow hiding of action labels (i. e. replace particular labels in A by τ). This allows us to maintain the effect of external steps without keeping their labels. For example we can hide the label Sync in the example Statechart. However, within this paper, we will keep the Sync label for simplicity. Given an extended Statechart and a set N_{cr} of safety-critical nodes (as specified by a *Safety Requirement*, cf. Fig. 1) we derive an LTS as follows.

Definition 4 (LTS Extraction). *Given a set of safety critical nodes* $N_{cr} \subset N$ *of* extended Statechart $SC = (N, A, V, G, S, E, m, r, d, c)$*, with initial configuration* $\mathfrak{c}_0$*,* SC *can be considered as an LTS* $M = (S^M, Act^M, C^M, T^M, s_0^M)$ *by setting*

- $S^M = \mathfrak{C} \,\dot\cup\, \{\mathfrak{c}_{init}\}$*, the set of all valid configurations in* SC *plus a unique pre-initial state* $\mathfrak{c}_{init}$.
- $Act^M = A \,\dot\cup\, \{\tau\} \,\dot\cup\, \{\textit{INIT}\}$*, the set of labels occurring in the Statechart steps plus a unique label* INIT.
- $C^M = \big\{\mathfrak{c} = (M_\mathfrak{c}, \sigma_\mathfrak{c}) \in S^M : M_\mathfrak{c} \cap N_{cr} \neq \emptyset\big\}$*, the set of safety critical states.*
- $T^M = \big\{(\mathfrak{c}_{init}, \textit{INIT}, \mathfrak{c}_0)\big\} \cup \big\{(\mathfrak{c}, a, \mathfrak{c}') : \mathfrak{c} \xrightarrow{a} \mathfrak{c}'\big\} \subseteq S^M \times Act^M \times S^M$*, the set of transitions possible between the Statechart configurations plus an additional transition* $\mathfrak{c}_{init} \xrightarrow{\textit{INIT}} \mathfrak{c}_0$*, introduced to represent the system start.*
- $s_0^M = \mathfrak{c}_{init}$*, a pre-initial configuration of Statechart* SC.

Note that here we use nodes, such as the node TLE in the example, to identify safety critical states of SC and thus also the set C^M of critical LTS states. Instead of this explicit automata based encoding other specification mechanisms such as temporal logic expressions could also be used.

Example 3. The left part of Fig. 3 shows the LTS that has been constructed from the extended Statechart in Fig. 2 with $N_{cr} = \{\text{TLE}\}$ (defining the top-level event to be an undetected sensor fault). The dashed boxes indicate which states are considered equivalent under branching bisimulation yielding the depicted quotient LTS. We use shortcuts to refer to the labelling in the extended Statechart (e. g. SF stands for START_W_FAIL_F). In Fig. 3 the state $\mathfrak{c}_{01}$ is the only safety critical state. If the Wheelsensor fails (leaving state $\mathfrak{c}_{02}$ by edge WS or state $\mathfrak{c}_{03}$ by edge WF)

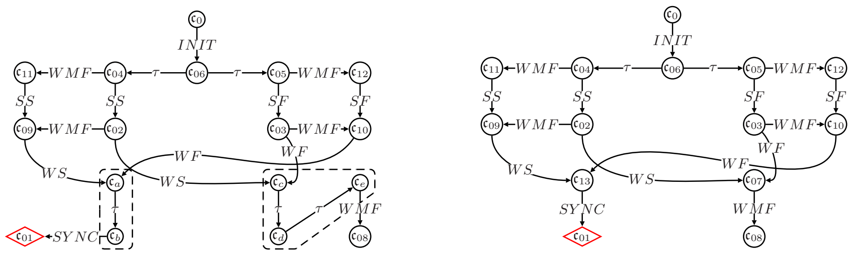

Fig. 3. LTS and quotient LTS for the odometer Statechart

the system goes to a configuration where no safety-critical states can be reached anymore (the `Monitor` detected the sensor fault). If the monitor fault occurs before the `Wheelsensor` fails (e. g. leaving state $\mathfrak{c}_{04}$ or state $\mathfrak{c}_{03}$ by edge `WMF`), then the system will finally enter state $\mathfrak{c}_{01}$. There are two pairs of (failure) sequences that lead to the safety critical state: (i) `INIT`,τ,`WMF`,`SS`,`WS`,`SYNC`;`INIT`,τ,`SS`,`WMF`,`WS`,`SYNC` and (ii) `INIT`,τ,`WMF`,`SF`,`WF`,`SYNC`; `INIT`,τ,`SF`,`WMF`,`WF`,`SYNC`. In each of this sequences `WMF` precedes the label `WS` or `WF`, respectively. The pairs result from the different failure scenarios in the *Wheelsensor* component.

Thus, to actually assess the *quantitative* impact of the faults on the top-level event, we have to distinguish the *paths* where these faults occur. Section 3 shows how we can incorporate the necessary path information into the state space of the investigated system. We complete the specification of the example by providing the time constraints for the extended Statechart of Fig. 2. We furthermore provide the uCTMDP derived for this model.

Example 4. The left part of Fig. 4 shows the time constraint IMCs (cf. Fig. 1) that are incorporated into the LTS of Fig. 3. Each of the three time constraints is derived by a simple absorbing CTMC (drawn grey). These CTMCs are then equipped with a new initial state and labelled edges for synchronisation with the system model LTS. Note that the derived IMCs are made uniform. This uniformity is preserved all along the tool chain. Given these time constraint IMCs and the LTS of Fig. 3 the tool chain presented in [5] computes the uCTMDP depicted in the right part of Fig. 4. Here state s_5 is the critical state as indicated by the self-loop labelled *tle*. State s_3 is a sink state that results from those paths in the LTS that do not finally yield a safety critical system state. In our example, the `Monitor` introduces such a sink by a *safe* shutdown. Non-determinism is present

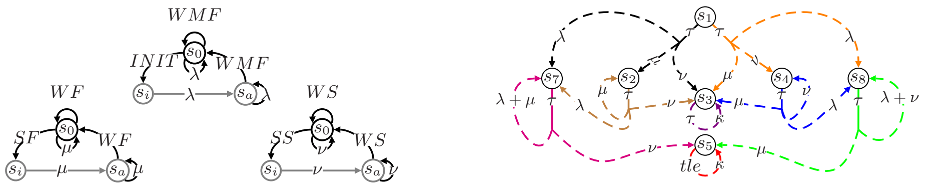

Fig. 4. Time constraints and uCTMDP for the odometer

in the states where more than one τ labelled edge emanates. The forking (dashed drawn) edges represent *races*. For the sake of brevity, we introduce rate $\kappa = \lambda + \nu + \mu$.

3 LTS Transformation

In this section, we show how we encode the context of faults into the state space of an LTS. The basic idea of our approach to establish model based importance analysis for minimal cut sets is as follows. First, given an LTS $M = (S^M, Act^M, T^M, C^M, s_0^M)$, we compute an *enhanced* variant $L = (S^L, Act^M, T^L, C^L, s_0^L)$, by coding label sets describing the path history (starting in the initial state of M) to each particular state in M into the state space of L. Note, that for optimisation purposes we follow [4,5] and use the equivalent quotient LTS, derived by symbolic branching bisimulation minimisation to compute the enhanced variant LTS.

A path $s_0 \xrightarrow{b} s_1 \xrightarrow{\tau} s_2 \xrightarrow{a} s_3 \xrightarrow{\tau} s_4 \xrightarrow{a} s_5$ in M, for example, would yield a new state, described by the pair $(s_5, \{a, b\})$ representing the fact that state s_5 is reachable in M by a path that comprises at least single occurrences of the labels b and a. That is, we do neither code the concrete number of labels nor occurrences of τ transitions into the state space of L, keeping the overall number of states, $O(|S^M| \cdot 2^{|Act^M|})$ in the worst case, in L manageable in size. We formalise this notion of *path history* in the following definition.

Definition 5 (Path History). *Let $M = (S^M, Act^M, T^M, C^M, s_0^M)$ be an LTS.*
Path. *A possibly infinite sequence of transitions*

$$\pi = \big(\pi_i = (s_i, a_i, s_i')\big)_{i\in\mathbb{N}} = \pi_0, \pi_1, ... \in (S^M \times Act^M \times S^M)^* \cup (S^M \times Act^M \times S^M)^\omega$$

is called a path *in M, iff $\forall (s_i, a_i, s_i') \forall (s_j, a_j, s_j')$*

$$\big((s_i, a_i, s_i') \in T^M \;\wedge\; (s_j, a_j, s_j') \in T^M \;\wedge\; j = i+1 \;\rightarrow\; s_j = s_i'\big).$$

We denote π_0 to refer to the first transition in π and π_i for the i-th transition. $src(\pi)$ denotes the source state of π_0, $last(\pi)$ the target state of the last transition in a finite path π.

Path History. *For a given label set $D \subseteq Act^M$ and state $s_0 \in S^M$, we define $\Pi_M^D(s_0) = \big\{\pi : src(\pi) = s_0 \;\wedge\; \big(\forall a \in D \;\exists s_i, s_i' \in S^M : \pi_i = (s_i, a, s_i')\big) \;\wedge\; \pi \text{ is a path in } M \;\big\}$ as the set of D-history paths in M.*

The set $\Pi_M^D(s_0)$ of *D-history paths*, describes all paths in M, starting in s_0, that comprise at least one occurrence of the labels in D. Given an LTS $M = (S^M, Act^M, T^M, C^M, s_0^M)$, we compute the *enhanced* variant LTS $L = (S^L, Act^M, T^L, C^L, s_0^L)$ as follows.

Definition 6 (eLTS). *Let $M = (S^M, Act^M, T^M, C^M, s_0^M)$ be an LTS. We call the LTS $L = \big(S^L \subseteq S^M \times 2^{Act^M}, Act^M, T^L, C^L, (s_0^M, \{\})\big)$* enhanced *$M$, iff*

1. $\big((s, D), a, (s', D \cup \{a\})\big) \in T^L \Leftrightarrow a \neq \tau \wedge (s, a, s') \in T^M \wedge \big(\exists \pi \in \Pi_M^D(s_0) : last(\pi) = s \vee s = s_0 \wedge D = \{\}\big) \wedge s \notin C^M$

2. $\big((s,D),a,(s',D)\big) \in T^L \Leftrightarrow a = \tau$ *and* $(s,a,s') \in T^M \wedge \big(\exists \pi \in \Pi_M^D(s_0) : last(\pi) = s \vee s = s_0 \wedge D = \{\}\big) \wedge s \notin C^M$

3. $C^L = \big\{(s,D) : s \in C^M \wedge \exists \pi \in \Pi_M^D(s_0) : last(\pi) = s\big\}$

While computing L, we preserve the semantics of M w. r. t. the timed reachability analysis, that is, we neither remove nor add paths starting in the initial state of M to the first occurrence of its critical states. In particular, the LTS M and its enhanced variant are obviously strong (and thus also branching) bisimilar [13,14] by construction. The equivalence classes on L are induced by the state-pairs state component, yielding the coarsest branching bisimulation equivalent M (that we originally computed using the symbolic branching bisimulation [15] as described in [4,5][1]). The property of the enhanced LTS to be in a strong bisimilarity relation to the original LTS is sufficient to justify their substitutability for the subsequent analysis (cf. [9], p.73, theorem 4.3.1 and [17]).

Example 5. For our running example, we derive the enhanced LTS of Fig. 5 from the LTS depicted in Fig. 3. This representation comprises the same sequences as the original LTS but allows for a more detailed analysis: The two critical states $(c_{01},\cdot)$ correspond to the different failure scenarios described in example 3. Thus, now these states may be distinguished by the failure sequences that caused them.

In a second step, we establish differentiated timed reachability analysis for L. We only have to shrink the set of critical states C^L to particular subsets dependent on a given minimal cut set $mcs_i \in MCS$, where $MCS = \{mcs_1, \dots, mcs_n\} \subseteq 2^{Act^M}$ denotes the set of all MCSs. In the enhanced variant LTS L for each state $s_{cr} = (s^{cr}, D^{cr}) \in C^L$ label set D^{cr} encodes the concrete failure (label) set that caused the particular safety critical state s_{cr}. As we are interested in the contribution of all failure scenarios (i. e. paths) that require at least the occurrence of the faults in the minimal cut set mcs_i, we analyse cut set specific LTS variants. For an enhanced LTS L we analyse the LTS $L_{mcs_i} = \big(S^L, Act^M, T^L, C_{mcs_i}, (s_0, \{\})\big)$, where $C_{mcs_i} = \big\{(s,D) : (s,D) \in C^L \wedge mcs_i \subseteq D\big\}$. Note that our prototypical implementation of the described LTS transformation is able to extract the set of minimal cut sets MCS by analysing all label sets D^{cr}. Fig. 6 summarises the intuition of the transformation step. Given an LTS (a), the enhanced variant (b) is computed by unfolding the paths in (a). This LTS encodes the paths of the original LTS in a differentiated manner. Here, each of the critical states is related to one particular minimal cut set. The analysis of the two cut set specific variants (c) and (d) using the tool chain back-end (cf. Fig. 1) entails the disregard of minimal cut set $\{a,c\}$ for (c) and $\{a,b\}$ for (d), respectively. One benefit of our model based approach is that only those failure sequences that indeed cause a safety critical state are considered. For example the multiple occurrence of fault c. In contrast the *naive* approach, frequently used in practice, to determine the cut set specific measures would be to multiply the single failure probabilities. For cut set $\{a,c\}$, given

[1] As detailed in [4,5] we consider safety-critical states and non safety-critical states non-bisimilar *by definition.* A more detailed discussion of the related issue of state vs. transition labelling in IMCs can be found in [16].

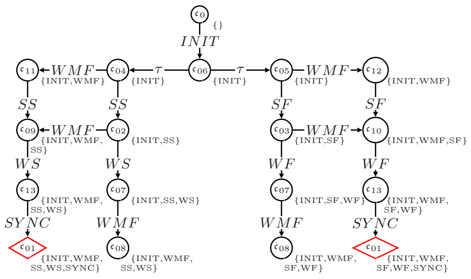

Fig. 5. Unfold LTS for the odometer

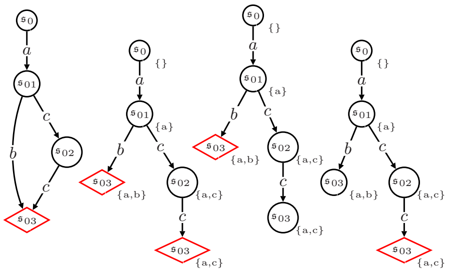

Fig. 6. Overview: (a) LTS (b) eLTS (c) LTS$_{mcs_{\{a,b\}}}$ (d) LTS$_{mcs_{\{a,c\}}}$

the constant failure rates λ_a, λ_c this approach yields a less accurate probability measure: $Q(t)_{\{a,c\}} = (1 - e^{-\lambda_a t}) \cdot (1 - e^{-\lambda_c t})$. We consider all faults (stochastic) independent and assume that *common mode analysis* has been carried out in order to validate this assumption.

4 Case Study

In this section, we enrich the train odometer example of Sect. 2 and furthermore present numbers for a more complex model, to show the general feasibility and scalability of our approach[2].

4.1 A Train Odometer Controller

The odometer system under study consists of two independent sensors used to measure speed and position of a train. A *Wheelsensor* is mounted to an unpowered wheel of the train to count the number of revolutions. A *Radarsensor* determines the current speed by evaluating the Doppler shift of the reflected radar signal. We consider transient faults for both sensors. For example water on or beside the track could interfere with the detection of the reflected signal and thus cause a transient fault in the measurement of the *Radarsensor*. Similarly, skidding of the wheel affects the *Wheelsensor*. Due to the sensor redundancy the system is robust against faults of a single sensor. However it has to be detectable to other components in the train, when one of the sensors provides invalid data. For this purpose a *Monitor* continuously checks the status of both sensors. We will focus on this monitoring aspect of the system. Figure 7 shows the corresponding Statechart model. The *Radarsensor* starts in the initial state ROK and, when a fault occurs, enters state RF. The transient nature of the fault is implemented by the transition back to the state ROK. The *Wheelsensor* behaves like the *Radarsensor* with the exception that the rate for the fault depends on the current, non-deterministically selected, Speed of the train. Whenever either

[2] The experiments were carried out on a PC with P4 3GHz processor and 1GB RAM.

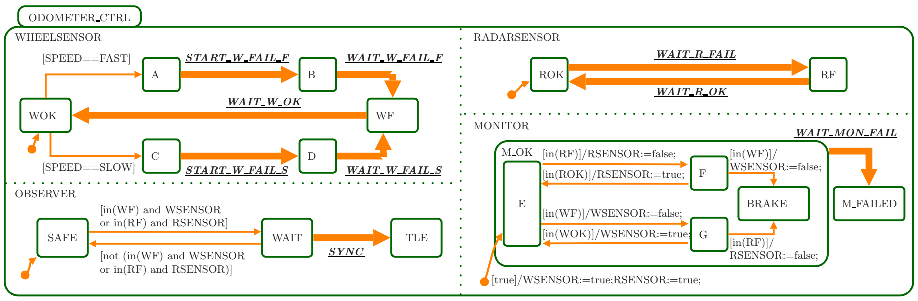

Fig. 7. A train odometer controller

the *Wheelsensor* or the *Radarsensor* fail and enter the WF or RF state respectively this is detected by the *Monitor* and the corresponding status variable (WSENSOR or RSENSOR) is set to *false*. This information can be used by other train components that have to disregard temporary erroneous sensor data. Due to the robustness against single faults and since both sensor faults are transient the system even can recover completely from such a situation. If both sensors fail the system is shut down by the *Monitor* (i. e. a brake manoeuvre is initiated), but also in this case the system is safe. Only if the *Monitor* fails first, any subsequent faults in the sensors will no longer be detected. Since now the train may be guided by invalid speed and position information such situations are safety critical. We therefore define the entering of node TLE of the *Observer* component as the safety critical state. Consequently, three minimal cut sets (mcs_a, mcs_b, mcs_c) exist in this model. Table 1 shows all labelled transitions and their mapping to the particular cut sets. Note, that we do not list implicit labels, such as the system-start transition (INIT) used for instance to delay the *Monitor* fault. Based on the listed rates, the lower part of the tables compare the concrete probabilities, we computed for the model using the stochastic model checker MRMC[3] [18],

Table 1. Rates and minimal cut set probabilities

Label		mcs_a	mcs_b	mcs_c	rate	mcs_a	mcs_b	mcs_c	rate
WAIT_MON_FAIL		yes	yes	yes	0.001	yes	yes	yes	0.01
WAIT_R_FAIL		yes	no	no	0.015	yes	no	no	0.04
START_W_FAIL_F		no	yes	no	-	no	yes	no	-
WAIT_W_FAIL_F		no	yes	no	0.025	no	yes	no	0.003
START_W_FAIL_S		no	no	yes	-	no	no	yes	-
WAIT_W_FAIL_S		no	no	yes	0.01	no	no	yes	0.3
WAIT_W_OK		no	no	no	240	no	no	no	0.001
WAIT_R_OK		no	no	no	360	no	no	no	0.002
time bound (h)	method	$P(t)$	$P(t)$	$P(t)$	-	$P(t)$	$P(t)$	$P(t)$	-
10	mrmc	0.0007122	0.0011479	0.0005027	-	0.0149402	0.0014070	0.0318196	-
	naive	0.0013860	0.0022010	0.0009469	-	0.0313732	0.0028125	0.0904247	-
100	mrmc	0.0479257	0.0605680	0.0397182	-	0.2343468	0.0724729	0.1887776	-
	naive	0.0739289	0.0873512	0.0601542	-	0.6205429	0.1638341	0.6321206	-
1000	mrmc	0.5973047	0.6078448	0.5677622	-	0.7310061	0.5490308	0.2095549	-
	naive	0.6321204	0.6321206	0.6320919	-	0.9999546	0.9501698	0.9999546	-

[3] That in particular supports uCTMDP analysis [17].

with those derived following the naive approach (cf. Sect. 3). It can be observed that our model based approach yields probabilities well below those of the naive approach. Moreover the numbers of the right table show that even the *ranking* of the MCSs induced by the probabilities can *diverge*. While the naive approach indicates 1. mcs_c, 2. mcs_a, 3. mcs_b, the model based approach yields 1. mcs_a, 2. mcs_b, 3. mcs_c for sufficiently large time bounds. Intuitively, in this example, the high failure rates make it probable that the monitor signals the sensor to be inoperable before failing itself and thus prevents a safety critical situation. Thus, here, the naive approach that does not take the concrete system dynamics into account, even yields a misleading importance measure.

4.2 Performance Remarks

The upcoming *European Train Control System (ETCS)*, is designed to replace the multitude of incompatible safety systems used by European Railways and enable safe fast transnational railway service. Based on a fault tree description taken from the ETCS specification [19], we developed an ETCS Level 2 train model that we use to show the inherent worst case complexity of the path encoding to be negligible in practice. Table 2 depicts the numbers of the intermediate models (cf. Fig. 1). Note that the LTS extraction and the branching bisimulation are implemented using efficient representations of the state space using binary decision diagrams. The remaining steps rely on explicit representations of the state space and make use of the CADP [20] tool box. We found that (i) the efficient symbolic branching bisimulation on the *LTS* yields enormous reductions of the state space in the *quotient LTS* and thus is a crucial preprocessing step to our unfolding step. Moreover (ii) the *stochastic* branching bisimulation [21,17] that interleaves the (one-by-one) incorporation steps of the stochastic time constraints balances the introduced complexity (cf. column *eLTS*) of the unwinding: the finally computed stochastic models are of similar size to the model generated for the original LTS (cf. uCTMDP-LTS and uCTMDP-$eLTS_{mcs_{1-4}}$).

Table 2. ETCS Level 2 Train Model - comparison of LTSs and uCTMDP state spaces

	LTS	quotient LTS	uCTMDP-LTS	eLTS	uCTMDP-$eLTS_{mcs_1}$	uCTMDP-$eLTS_{mcs_2}$	uCTMDP-$eLTS_{mcs_3}$	uCTMDP-$eLTS_{mcs_4}$
states	8266964	142789	3230	1071250	10023	3911	3919	1654
transitions	18313109	727609	15680	5538073	48066	22347	19137	8348
time (sec.)	12222.1	4714.24	173.15	5617.79	1013.48	674.42	635.77	628.52

5 Conclusion

In this paper we presented an automated model-based approach to determine the quantitative contribution of safety critical fault configurations (MCSs) to the over-all probability of reaching a safety critical state.

Therefore we extended the tool chain presented in [4,5] by an encoding of path information into the state space of an LTS that enables to distinguish (and relate) all relevant paths leading to a safety critical state to a MCS. This also allows for the extraction of the MCSs itself, but this is secondary. We gave a detailed explanation of the LTS encoding as well as of how the LTS is derived from an extended Statechart [5]. The over-all approach was successfully applied to a case-study taken from the train-control domain. In particular, we observed that the sophisticated combination of the symbolic branching and stochastic branching minimisation steps balances the encodings inherent complexity.

On the application side the benefits are obvious. The derived MCS importance measures enable the system developer to direct their work on those parts of the systems where improvements will yield most impact. Due to the preservation of the actual sequences of faults, the derived measures are more accurate than conventional naive or manual safety assessment techniques that may even yield misleading results. And – thanks to the automation – they can be derived without any extra effort directly from the design model. Furthermore, by incorporating stochastic non failure behaviour (e. g. repair rates of transient faults) into the modelling and cut set analyses, again the accuracy can be improved, that is, a less pessimistic assessment is derived.

All in all the (extended) tool chain seems to be a crucial step towards a better integration of system development and quantitative safety analysis. Future work will therefore strive for further integration of conventional safety assessment tasks and refinements such as the inclusion of common mode failures.

Acknowledgements. We would like to thank our co-authors of [4,5] for the tight cooperation resulting in those works making this work possible at all.

References

1. Åkerlund, O., et al.: ISAAC, a framework for integrated safety analyses of functional, geometrical and human aspects. ERTS (2006)
2. Vesely, W.E., Dugan, J., Fragola, J., Minarick III, J., Railsback, J.: Fault Tree Handbook with Aerospace Applications. National Aeronatics and Space Administration (August 2002)
3. Peikenkamp, T., Cavallo, A., Valacca, L., Böde, E., Pretzer, M., Hahn, E.M.: Towards a unified model-based safety assessment. In: Górski, J. (ed.) SAFECOMP 2006. LNCS, vol. 4166, pp. 275–288. Springer, Heidelberg (2006)
4. Böde, E., Herbstritt, M., Hermanns, H., Johr, S., Peikenkamp, T., Pulungan, R., Wimmer, R., Becker, B.: Compositional performability evaluation for statemate. In: 3rd International Conference on the Quantitative Evaluation of Systems, QEST 2006, Riverside (USA), pp. 167–178. IEEE Computer Society Press, Los Alamitos (2006)
5. Böde, E., Herbstritt, M., Hermanns, H., Johr, S., Peikenkamp, T., Pulungan, R., Rakow, J., Wimmer, R., Becker, B.: Compositional performability evaluation for statemate. In: Quantitative Evaluation of Computer Systems - Special issue of IEEE Transactions on Software Engineering (to appear, 2008)
6. Harel, D., Politi, M.: Modelling Reactive Systems with Statecharts: The STATEMATE Approach. McGraw-Hill, New York (1998)

7. Hermanns, H., Katoen, J.P.: Automated compositional markov chain generation for a plain-old telephone system. Science of Computer Programming 36(1), 97–127 (2000)
8. Pulungan, R., Hermanns, H.: Orthogonal distance fitting for phase-type distributions. Reports of SFB/TR 14 AVACS 10, SFB/TR 14 AVACS (November 2006) ISSN: 1860-9821, `http://www.avacs.org`
9. Hermanns, H.: Interactive Markov Chains – The Quest for Quantified Quality. LNCS, vol. 2428. Springer, Heidelberg (2002)
10. Hermanns, H., Johr, S.: Uniformity by construction in the analysis of nondeterministic stochastic systems. In: International Conference on Dependable Systems and Networks, DSN 2007 (2007)
11. Boudali, H., Crouzen, P., Stoelinga, M.: Dynamic fault tree analysis using input/output interactive markov chains. In: DSN 2007: Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Washington, DC, USA, pp. 708–717. IEEE Computer Society Press, Los Alamitos (2007)
12. Harel, D., Naamad, A.: The STATEMATE semantics of statecharts. ACM Transactions on Software Engineering and Methodology 5(4), 293–333 (1996)
13. Milner, R.: A Calculus of Communicating Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
14. Glabbeek, R., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. Journal of the ACM 43(3), 555–600 (1996)
15. Wimmer, R., Herbstritt, M., Hermanns, H., Strampp, K., Becker, B.: Sigref – a symbolic bisimulation tool box. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 477–492. Springer, Heidelberg (2006)
16. Hermanns, H., Johr, S.: May we reach it? or must we? in what time? with what probability? In: Proceedings 14th GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB 2008), Dortmund, Germany, March 31 - April 2, 2008, VDE Verlag (to appear, 2008)
17. Johr, S.: Model Checking Compositional Markov Systems. PhD thesis, Universität des Saarlandes, Saarbrücken (2007)
18. Katoen, J.P., Khattri, M., Zapreev, I.S.: A markov reward model checker. In: Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), Torino, Italy, 19-22 September 2005, pp. 243–244. IEEE Computer Society Press, Los Alamitos (2005)
19. ERTMS User Group, UNISIG: ETCS Application Level 2 - Safety Analysis - Part 1 - Functional Fault Tree. Technical report, ALCATEL,ALSTOM,ANSALDO SIGNAL,BOMBARDIER,INVENSYS RAIL,SIEMENS
20. Garavel, H., Lang, F., Mateescu, R.: An overview of CADP 2001. European Assoc. for Software Science and Technology (EASST) Newsletter 4, 13–24 (2002)
21. BCG_MIN: Project Website (March 2006), `http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html`