A Framework for Qualitative and Quantitative Formal Model-Based Safety Analysis

Matthias Gudemann
Computer Systems in Engineering
Otto-von-Guericke University Magdeburg
Magdeburg, Germany
matthias.guedemann@ovgu.de

Frank Ortmeier
Computer Systems in Engineering
Otto-von-Guericke University Magdeburg
Magdeburg, Germany
frank.ortmeier@ovgu.de

Abstract—In model-based safety analysis both qualitative aspects (i.e., what must go wrong for a system failure) and quantitative aspects (i.e., how probable is a system failure) are very important. For both aspects methods and tools are available. However, until now for each aspect new and independent models must be built for analysis.

This paper proposes the SAML framework as a formal foundation for both qualitative and quantitative formal model-based safety analysis. The main advantage of SAML is the combination of qualitative and quantitative formal semantics which allows different analyses on the same model. This increases the confidence in the analysis results, simplifies modeling and is less error-prone.

The SAML framework is tool-independent. As proof-of-concept, we present sound transformation of the formalism into two state of the art model-checking notations. Prototypical tool support for the sound transformation of SAML into PRISM and MRMC for probabilistic analysis as well as different variants of the SMV model checker for qualitative analysis is currently being developed.

Keywords—fault tolerance, reliability, formal methods, model checking

I. INTRODUCTION

The complexity of software intensive systems is rising constantly. More and more functions are moved to software controlled units, e.g. X-by-wire systems. In many domains like avionics or automotive, malfunctioning of a system poses a severe safety risk.

To control this evolution, safety analysis has become an important focus in most engineering domains. Requirements of how to develop, analyze and manage the life cycle of a safety-critical system are specified in many different standards, like IEC 61508 [1], DO-178 B [2] (aviation) or ISO 26262 [3] (automotive). All of those standards require a safety assessment of the product prior to its deployment.

For high risk areas, even formal safety analysis methods are needed or at least highly required. In the last decade, numerous formal and model-based analysis techniques have been developed. They help engineers in analyzing a system’s safety properties. These methods can typically be split into two major groups. The first one deals with qualitative aspects, i.e. what must go wrong for a critical system failure. These methods most often rely on formal verification and model checking techniques [4][5][6][7]. The second group addresses quantitative aspects, i.e. what is the probability of a critical system failure. They rely on stochastic models, quantitative approximations and/or Markov-chains/processes [8][9][10][11].

For a safety case (a set of evidence which shows the safety of a system), typically qualitative as well as quantitative analysis is needed. Until now, both analysis methods are performed independently. Even if model-based approaches are applied, the models used for qualitative analysis and those for quantitative analysis have to be built separately which is potentially very time-consuming and error-prone.

This paper presents the safety analysis modeling language (SAML) which can be used for both qualitative and quantitative analyses and allows for the combination of discrete probability distributions and non-determinism. The framework is used to create a system model comprised of software control, hardware components, environment and failure mode modeling. For safety critical systems, all those aspects have to be considered together. This model is then analyzed using deductive, formally founded model-based analysis techniques which are applicable for both reliability and safety analysis.

The rest of the paper is structured as follows: some related work is discussed in Sect. II. The syntax and semantics of SAML are presented in Sect. III. In Sect. IV modeling guidelines for different failure types are presented. Sect. V shows the approach to transform SAML models into semantically equivalent models for state of the art analysis tools. First results of this implementation and the combination of the currently supported safety analysis techniques with SAML are presented in Sect. VI for a small case study.

II. RELATED WORK

A widely accepted qualitative method to find critical combinations of failure modes is fault tree analysis (FTA) [12] or formal fault tree analysis [13], [5] which constructs an additional model orthogonal to the functional model by breaking down the overall hazard into smaller intermediate events down to basic failure modes, resulting in minimal cut sets. They can often be constructed complete, i.e. not forgetting any critical combination. The problem is to construct them in a correct way such that every minimal cut set can cause a hazard. A fault tree can be complete but incorrect in the sense that the analysis is too pessimistic [14].
To prevent this problem, other methods extract the critical failure combinations directly from a model using automatic deduction techniques, skipping the construction of an additional model. One example is DCCA [14], other methods relying on fault injection were developed in the ESACS project [15] like the FSAP/NuSMV-SA framework [6] or in the ISAAC project [7] were SCADE was used for modeling and safety analysis [16]. In contrast to DCCA, these methods have no formal guarantee for completeness and correctness of the analysis.

For quantitative safety analysis, many approaches rely on the analysis of a previous qualitative analysis and the resulting critical combination of failures. To estimate the global hazard probability, the probabilities of the single failure modes are assumed to be stochastically independent. Therefore the hazard probability can only be approximated [5]. This approximation can be quite coarse and the assumption of stochastic independence is often false.

Newer approaches have been developed to deduce quantitative information from a formal system model, mostly using probabilistic model-checking techniques. In the COMPASS project [17] the FSAP-NuSMV/SA [6] framework is combined with the MRMC [18] probabilistic model checker to allow for the analysis of systems for Aerospace applications specified in the SLIM [19] language, which is inspired by AADL [20], [21]. The hybrid behavior of the SLIM models and all internal transitions are removed by lumping and the resulting interactive Markov chain is analyzed with MRMC.

Another approach for probabilistic safety analysis is probabilistic failure modes and effects analysis (FMEA) [10] where per-time failure behavior is integrated into system models via failure injection. After then, FMEA tables are computed which are often used in industrial safety analysis processes.

Yet, none of these approaches allows for the integration of both per-time and per-demand failure modes. These two different probabilistic occurrence pattern are described in IEC 61508 [1] as high or continuous demand and low demand failure modes. The approaches use continuous Markov chains models (CTMC) as system models which allows per-time failure mode modeling with failure rates. CTMCs are well suited for modeling asynchronous, interleaved systems but not for synchronous parallel systems [22] as many safety critical system are.

III. SYNTAX AND SEMANTICS

A SAML model describes finite state automata that are executed in a synchronous parallel fashion with discrete time-steps. The automata are described by modules that contain state variables that are updated according to transition rules. These transitions can contain both non-determinism and probabilistic choice.

A. Syntax

In Fig. 1, an example SAML model is shown. It is comprised of two modules, \( A \) and \( B \). Module \( A \) contains the state variable \( V_A \) with a value range from 0 to 2 and an initial value of 0. Module \( B \) contains 2 state variables, both with a value range from 0 to 1 and an initial value of 0. In module \( A \), there are 5 update rules. Module \( B \) has only 1 update rule, but with two non-deterministic choices. For both modules, every choice contains probabilistic transitions with parallel assignments for all state variables contained in the respective module.

\[
\text{constant double } P_A := 0.1; \\
\text{constant double } P_B1 := 0.2; \\
\text{constant double } P_B2 := 0.3; \\
\text{constant double } P_B3 := 0.5; \\
\text{formula } CASE_3 := (V_A = 0) \& (V_B1 = 0 \& V_B2 = 0) \lor (V_B1 = 1 \& V_B2 = 1); \\
\text{module } A \\
V_A := 0..2 \text{ init } 0; \\
V_A := 0 \& V_B1 := 0 \& V_B2 := 0 \Rightarrow \text{choice } (P_A : (V_A' = 0) + (1 - P_A) : (V_A' = 1)); \\
V_A := 0 \& V_B1 := 1 \& V_B2 := 1 \Rightarrow \text{choice } (1 : (V_A' = 2)); \\
CASE_3 := \text{choice } (1 : (V_A' = 1)); \\
V_A := 1 \Rightarrow \text{choice } (1 : (V_A' = 1)); \\
V_A := 2 \Rightarrow \text{choice } (1 : (V_A' = 2)); \\
\text{endmodule}
\]

\[
\text{module } B \\
V_B1 := 0..1 \text{ init } 0; \\
V_B2 := 0..1 \text{ init } 0; \\
true \Rightarrow \text{choice } (P_B1 : (V_B1' = 0) \& (V_B2' = 0) + \\
P_B2 : (V_B1' = 1) \& (V_B2' = 0) + \\
P_B3 : (V_B1' = 1) \& (V_B2' = 1) + \\
\text{choice } (1 : (V_B1' = 1) \& (V_B2' = 1)); \\
\text{endmodule}
\]

Fig. 1. Example SAML model

The syntax of a SAML model is shown in Fig. 2 in Extended Backus Naur Form (EBNF) notation. To simplify its description, only the most important grammar rules are presented\(^1\). The actual implementation is done using the ANTLR parser generator [23]. The syntax is very similar by the input language of the PRISM model checker [24][25]. The main differences are the absence of synchronization labels and the explicit modeling of non-deterministic choices with the choice keyword.

A SAML model consists of the declaration of zero or more constants and formulas and at least one module description. Constants have an associated name (identifier), a type (double, int or bool) and an optional value of that type. A formula is comprised of its name and a boolean expression in propositional logic. A term used in a boolean expression can be either the value of \( true \) or \( false \), a declared boolean constant or an expression over the state variables. Formulas are basically used as abbreviations for complex boolean formulas.

After the optional declaration of constants and formulas, the module declarations follow. Each module is delimited by the keywords module and endmodule and has an associated name. The content of the module is at least one declaration of a state variable and at least one update rule. Every state variable represent a range of integers and is declared with a name, its lower and upper bound, as well as its initial value.

\(^1\)Lexer rules are written uppercase, parser rules in lowercase. Bold font indicates keywords or literal symbols without explicit lexer rules.
saml-model : (constant | formula)* module+ ;
countant : constant TYPE IDENT (⇒ value)? ;
formula : formula IDENT := condition ;
condition : ( condition ) |
| ! condition |
| condition & condition |
| condition | condition |
| term ;
term : IDENT (⇐ | <= | => | <<=) state_expr |
| IDENT | true | false ;
module : module IDENT declaration+ update+ endmodule ;
declaration : IDENT : [ INT - INT ] init INT ;
update : condition -> non-det_assigns |
| prob_assigns ;
non-det_assigns : non-det_assign |
(∗ non-det_assign)* ;
non-det_assign : choice ( prob_assigns ) ;
prob_assigns : prob_assign (∗ prob_assign)* ;
prob_assign : probability ; nextstate_assign |
& nextstate_assign)* ;
probability : IDENT | DOUBLE | arith_expr ;
nextstate_assign : ( IDENT = state_expr ) ;
state_expr : IDENT | INT | ( state_expr ) |
state_expr (∗ | + | - | / | *) state_expr ;
arith_expr : INT | DOUBLE | IDENT | ( arith_expr ) |
arith_expr (∗ | + | - | / | *) arith_expr .

Fig. 2. Basic SAML syntax

Every update of a module is comprised of a boolean activation condition and at least one non-deterministic choice for assignments or at least one probabilistic assignment. Several non-deterministic assignments are specified as a sum of choices. Each single such non-deterministic assignment is comprised of the keyword choice and probabilistic assignments. Each single probabilistic assignment starts with its probability (either a double value, an identifier or an arithmetic expression) and is comprised parallel assignments of a new value to a state variable (separated by ∗). Omitting the keyword choice is possible if only one probability distribution is given.

Each assignment of a new value to a state variable is comprised of the name of the variable and a state expression. For a state variable v the name of the variable for the next value is v’. A state expression is either an integral value or an arithmetic expression over the state expressions.

B. Semantics

In order to integrate both quantitative and qualitative analysis on one common system model, the semantics must be defined on a powerful enough model that supports both non-deterministic and probabilistic behavior. Therefore the semantics of a SAML module is defined as a Markov Decision Process (Def. (1)). The following definitions are based on [7] and [26]. A MDP represents a labeled transition system with a finite set of states and allows the combination of probabilistic and non-deterministic transitions. The semantics is defined for one single module, in Sect. III-C a parallel composition operator is presented which constructs on the syntactical level one single SAML module from several parallel ones.

Definition 1: Markov Decision Process A Markov Decision Process (MDP) is a tuple

\[ \tau_{MDP} = (S, \text{Steps}, AP, L, s_0) \]

- S is a finite set of states
- Steps : S → 2^Idx×Dist(S) is the transition probability function, where Idx ⊂ N is the set of indices for the elements of Dist(S), which is the set of discrete probability distributions over the set S. Each pair (j, p) ∈ Idx × Dist(S) defines a function p : S × S → [0, 1] with
  - p(s, _) is a probability distribution on S
  - ∀s ∈ S : Σt∈Tp(s, t) = 1
- AP is a finite set of atomic propositions
- L is a labeling function L : S → 2^AP that labels each state s ∈ S with a subset of the atomic propositions that hold in this state
- s_0 ∈ S is the initial state

A SAML model is interpreted as MDP by defining a state s ∈ S for each possible valuation of the state variables. The initial valuation of the state variables corresponds to the initial state s_0. Each update rule of the SAML model represents a set of possible transitions of the MDP. The condition of the update rule describes a subset of the atomic propositions that hold in this state.

In order to be a proper MDP, a SAML model must fulfill the following constraints on the activation conditions and probabilities of the updates:

1) \( \sum_i p_i = 1 \) for each probabilistic assignment
2) \( \bigwedge_j \phi_j \equiv \text{true} \) for all activation conditions \( \phi \)
3) \( \forall i \neq j : \phi_i \land \phi_j \equiv \text{false} \) for each activation condition

The first constraint assures that each probabilistic assignment is a proper probability distribution. The second constraint assures that there is always an activation condition that holds. Therefore for each state there exists always a successor state that is reached with a positive probability. The third constraint assures that there are no overlapping activation conditions, i.e. the activation conditions describe a disjoint partitioning of the set of states.

A run of a MDP is a sequence of states that are reached. This sequence together with the non-deterministic choices for the probability distributions is called the path of a MDP (Def. (2)). Each state is labeled via the labeling function L. The resulting sequence of labels is called the trace of a MDP (Def (3)).

Definition 2: Path of a MDP A finite or infinite path of a MDP is a sequence of states and pairs of actions and distributions such that for a sequence s_0(j_0, p_0)s_1(j_1, p_1) . . .

\( (j_i, p_i) \in \text{Steps}(s_i) \) and \( \forall i \geq 0 : p_i(s_i, s_{i+1}) > 0 \)

Definition 3: Trace of a MDP A trace \( \sigma \) of a path s_0(j_0, p_0)s_1(j_1, p_1) . . . of a MDP Model \( \tau_{MDP} \) is the word
over the alphabet $2^{AP}$, generated from
\[
\sigma(s_0(j, p_0)s_1(j_1, p_1) \ldots) = L(s_0)L(s_1) \ldots
\]

In the path of an MDP, each state $s$ is followed by a pair $(j, p)$ of an index $j$ and an associated probability $p$ as well as a successor state $s'$. The index represents the non-deterministic choice that has been made, i.e. which probability distribution of $P = Steps(s)$ has been chosen. The probability $p$ is then $P(s, s')$, by definition $p > 0$. A sequence of actions that is chosen non-deterministically is described by an adversary (Def. (4)).

**Definition 4: Adversary** An adversary $A$ of an MDP $\tau_{MDP}$ is a function that maps all finite paths $\omega = s_0(j, p_1)s_1 \ldots s_n$ to one element of $Steps(s_n)$.

The exact definitions of semantics and probability measures are out of the scope of this paper but can be found in [27][?][26]. An adversary basically decides non-deterministically in each state $s$ (beginning from the initial state) which discrete probability distribution is selected. This probability distribution $P$ defines at least one successor state $s'$ for which $P(s, s') > 0$ holds.

**C. Parallel Composition**

The semantics of SAML, as described in Sect. III-B is defined as single module. Analogously to [25], Def. (III-C) introduces the operator $||$ which defines the syntactical parallel composition of two SAML modules.

**Definition 5: Parallel composition of SAML modules** For $M = M_i || M_j$ create for each update
\[
\phi^i \rightarrow \sum_{k=1}^c choice^i_k \left( \sum_{l=1}^{d_k} p_{kl} : u_{kl}^i \right)
\]
of $M_i$ and
\[
\phi^j \rightarrow \sum_{m=1}^c choice^j_m \left( \sum_{n=1}^{f_m} p_{mn}^j : u_{mn}^j \right)
\]
of $M_j$, a product update for $M$ of the form
\[
\phi^i \land \phi^j \rightarrow \sum_{k=1}^c \sum_{m=1}^c choice \left( \sum_{l=1}^{d_k} \sum_{n=1}^{f_m} p_{kl}^i \cdot p_{mn}^j : u_{kl}^i \land u_{mn}^j \right)
\]

For the parallel composition of updates from two modules $M_i$ and $M_j$, each probability distribution of each of the $c$ non-deterministic choices of the first module (Eq. (1)) is combined with each probability distribution of each of the $f_m$ non-deterministic choices of the second module (Eq. (2)). The result is an update with an activation condition which is the conjunction of the activation conditions of the original updates and $c \cdot f_m$ non-deterministic choices. Each of these choices is comprised of a new probability distribution and parallel assignments of the state variables.

**Definition 6: Complete Model** For a SAML model which consists of the single modules $M_1, \ldots M_n$, the complete model $M$ is defined as $M := M_1 || \ldots || M_n$.

The complete model generated from the parallel composition of the two modules shown in Fig. 1 is $A_B := A || B$. The result is shown in Fig. 3. Although every SAML model can be explicitly expressed as one single module, it is obvious that it is much more convenient to specify parallel modules that are combined with the $||$ operator. The manual construction of the product module is much too error prone and even in this simple example the resulting model is rather confusing.

**IV. Failure Mode Modeling**

The main idea of failure mode modeling in model-based analysis is to separate the occurrence pattern from the effect. The occurrence pattern describes when a failure mode may occur. The failure effect models the direct effect if a failure mode occurs.
A. Basic Failure Mode Modeling

The occurrence pattern of a failure mode is basically described by a “failure module”. The most basic failure automaton has two states. A state “no” signaling the absence and a state “yes”, signaling the presence of a failure mode. The two simplest forms of failure mode modeling using these two states are transient failures which can disappear again and persistent failures which stay permanent once they occurred. Examples for transient failure modes are sensor failures which are often only temporal. Examples of persistent failure modes are non-recoverable hardware failures. These two are only the simplest failure mode modeling possibilities, more complicated occurrence pattern like repairable failure modes can be modeled analogously.

The failure effect modeling is done directly in the system model. No general instructions can be given, as this is very dependent on the actual system under consideration. Important for the soundness of the analysis is the conservative integration of the failure modes. This means that the original behavior of the system is still contained in the extended system model (the system with failure effect modeling). This means that all traces of the original system are also traces of the extended system model (if no failure occurs). Guidelines with proven rules to achieve this are presented in [28].

For quantitative failure analysis, the occurrence probability of a failure mode is of greatest importance. Information about the occurrence probability can be specified either as a failure rate \( \lambda \) which describes the number of expected failure occurrences in a given time (most often \( 1 \) s). Such a failure mode describes a per-time failure mode that can only occur if there is a demand to the safety critical system.

B. Temporal Resolution

As described in Sect. III the semantics of SAML modules is synchronously parallel with discrete time steps. This means that all modules execute a discrete step together and the same amount of time passes at each such step. This smallest amount of time is called the temporal resolution \( \delta t \) of the system.

This means that all quantitative modeling in a SAML model must be made according to \( \delta t \). This includes environment modeling as shown in [29][9] which is model specific and is not part of this paper as well as failure mode modeling.

C. Per-Time Failure Modeling

A per-time failure mode rate \( \lambda \) specifies the parameter for the exponential distribution as shown in Eq. (3). Here \( P(X \leq t) \) describes the probability that the failure mode appears before or at time \( t \).

\[
P(X \leq t) = \int_0^t e^{-\lambda t} dt = 1 - e^{-\lambda t}
\]

This distribution is often used for failure modes in continuous time models. As a continuous probability distribution, it is not directly expressible in a discrete time context. Nevertheless, using \( \delta t \) it can be approximated using the geometric distribution as shown in Eq. (4). This distribution computes the probability that the failure mode appears before or at \( k \) time-steps of length \( \delta t \).

\[
P(X \leq k) = 1 - P(X > k) = 1 - (1 - p)^k
\]

The smaller the basic time unit \( \delta t \) is, the better is this approximation. Its actual error is often orders of magnitude...
smaller than the probabilities [9]. In order to compute a probability for every discrete time step of the system, the per-step failure probability as in Def. (7) is computed.

**Definition 7: Per-Step Failure Probability** For a per-time failure mode with a failure rate \( \lambda \) (unit \( \frac{1}{t} \)), the per-step failure probability \( p_{\text{step}} \) for \( t = k \cdot \delta t \) is defined as:

\[
p_{\text{step}} = \lambda \delta t
\]

Overall the continuous exponential probability distribution is approximated using the geometric distribution with the per-step probability \( p = \lambda \delta t \) as shown in Eq. (5). The derivation can be found in [9].

\[
1 - e^{-\lambda t} \approx 1 - (1 - \lambda \delta t)^k = 1 - (1 - p)^k \tag{5}
\]

In SAML per-time failure modes are modeled as failure modules as shown in Fig. 5. Both a transient and persistent occurrence pattern is shown. The state variable \( \text{occurs} \) indicates whether a failure occurs.

```plaintext
module TRANSIENT-PER-TIME-FAILURE
occurs:[0..1] init 0; // 0 = "no"; 1 = "yes"
true -> P_STEP:(occurs'=1)+(1-P_STEP):(occurs'=0);
endmodule

module PERSISTENT-PER-TIME-FAILURE
occurs:[0..1] init 0;
occurs=0 ->
P_STEP:(occurs'=1)+(1-P_STEP):(occurs'=0);
occurs=1 -> 1:(occurs'=1);
endmodule

Fig. 5. Per-time Failure Mode Modeling
```

**D. Per-Demand Failure Modeling**

The occurrence pattern for a per-demand failure mode is more difficult to model. It must be assured, that the failure effect can only appear at the moment of a demand to the safety critical component. The basic idea is to define a predicate \( \text{demand} \) for each possible per-demand failure mode.

For each such demand, the activation condition of the failure automaton is then the conjunction with the \( \text{demand} \) predicate. This assures that the failure can occur only when the safety critical system is requested.

In this solution the failure module and the module in which the failure effect is modeled, take the transition at the same time. The problem is that the transition of the failure effect modeling is dependent on the value of the failure module, but the information whether the failure mode occurred is available one time step too late. This problem is solved as shown in Fig. 6. The basic idea it to use the result of a previous probabilistic transition to signal whether the current demand can be met. So if \( t_i \) specifies the time of the \( i^{th} \) demand, the transition at time \( t_{i-1} \) decides the outcome at \( t_i \). In order to cope with the very first demand, an additional state is introduced to the failure module, the state \(-1\), which is also the initial state. This initial state is left in the first time-step and decides the outcome for the first demand to the safety critical system. Strictly speaking, this solution delays the occurrence possibility of a per-demand failure mode for one time-step after the beginning of a system run. If that is a problem, an additional “beginning” state for each module can be added. Another possibility is to use the per-demand failure mode integration as described in [9], which introduces additional “undecided” states and requires elaborate changes of the module in which the failure effect is modeled. In most cases, the per-demand integration is feasible the way described here which is simpler than the outlined alternatives.

**V. Model Transformation**

In order to analyze an extended system model in SAML, it must be transformed into the input language of different model-checking tools, dependent on the type of the model and on the nature of the property to be verified.

**A. Transformation to Probabilistic Model-Checkers**

The SAML syntax is very similar to the input language [25] of the PRISM model-checker. So its transformation is straightforward. PRISM allows full asynchronicity and specifies synchronous transition explicitly. SAML models are fully synchronous, so that every transition gets synchronized via the same synchronization label \( \text{tick} \) (for \( \text{tick} \)). Non-deterministic choices are described slightly different in PRISM. Instead of explicitly specifying the non-deterministic choices in one update, two or more updates with the same activation condition are specified, as shown in Fig. 8. In addition to PRISM, the SAML model can also be transformed into the input language of the MRMC [18] model-checker. The easiest way to get this is to use PRISM which has an export feature for MRMC models. Unfortunately, MRMC cannot handle nondeterminism for discrete time models. So at the moment only
SAML models that have only one probability distribution for each update can be analyzed with MRMC.

B. Transformation to Qualitative Model-Checkers

A MDP cannot directly be analyzed with qualitative model-checking tools like NuSMV [30] or Cadence SMV [31]. In order to use any of these tools, the qualitative semantics of an MDP must be specified.

1) Qualitative Semantics: The qualitative semantics of a MDP is defined as a Kripke structure (Def. (8)) which is embedded in the quantitative MDP model. A Kripke structure is a labeled transition system with non-deterministic but without probabilistic transitions.

**Definition 8: Kripke structure** A Kripke structure over a set AP of atomic propositions is a tuple $\tau_{Kripke} = (S, s_0, T, L, AP)$ with

- $S$ a finite, non-empty set of states
- $s_0 \in S$ an initial state
- $T \subseteq S \times S$ a total transition relation, i.e. for each state $s \in S$ there is a state $s' \in S$ such that $T(s, s')$ holds
- $L : S \rightarrow 2^{AP}$ is a mapping that labels each state $s \in S$ with a subset of the atomic propositions that hold in this state (all others are false)

The paths $\pi_{Kripke}$ and traces $\sigma_{Kripke}$ of a Kripke structure are defined analogously to those of a MDP, as sequences of states. The difference here is that successor states $(s, s')$ must be an element of the transition relation $T$, but there is no probability associated with such a transition.

For the integration of qualitative analysis, we show that a MDP defines a proper embedded Kripke structure (see Thm. (1)).

**Theorem 1:** Let $\kappa(\tau_{MDP})$ be a mapping of a MDP $\tau_{MDP} = (S, \text{Steps}, AP, L, s_0)$ to a tuple $(S, s_0, T, L, AP)$ with

$$T := \{(s, t) | s, t \in S \land \exists (j, p) \in \text{Steps}(s) : p(s, t) > 0\}$$

Then $\tau_{Kripke} = (S, \text{Steps}, L, AP)$ is a Kripke structure.

**Proof:** By definition the set of states, initial state, labeling and atomic propositions is the same in $\tau_{MDP}$ as in $\tau_{Kripke} = \kappa(\tau_{MDP})$.

In order for $\tau_{Kripke}$ to be a proper Kripke structure, the transition relation $T$ must be total, i.e. $\forall s \in S : \exists t \in S : (s, t) \in T$. From the definition of $T$ and Def. 1 we know that each state $s$ has a set of action, discrete probability distribution pairs $(j, p)$. For each such $p$ we know that $\sum_{t \in S} p(s, t) = 1$, i.e. for each state $s$ there is a state $t$ with $p(s, t) > 0$ and $(s, t)$ is in $T$ (by definition).

This embedded Kripke structure describes the same state space, the same transitions (modulo probability distributions) and therefore the same state sequences as its associated MDP (Thm. (2)).

**Theorem 2:** Let $\rho$ be the projection of a path of the MDP of the form $\omega = s_0(j_0, p_0)s_1 \ldots$ to a sequence $\pi = s_0s_1 \ldots$. Then the diagram in Fig. 9 is commutative, i.e.: $\forall \pi : \pi \in \text{Paths}(\kappa(\tau_{MDP})) \Leftrightarrow \exists \omega \in \text{Paths}(\tau_{MDP}) : \rho(\omega) = \pi$

![Fig. 9. Mapping of MDP Traces to Kripke Structure Traces](image)

**Proof:** $\Rightarrow$: $\kappa$ maps a MDP $\tau_{MDP}$ to a Kripke structure $\tau_{Kripke}$ with the same states and for each pair of states $(s, t)$ in the transition relation $T$ of $\tau_{Kripke}$ there exists an action probability distribution pair $(j, p)$ in $\tau_{MDP}$ such that $p(s, t) > 0$ holds (Thm (1)). Therefore for each $\pi = s_0s_1 \ldots$, for every pair $(s_i, s_{i+1}) \in T$ there exists a pair $(j, p)$ such that $p(s_i, s_{i+1}) > 0$ in the MDP and $\omega = s_0(j_0, p_0)s_1 \ldots$ is a path of $\tau_{MDP}$ (Def. (2)) and by its definition, $\rho$ projects $\omega$ onto $\pi$ (as the sequence of states is preserved)

$\Leftarrow$: By contradiction: Assume $\exists \omega \in \text{Paths}(\tau_{MDP}) : \rho(\omega) = \pi \land \pi \notin \text{Paths}(\kappa(\tau_{MDP})).$ This means that for $\omega = s_0(j_0, p_0)s_1 \ldots s_i(j_i, p_i)s_{i+1} \ldots$ there exists $i$ such that $p_i(s_i, s_{i+1}) > 0$ holds (else $\pi$ would be an element of $\text{Paths}(\kappa(\tau_{MDP})))$, see Thm (1)). But if such a pair $(j_i, p_i)$ exists, then $\omega$ is not a path of a MDP (see Def. (2)).

As both the MDP and the embedded Kripke structure describe the same state sequences and, by construction Thm. (1), have the same labeling functions $L$, they also describe the same traces.

2) Transformation into Model-Checkers: We give the transformation for NuSMV [30], an open-source implementation which is the most current and generally available SMV variant. The transformation outlined here is possible for the other variants as well, the concrete syntax differs only slightly.

NuSMV can handle deterministic and non-deterministic transitions, synchronous parallel modules and multiple state variables in every module. State variables are not visible globally, but must be exported and imported explicitly at the instantiation of the respective module. Each state variable is declared via its domain, and its possible initial states are declared via the keyword init. The new values for a state variable in the next time step are declared via the keyword next. It consists of a case statement of activation.
conditions in which each case corresponds to exactly one case of the update rules of the SAML model. For each activation condition either a single value or a set of new values are specified for each state variable of the module. If one value is given, the transition is deterministic, if a set of new values is given, it is a non-deterministic choice between any of the set. Fig. 10 shows the conversion of the first module of the SAML example model.

**MODULE A(V_B1, V_B2)**

VAR
V_A : 0..2;

DEFINE
CASE_3 = !(V_B1 = 0 & V_B2 = 0 | V_B1 = 1 & V_B2 = 1);

ASSIGN
init(V_A) := 0;
next(V_A) := case
V_A = 0 & V_B1 = 0 & V_B2 = 0 : {0, 1};
V_A = 0 & V_B1 = 1 & V_B2 = 1 : 2;
V_A = 0 & CASE_3 : 1;
V_A = 1 : 1;
V_A = 2 : 2;
esac;

Fig. 10. NuSMV module for module A

Fig. 11 shows the transformation of the second module of the SAML example model. Here the state variable v_a is passed explicitly. In contrast to SAML, NuSMV does not allow parallel assignments in a transition. The solution is to introduce a new state variable. This variable is an index to specify which of the possible parallel assignments should be used and is updated completely non-deterministically.

**MODULE B(V_A)**

VAR
V_B1 : 0..1;
V_B2 : 0..1;
SELECTED_ASSIGN : 0..3;

ASSIGN
init(SELECTED_ASSIGN) := {0,1,2,3};
next(SELECTED_ASSIGN) := {0,1,2,3};

init(V_B1) := 0;
next(V_B1) := case
true & SELECTED_ASSIGN = 0 : 0;
true & SELECTED_ASSIGN = 1 : 1;
true & SELECTED_ASSIGN = 2 : 1;
true & SELECTED_ASSIGN = 3 : 1;
esac;

init(V_B2) := 0;
next(V_B2) := case
true & SELECTED_ASSIGN = 0 : 0;
true & SELECTED_ASSIGN = 1 : 0;
true & SELECTED_ASSIGN = 2 : 1;
true & SELECTED_ASSIGN = 3 : 1;
esac;

Fig. 11. NuSMV module for module B

SAML modules are directly mapped to NuSMV modules and SAML state variables directly to NuSMV state variables with the same domain. The differences to NuSMV are the need for explicit passing of visible state variables in the module declaration and the absence of parallel assignments. Non-deterministic choices and probabilistic distributions can always be transformed as shown in Fig. 11, by introducing an explicit enumeration of the different possible parallel assignments per update. In order to get a proper NuSMV model, all the transformed modules must be instantiated in a special main module. Due to space restrictions this has been omitted here.

VI. ANALYSIS OF CASE STUDY

To illustrate the benefit of different analysis foci for safety analysis, consider the following simple example system which was taken from literature [32]. The system measures an input signal (I). The measurement is conducted by two redundant sensors, S1 and S2. The measured signal is then fed into an arithmetic unit (A1) which computes the output (O) signal. In this case the A1 unit is fed with the measured signal from both sensors, in order to compensate for possible differences due to transient sensors errors resulting in a wrongly read signal. In addition if one sensor fails altogether, A1 can still produce an output signal.

If, on the other hand, A1 fails, then the monitor (M) which acts as a watchdog, switches on the second arithmetic unit (A2). In difference to A1, the unit A2 is only connected to S2, i.e. cannot compensate for erroneous signals from this sensor. A hazard of this system can be either a wrongly computed output signal (HVal) or the omission of an output signal (HSig). From the viewpoint of safety analysis, several different questions are of interest here:

- Which failure modes are possible in the system?
- What is the direct effect of the failure modes?
- Which combination of failure modes is critical with respect to one of the hazards?
- Are there temporal dependencies between the critical combination of failure modes?
- What is the probability of the occurrence of a hazard, given failure probabilities or failure rates for the failure modes?

The answer to the first question is a general problem for all safety analysis techniques and is not in the scope of this paper. To find such possible failure modes, approaches like HaZop [33] or failure sensitive specification [34] can be applied. Depending on the type of system components, there may also be list of possible failure modes already available. In this scenario, a variety of failures modes is possible. The sensors can omit a signal (S1FailsSig, S2FailsSig), making it impossible for the arithmetic unit to process data. The

![Fig. 12. System with Spare Redundancy](image-url)
arithmetic units themselves can omit producing output data (A2FailsSig, A1FailsSig). The monitor can fail to detect that situation (MonitorFails), either switching if not necessary or not switching if necessary. The activation of the A2 unit can fail (A2FailsActivate). One possible hazard is “no signal at output (O)” (Hsig).

The failure effect of the failure modes is modeled as follows: If a sensor failure or a failure of an arithmetic unit occurs, there is no signal passed to its successor unit. This unit cannot produce an output signal in return then. If the monitor fails at any time, it switches between any of the two arithmetic units (independent if the activated is operational), if the activation of the A2 unit fails, no operational unit produces any output. Finally, if there is no signal from either A1 or A2 for more than 2 time-steps (one is allowed for switching), then the hazard Hsig occurs. For the analysis of the example case study, all failure modes were modeled as per-time failure modes with the exception of the per-demand failure A2FailsActivate.

After the set of failure modes is specified and their direct effects are integrated into the extended system model, deductive cause consequence analysis (DCCA) [14] (Def. (9)) is used to compute the (inclusion) minimal critical combinations of failure modes that can lead to Hsig. DCCA is proven to be complete, no critical combinations are forgotten. The analysis is also proven to be correct in the sense that the critical combinations can lead to a hazard. DCCA constructs a proof as a witness trace of the model. The proof obligation of DCCA is shown in Eq. (6). It is expressed in CTL temporal logic [35] and states that: “There exists a trace of the model on which only a subset of the failure modes appear before the hazard occurs”. When this holds for a subset Γ of the set of all failure modes ∆, then its failure modes can cause the hazard. If Γ has no critical subset it is a minimal critical set. Criticality is monotone, so normally DCCA is started from single failure modes, skipping all supersets of minimal critical sets.

**Definition 9: DCCA / minimal critical set** A subset Γ of the finite set ∆ of failure modes is critical wrt. a hazard H if

\[ E[\Gamma U H], \text{ where } \Gamma := \bigwedge_{\delta_1 \in \Delta \backslash \Gamma} \delta_1 \quad (6) \]

holds. Γ is minimal critical if is has no proper critical subset.

In the case study the NuSMV [30] model-checker was used to compute the minimal critical sets: Both arithmetic units fail (\{A2FailsSig, A1FailsSig\}), one arithmetic unit and the monitor fails (\{A2FailsSig, MonitorFails\}) or (\{A1FailsSig, MonitorFails\})², the primary unit A1 and the second sensor fail (\{A1FailsSig, S2FailsSig\}), the monitor and the second sensor fail (\{MonitorFails, S2FailsSig\}), both sensors fail (\{S2FailsSig, S1FailsSig\}), the monitor fails and the activation of A2 fails (\{MonitorFails, A2FailsActivate\}) and the primary unit fails and the activation of A2 fails (\{A1FailsSig, A2FailsActivate\}).

²The failure modes \{A1FailsSig, MonitorFails\} are only critical if the monitor fails before the arithmetic unit, this can be automatically computed using temporal ordering analysis [36].

Assuming an error rate of $10^{-2.1}$ for each per-time failure mode and using a temporal resolution $\delta t = 10 ms$, this translates to a per-step failure probability of $p_{fail} = 2.7 \cdot 10^{-8}$ for the per-time failure modes. For the failure mode A2FailsActivate, a per-demand failure probability of $10^{-4}$ is assumed.

For the analysis of the hazard probability, probabilistic reachability analysis as shown in Def. (10) is used [9]. The properties are defined using PCTL probabilistic temporal logic [27], [27]. As the example is a reactive system, only bounded analysis as shown in Eq. (8) is sensible. From the point of safety analysis the worst case is of interest. Computing the maximal probability assures that no possible adversary can produce a system trace with larger probability on which the hazard occurs.

**Definition 10: Probabilistic DCCA (pDCCA) For a MDP $\tau_{MDP}$ and a hazard $H$**

\[ P_{max=?}[true U H] \quad (7) \]

computes the maximal probability that the hazard $H$ occurs on a trace of $\tau_{MDP}$.

\[ P_{max=?}[true U \leq k H] \quad (8) \]

computes the maximal probability that the hazard $H$ occurs within $k$ time units.

Using PRISM, the probability of $H_{sig}$ can be computed for a $k = 360000$ (running time of 1h). The result is shown in Eq. (9). The analysis was conducted on an AMD64 3Ghz and needed ca. 140s to complete. In this case the dominant factor in the analysis time is the large number of iterations, as $k$ matrix multiplications are required. This can very likely be made much faster using new approaches for probabilistic model checking [37], [38].

\[ P(H) = P_{max=?}[true U \leq 36 \cdot 10^{5} H] = 2.964375 \cdot 10^{-17} \quad (9) \]

VII. CONCLUSION AND OUTLOOK

The presented SAML framework allows for different model-based safety analyses on the exact same system model. This increases both the confidence in the analysis results as well reduces the possibilities of errors introduced by the creation of different models. Although the example shown here is small, case-studies with more than $10^7$ states have successfully been analyzed [29] using our quantitative analysis methods.

The analyses were focused solely on safety analysis, but functional aspects as well as other non-functional aspects can also be analyzed, at the moment all properties formulated in (probabilistic) temporal logic like PCTL, CTL or LTL. We showed how a SAML model can be soundly transformed into different analysis tools. The advantage of keeping the SAML framework as simple as possible in order to be able to specify the desired models, helps keeping it as tool-independent as possible. This means that the analysis can benefit from both advances in existing tools like PRISM, MRMC, NuSMV and
Cadence SMV as well as from other tools in which yet different aspects can be analyzable.

Further work will include using SAML as intermediate language for more abstract frameworks, for example for the specification of an explicit error model for different types of failures as in SLIM [19] for the AADL error annex. In addition to the probabilistic transition, the explicit specification of timing would be an interesting extension to SAML.

ACKNOWLEDGMENT

Matthias Güdemann is funded by the German Ministry of Education and Science (BMBF) within the ViERforES project (no. 01IM08003C).

REFERENCES


