Model-Based Safety Analysis of Mode Transitions

Marco Bozzano[1], Peter Munk[2], Markus Schweizer[2], Stefano Tonetta[1], and
✉ Viktória Vozárová[1]

[1]Fondazione Bruno Kessler, Trento, Italy
e-mail: {bozzano, tonettas, vvozarova}@fbk.eu
[2] Robert Bosch GmbH, Research and Advance Engineering, Renningen, Germany
e-mail: {Peter.Munk, Markus.Schweizer}@de.bosch.com

Abstract The verification of safety requirements is fundamental in many safety-critical domains. In order to reach the highest level of required safety assurance, system engineers design components with a variety of safety mechanisms. The resulting potential combination and sequence of operational modes may become very complex and requires automated analysis support.

In this paper, we propose new formal methods, based on minimal cut sets, to generate explanations for operational mode transitions, in terms of causes defined as combinations of basic events, namely faults and recovery actions. The problem is quite subtle, as it requires to consider events occurring before, and in between, the source and target operational modes, identifying those that are necessary to bring the system into the source mode. We implemented the approach on top of the xSAP safety analysis platform, and evaluated it on an industrial design, namely an electronic control unit of a power steering system with redundancy and multiple safety mechanisms.

1 Introduction

The increasing level of autonomy and complexity of networked systems and system of systems in automotive as well as in other safety-critical domains augments the required level of functional safety and reliability of Electronic Components and Systems (ECS) [9]. In turn, the growing requirements in terms of functional safety and reliability push the development of new design technologies to analyze the safety of ECS. Fail-operational architectures include various safety mechanisms such as redundancies and fault detection components inside a single control unit. The interplay of multiple faults and mechanisms for fault masking and fault recovery may become very complex and requires automated methods and tools for its analysis.

In this paper we tackle the problem of analyzing the various faults, or in general events, that may lead a system from an operational mode to another. The system usually runs in nominal mode and switches to different backup or degraded modes upon the occurrence of faults or recovery actions. Due to the presence of different components and overlay of various redundancies and monitors, the system can switch to an operational mode for various reasons. We propose a model-based approach to the analysis of these mode transitions building on symbolic model-based safety analysis techniques for minimal cut sets and fault-tree generation [4].

The problem is quite subtle because the transition from mode m_1 to mode m_2 can be caused by events that occurred before m_1 but due to some propagation have effect with some delay or the effect is enabled by the new operational mode m_1. At the same time, we should not consider the events before m_1 that caused the system to go to m_1. We propose a formulation that takes into account these aspects and reduce the problem to parameter synthesis for temporal logic [7].

We implemented the approach on top of the xSAP tool [2] and evaluated the results on the architecture of an automotive Electronic Control Unit. This includes a dynamically redundant dual channel, each channel with a dual fail-safe core, extended with a watchdog that may trigger the recovery of a passive channel. The results are very useful to understand the interplay of events that cause the mode transitions and show the scalability of the approach.

The rest of this paper is structured as follows. In Section 2 we discuss related work. We describe the case study in Section 3. In Section 4 we discuss some background notions. In Section 5 we discuss our formal approach. The experimental evaluation is presented in Section 6. Finally, we conclude and discuss future work in Section 7.

2 Related Work

The problem addressed in this paper builds upon, and extends previous work on Fault Tree Analysis (FTA), namely generation of minimal cut sets (MCSs) for a given top level event (TLE). The semantics of MCSs is given in terms of fault events occurring on a trace reaching the TLE [4,14]. The problem of computing the cut sets can be reduced to reachability analysis and solved using Binary Decision Diagrams as in [4], or using satisfiability (SAT)-based techniques for parameter synthesis [6]. The region of cut sets can be minimized to obtain the MCSs using classical routines for minimization of Boolean functions [8]. In [3] the SAT-based approach is extended with several enhancements based on the specific features of the problem, such as on-the-fly minimization and layered computation of the MCSs for increasing cardinality.

In this work, the trace-based semantics for MCSs is extended to encompass the case of generic (fault and recovery) events that explain the transitions between different system modes, rather than a TLE. The problem is reduced to parameter synthesis on a property expressed in LTL, and solved using techniques that build upon those in [3].

A qualitative analysis of the EPS case study has been carried out in [1] using FTA. The author performed the analysis by manually inspecting all possible states and transitions, and demonstrated that the order of events causing the mode transitions can be neglected. However, manual analysis is error-prone and does not scale up when additional channels or states are considered. In this paper, we give a formal definition of the problem and solve it using a formal approach, based on model-checking.

In [11] a methodology is presented, based on Hip-Hops, to construct fault trees structured in terms of a set of (critical and non-critical) modes organized into a mode chart. The methodology is focused on the investigation of failure propagation, based on the annotation of system components with their (dynamic) mode-based behavior. In our case, instead, we are interested in synthesizing the mode-based (failure propagation) behavior automatically from a given behavioral model of the system.

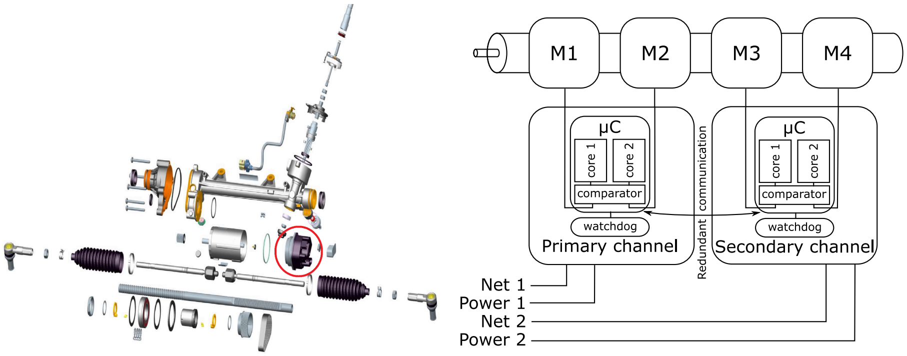


Figure 1: EPS case study: assembly view with the electronic control unit (ECU) circled in red (left) and schematic overview (right).

The concept of events triggering mode transitions is related with the notions of causality as given, e.g., in the theory of counterfactual causality [10]. The latter is defined using structural equations, but can be readily re-formulated for transition systems [5]. However, the notion of causality is more fine-grained, in that it aims at distinguishing the notions of causality and temporal correlation, and addresses concepts such as responsibility and blame. Moreover, we are interested in sets of events that are necessary to explain a mode transition in all possible scenarios, whereas classical causality focuses on identifying such causes in a given scenario of interest. Finally, in our setting a cause may not be sufficient to trigger a mode transition – an additional side condition, a *contingency* in causality terminology, may be needed to make it sufficient.

3 Motivating Case Study

As case study, we selected an electronic power steering (EPS) system designed for highly-automated driving vehicles, as shown in Fig. 1. The system is not only able to support the driver in steering, but also to steer the vehicle without any input from the driver, by receiving steering commands from a redundant vehicle bus. Hence, the EPS system has high safety, reliability, and availability requirements.

3.1 ECU Design

In this case study, we focus only on the electronic control unit (ECU) of the EPS system, circled in red in Fig. 1 (left). A schematic overview of the EPS ECU is given in Fig. 1 (right). The ECU includes two separate channels, named *primary* and *secondary* channel in the following. Each channel has its own and independent power supply and connection to an individual vehicle bus. Both channels can communicate with each other by redundant intra-ECU communication channels. Each channel contains a lock-stepped microcontroller with an external watchdog and is able to drive 2 electric motors. The lock-stepped microcontroller contains two cores that compute the same instructions in parallel. At each cycle, a comparator circuit inside the controller compares the state of both cores. The microcontroller shows fail-silent behavior, so in case the two core states are not equal no result is forwarded. In order to check whether the comparator is working correctly, an external watchdog sends challenges to the comparator and checks the correctness of the response. If the challenge is answered incorrectly or if a timeout error occurs, the entire microcontroller is reset.

3.2 System Modes

Each channel is either in mode *master*, *slave*, or *passive*. In master mode, the channel calculates the torque for its two motors and sends a request to the other channel in slave mode to set the same torque to its connected motors, so all four motors provide the same torque. In slave mode, the channel awaits the torque requests from the other channel and sets the torque as described before. If the torque request is not received, the channel in slave mode has to assume that the other channel has failed silently, hence it becomes master and calculates the required torque itself. Since one channel and its directly connected two motors are sufficient to steer the vehicle, the EPS system is still available even if one channel fails.

3.3 Expected Faults and their effects

A channel has fail-silent behavior, therefore it enters the passive state only when an internal error occurs and it is detected, e.g. by the lock-step comparator or the watchdog. In passive state, the channel does not send any torque to its two motors anymore. A fault in the power supply of a channel leads to it entering the passive mode. When an erroneous or missing message is received from the vehicle bus connected to a channel, the channel switches or stays in slave mode, relying on the torque requests from the other channel in master mode. A fault in the communication between the channels is critical, as the torque requests cannot be exchanged anymore. For this reason, this intra-ECU inter-channel communication is implemented by heterogeneously redundant links. A fault in the microcontroller and its core, respectively, is very likely to be detected by the comparator circuit. A fault in the comparator itself is critical, for this reason it is implemented in hardware directly. In order to ensure the correct functionality of the comparator circuit, a watchdog, which is external to the microcontroller, monitors it by a challenge-response protocol. In case the comparator does not provide the correct response in time, the entire microcontroller is reset by the watchdog. A fault in the watchdog itself is critical again, as it potentially resets the microcontroller.

The individual faults have different occurrence probabilities, depending e.g. on the complexity of the hardware or the employed level of redundancy. In order to argue the safety of the entire EPS system and its ECU specifically, it is indispensable to analyze the combination and probability of faults that lead to unwanted behavior of the system. In general, the EPS system can exhibit unwanted behavior whenever no channel is in master mode and one of them is in slave mode. Given three modes per channel, overall nine modes exist in the system. Not all nine system modes are equally critical, e.g. one channel being in master mode and the other channel being in passive mode is acceptable for a specific duration. On the contrary, both channels being in master mode and potentially calculating opposite torque request is very critical and it potentially leads to steering in the wrong direction and even hinders the driver to overrule the system.

4 Background

In this section we present some background, in particular we introduce transitions system, temporal logic, model checking, parameter synthesis, and minimal cut sets.

4.1 Symbolic Transition Systems

The system under analysis is a reactive system, whose behavior is characterized by a (possibly infinite) sequence of state changes triggered by events. In this paper, we adopt a standard symbolic representation of the system, where the system states are represented by a finite set V of variables and the state transitions by symbolic formulas that specify how the values of V change [13]. This is usually obtained by using a copy v' of each variable $v \in V$ to represent the next value of v after a transition. We denote by V' the set of next versions v' of the variables in V. We also use a finite set E of event variables to label the transitions and represent the events that triggered a state change. For simplicity, we assume that the variables have all a Boolean domain, but this can be easily lifted and the tool implementation of the approach considers also more complex and infinite-domain variables.

Formally, a Transition System (TS) is a tuple $S = \langle V, E, I, T\rangle$ where:

- V is a set of state variables;
- E is a set of event variables;
- I is a formula over V, representing the initial states;
- T is a formula over $V \cup E \cup V'$, representing the transitions.

A state of S is an assignment to the variables in V. Similarly, an event is an assignment to the variables in E. A trace of S is an infinite sequence $\sigma = s_0, e_0, s_1, e_1, \ldots$ of states and events such that $s_0 \models I$ and $s_i, e_i, s_{i+1} \models T$ for every $i \geq 0$.

4.2 LTL model checking

We use Linear-time Temporal Logic (LTL) [15] with future and past operators (see for example [12]) to represent sets of traces. Given a TS $S = \langle V, E, I, T\rangle$, the set of Linear Temporal Logic (LTL) formulas is inductively defined as

$$\varphi ::= p \bigm| \neg\varphi \bigm| \varphi \vee \varphi \bigm| \mathbf{X}\,\varphi \bigm| \varphi\,\mathbf{U}\,\varphi \bigm| \mathbf{Y}\varphi \bigm| \varphi\,\mathbf{S}\,\varphi$$

with $p \in V \cup E$. Here $\mathbf{X}$ stands for *next*, $\mathbf{U}$ for *until*, $\mathbf{Y}$ for *previous*, and $\mathbf{S}$ for *since*. Other logical constants and operators like $\top$, $\bot$, $\wedge$, $\rightarrow$ and $\leftrightarrow$ are used as syntactic sugar with the standard meaning. The following abbreviations for temporal operators are also used: $\mathbf{F}\,\varphi := \top\mathbf{U}\,\varphi$, $\mathbf{G}\,\varphi := \neg\mathbf{F}\,\neg\varphi$, $\mathbf{O}\,\varphi := \top\mathbf{S}\,\varphi$, $\mathbf{H}\,\varphi := \neg\mathbf{O}\,\neg\varphi$, $\mathbf{Z}\varphi := \neg\mathbf{Y}\neg\varphi$.

Given a trace $\sigma = s_0, e_0, s_1, e_1, \ldots$ of S and $i \geq 0$, we define the relation $\sigma, i \models \varphi$ as follows:

- if $\varphi = p \in V$, then $\sigma, i \models \varphi$ iff $s_i \models p$
- if $\varphi = p \in E$, then $\sigma, i \models \varphi$ iff $e_i \models p$
- if $\varphi = \neg\phi$, then $\sigma, i \models \varphi$ iff $\sigma, i \not\models \phi$
- if $\varphi = \phi \vee \psi$, then $\sigma, i \models \varphi$ iff $\sigma, i \models \phi$ or $\sigma, i \models \psi$
- if $\varphi = \mathbf{X}\phi$, then $\sigma, i \models \varphi$ iff $\sigma, i+1 \models \phi$
- if $\varphi = \phi\,\mathbf{U}\psi$, then $\sigma, i \models \varphi$ iff for some $j \geq i$, $\sigma, j \models \psi$ and for all $i \leq k < j$, $\sigma, k \models \phi$.
- if $\varphi = \mathbf{Y}\phi$, then $\sigma, i \models \varphi$ iff $i > 0$ and $\sigma, i-1 \models \phi$
- if $\varphi = \phi\mathbf{S}\psi$, then $\sigma, i \models \varphi$ iff for some j, $0 \leq j \leq i$, $\sigma, j \models \psi$ and for all $j < k \leq i$, $\sigma, k \models \phi$.

The (universal) model checking problem is the problem to check if $\sigma, 0 \models \varphi$ holds for every trace σ of S (denoted by $S \models_\forall \varphi$ or simply $S \models \varphi$). The existential model checking problem is the dual problem of checking if $\sigma, 0 \models \varphi$ holds for some trace σ of S (denoted by $S \models_\exists \varphi$). Note that $S \models_\exists \phi$ iff $S \not\models \neg\phi$.

4.3 Parameter Synthesis

In parametric systems, formulas can include also *parameters*, which are rigid symbols whose value does not change along the execution of the system [7]. Let U be the set of parameters. A *parameter valuation* is an assignment to the parameters. Given a propositional or an LTL formula ϕ and a parameter valuation γ, we denote by $\gamma(\phi)$ the formula obtained from ϕ by replacing each parameter in U with the assignment given by γ.

A *parametric transition system S* is a tuple $S = \langle U, V, E, I, T\rangle$ where U is the set of parameters, V is the set of state variables, E is the set of event variables, $I(U, V)$ is the initial formula, and $T(U, V, E, V')$ is the transition formula. Each parameter valuation γ induces a transition system $S_\gamma = \langle V, E, \gamma(I), \gamma(T)\rangle$.

In the scope of this paper, we are interested in the parameter synthesis for LTL existential model checking, i.e., given an LTL formula φ over $U \cup V \cup E$, the problem of finding all parameter valuations γ such that $S_\gamma \models_\exists \gamma(\varphi)$. We denote by $\rho(U, S, \varphi)$ the set of all such parameter evaluations. This set can be computed effectively with a sequence of incremental model checking problems [7].

4.4 Minimal Cut Sets

Minimal Cut Sets (MCS) analysis produces all possible configurations of system faults (called *fault configurations*) that cause the reachability of an unwanted condition, called the Top Level Event (TLE). More formally, given a transition system $\langle V, E, I, T\rangle$ and a set of faults represented as event variables $\mathcal{F} \subseteq E$, we call *fault configuration* a subset $FC \subseteq \mathcal{F}$.

A *cut set* represents a fault configuration that may cause the top event. Formally, we generalize the definition in [4] as follows. Let $S = \langle V, E, I, T\rangle$ be a TS and let TLE be a propositional formula over V. We say that FC is a cut set of TLE in S, written $FC \in CS(S, TLE, \mathcal{F})$, iff there exists a trace σ of S such that:

1. $\sigma, j \models TLE$ for some $j \geq 0$;

2. $FC \subseteq \mathcal{F}$ and for all $f \in FC$ there exists i, $0 \leq i < j$ such that $\sigma, i \models f$.

Intuitively, a cut set corresponds to the set of faults that occur along a trace reaching the TLE. *Minimal cut sets* (MCSs), written $MCS(S, TLE, \mathcal{F})$, are those that are minimal in terms of faults: $MCS(S, TLE, \mathcal{F}) = \{cs \in CS(S, TLE, \mathcal{F}) \mid \forall cs' \in CS(S, TLE, \mathcal{F})\ (cs' \subseteq cs \rightarrow cs' = cs)\}$. When S and $\mathcal{F}$ are clear from the context, we just use the notation $CS(TLE)$ and $MCS(TLE)$.

In practice, MCS are of interest since they represent the simpler (and more probable) explanations for a given TLE. The *monotonicity assumption* (i.e, if cs is a cut set, then any superset $cs' \supseteq cs$ is also a cut set) is commonly adopted, since most systems are monotonic and for non-monotonic systems, the assumption leads to a conservative (and accurate) over-approximation of the unreliability of the TLE. Non-monotonic analysis can be addressed by generalizing the concept of MCS to the one of *prime implicant* [8].

4.5 Computing MCSs Using Parameter Synthesis

Given a transition system $S = \langle V, E, I, T \rangle$ and a set of event variables $\mathcal{F} \subseteq E$, the region of cut sets can be computed via parameter synthesis [3]. Let us consider a parameter p_e for every event $e \in \mathcal{F}$ and the LTL formula $\Psi_{TLE} := (\bigwedge_{e \in \mathcal{F}} (e \rightarrow p_e))\ \mathbf{U}\ TLE$ (see also similar approach in [14]). Then the set of cut sets is given by $\rho(U, S, \Psi_{TLE})$.

The set of MCSs can be computed as the set of minimal such valuations, i.e. the set of valuations $\gamma \in \rho(U)$ such that for each $\gamma' \in \rho(U)$, $\gamma' \subseteq \gamma$ implies $\gamma' = \gamma$ (where we define $\gamma' \subseteq \gamma$ iff $\gamma'(u)$ implies $\gamma(u)$ for each $u \in U$). This can be computed with standard BDD-based operations.

5 Formal Problem and Solution

5.1 Formalization of Modes and Mode Transitions

An operational mode can be considered from the formal point of view as a macro state, i.e. a set of concrete states. For example, in the EPS case study described above, the *master-slave* mode, where the primary channel is in master mode and the backup channel is in slave mode, includes various states where the power may or may not be supplied to the channels, the data has been provided or not, the cores are processing the data, the comparator state represents the consistency of the cores' output, etc.

On this line, a mode transition is achieved with a sequence of state transitions. For example, in order to switch from *master-slave* to *master-passive*, the system performs different state transitions, where for example a core of the backup channel fails, the comparator silences the output torque, and the channel goes to passive mode.

Formally, we define a mode of a system $S = \langle V, E, I, T \rangle$ as a set of states of S. A mode can be therefore represented by a propositional formula over the state variables V. With abuse of notation, given a formula m over V, the mode m refers to the set of states satisfying m.

Given two modes m_1 and m_2, a mode transition from m_1 to m_2 is a sequence of $s_0, \ldots, s_n$ such that $n > 0$ and there exists a trace σ of S and $i \geq 0$ such that

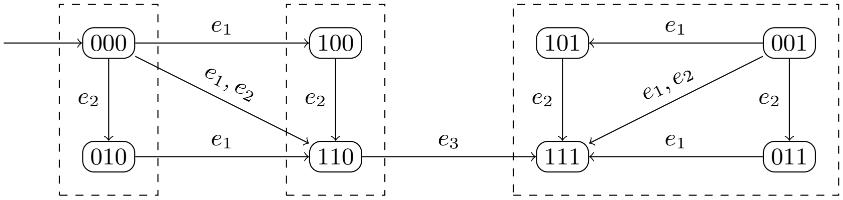


Figure 2: States and transitions for the system in Example 1.

- for k, $0 \leq k \leq n$, $\sigma_{i+k} = s_k$ (the sequence is part of the trace σ of S);
- for k, $0 \leq k < n$, $s_k \models m_1$ and $s_n \models m_2$ (the sequence leads from m_1 to m_2);
- $i = 0$ or $\sigma_{i-1} \models \neg m_1$ (the sequence is maximal as it is either the first mode transition of σ or is preceded by another mode transition leading to m_1).

5.2 Model Checking Mode Transitions

It is easy to prove that S has a mode transition from m_1 to m_2 (denoted by $S \models_\exists m_1 \Rightarrow m_2$) iff $S \models_\exists \mathbf{F}(m_1 \wedge \mathbf{X} m_2)$. In fact, one can see that the definition is one-to-one with the LTL formula $\mathbf{F}(\mathbf{Z}\neg m_1 \wedge m_1 \wedge \mathbf{X}(m_1 \ \mathbf{U} m_2))$. We proved also with a model checker that this formula is equivalent to $\mathbf{F}(\mathbf{Z}\neg m_1 \wedge m_1 \ \mathbf{U}(m_1 \wedge \mathbf{X} m_2))$ and to $\mathbf{F}(m_1 \wedge \mathbf{X} m_2)$.

We can also generate with parameter synthesis the set of events that occur in mode transitions between m_1 and m_2. Let us introduce a parameter p_e for every event $e \in E$ and define the formula ψ_E as $\psi_E := \bigwedge_{e \in E} e \rightarrow p_e$. Then, we build the LTL formula: $\mathbf{F}((\mathbf{Z}\neg m_1) \wedge ((m_1 \wedge \psi_E) \ \mathbf{U}(m_1 \wedge \psi_E \wedge \mathbf{X} m_2)))$.

5.3 Discussion

The analysis discussed in the previous section is quite related to the problem of understanding which events cause a mode transition. A deeper look at the problem shows that it is not what we need.

Example 1. Consider for example the transition system shown in Figure 2 formalized by $\langle \{b_1, b_2, b_3\}, \{e_1, e_2, e_3\}, \neg b_1 \wedge \neg b_2 \wedge \neg b_3, T_1 \rangle$, where T_1 is a disjunction of conjuctions representing the set of transitions (for example, the transition from state 000 to state 100 is represented by $\neg b_0 \wedge \neg b_1 \wedge \neg b_2 \wedge e_1 \wedge \neg e_2 \wedge \neg e_3 \wedge b_0' \wedge \neg b_1' \wedge \neg b_2'$). In the figure, the states are labeled by the value of the variables b_1, b_2, b_3. Thus for example, the state 001 assigns b_1 and b_2 to false and b_3 to true.

Suppose we are interested in the transitions from mode $m_1 = b_1 \wedge \neg b_3$ and $m_2 = b_3$, which correspond to the central and right dashed boxes respectively. The mode transitions are two: $110, 111$ and $100, 110, 111$. The events that occur in these transitions are e_3 and e_2, e_3. Thus, it seems that the cause of the mode transition is e_3 (since it labels the only incoming transition into m_2). However, also e_2 is necessary to reach m_2, but not necessarily to reach m_1: in some traces e_2 occurs before entering mode m_1. Hence this interpretation is not captured by the definition in Section 5.2.

5.4 Problem Definition

Intuitively, given a transition system $\langle V, E, I, T\rangle$ and two modes m_1 and m_2, we are interested in the sets of events in E that are necessary to go from m_1 to m_2. We call such set of events *Minimal Transition Cut Set* (MTCS) for $m_1 \Rightarrow m_2$ and we denote by $\mathit{MTCS}(m_1, m_2)$ the set of all MTCSs for $m_1 \Rightarrow m_2$. For simplicity, we assume that an event can occur only once. The framework can be extended to consider multiple occurrences of the same event. Note that: 1) a MTCS for $m_1 \Rightarrow m_2$ should not contain the events needed to reach m_1; 2) the event in a MTCS for $m_1 \Rightarrow m_2$ may occur even before m_1.

We formalize the definition of Transition Cut Sets (TCSs) and MTCSs as follows:

Definition 1. *$F \in \mathit{TCS}(m_1, m_2)$ iff $F \subseteq E$ and there exist a trace σ and $i, j \in \mathbb{N}$ s.t.*

1. *$i < j$, $\sigma(j) \models m_2$, and $\sigma(k) \models m_1$ for all k, $i \leq k < j$ (i.e., it contains a mode transition from m_1 to m_2);*
2. *there exists $C \in MCS(m_1)$ such that $C \cap F = \emptyset$ and for each $e \in C$ there exists k, $0 \leq k < i$, such that $\sigma(k) \models e$ (i.e., F does not contain a MCS necessary to reach m_1);*
3. *for each $e \in E \setminus C$, if there exists k, $0 \leq k < j$, such that $\sigma(k) \models e$, then $e \in F$ (i.e., F contains all other events occurring until m_2).*

The set $\mathit{MTCS}(m_1, m_2)$ is the set of cut sets in $\mathit{TCS}(m_1, m_2)$ that are minimal: $\mathit{MTCS}(m_1, m_2) := \{F \in \mathit{TCS}(m_1, m_2) \mid \forall F' \in \mathit{TCS}(m_1, m_2)\ (F' \subseteq F \rightarrow F' = F)\}$

5.5 Solution Based on Parameter Synthesis

In this section, we reduce the problem of finding $\mathit{MTCS}(m_1, m_2)$ to a parameter synthesis problem. We first compute $MCS(m_1)$. We introduce a parameter p_e for every event $e \in E$. Finally, we build the LTL formula:

$$\Psi(m_1, m_2) := \bigvee_{C \in MCS(m_1)} \mathbf{F}(m_1 \wedge \mathbf{Y} \bigwedge_{f \in C} \mathbf{O} f \wedge \mathbf{X}(m_1\ \mathbf{U}(m_2 \wedge \mathbf{YH} \bigwedge_{e \notin C} e \rightarrow p_e)))$$

Theorem 1. *Given a TS $S = \langle V, E, I, T\rangle$ and two modes m_1 and m_2,*

$$\mathit{TCS}(m_1, m_2) = \{F \subseteq E \mid S \models_\exists \gamma_F(\Psi(m_1, m_2))\}$$

where γ_F is an assignment to parameters defined as follows: $\gamma_F(p_e) = \top$ iff $e \in F$.

Proof. Given $F \subseteq E$, we prove that $F \in \mathit{TCS}(m_1, m_2)$ iff $S \models_\exists \gamma_F(\Psi(m_1, m_2))$.

Note that, for any trace σ of S, $\sigma \models \gamma_F(\Psi(m_1, m_2))$ iff there exists $C \in MCS(m_1)$ and $\sigma \models \mathbf{F}(m_1 \wedge \mathbf{Y} \bigwedge_{f \in C} \mathbf{O} f \wedge \mathbf{X}(m_1\ \mathbf{U}(m_2 \wedge \mathbf{YH} \bigwedge_{e \notin C} e \rightarrow \gamma_F(p_e))))$.

Thus, $\sigma \models \gamma_F(\Psi(m_1, m_2))$ iff there exists $C \in MCS(m_1)$, $i \geq 0$ such that $\sigma, i \models m_1$ and $\sigma, i \models \mathbf{Y} \bigwedge_{f \in C} \mathbf{O} f$, $\sigma, i+1 \models m_1\ \mathbf{U}(m_2 \wedge \mathbf{YH} \bigwedge_{e \notin C} e \rightarrow \gamma_F(p_e))$.

Thus, $\sigma \models \gamma_F(\Psi(m_1, m_2))$ iff there exists $C \in MCS(m_1)$, $i, j \geq 0$ such that 1) $i < j$ and $\sigma, j \models m_1$ and for all k, $i \leq k < j$, $\sigma, i \models m_1$; 2) $\sigma, i \models \mathbf{Y} \bigwedge_{f \in C} \mathbf{O} f$, 3) $\sigma, j \mathbf{YH} \bigwedge_{e \notin C} e \rightarrow \gamma_F(p_e))$. These are the three conditions of Definition 1. In fact, 3) holds iff $\sigma, k \models e$ for some k, $0 \leq k < j$ implies $\gamma_F(p_e))$.

Once we obtain the set tcs of Transition Cut Sets, we can compute the minimal ones (*MTCS*) as described in Section 4.5.

6 Experimental Evaluation

6.1 Implementation

We have implemented the solution for computing *MTCS* described in Section 5.5 as a command in the xSAP tool [2]. A model in xSAP is written in the SMV language; it can be manually specified or it can be the result of fault injection (the functionality to automatically extend a nominal model with the fault specification – see [2] for more details). Modes can be specified as Boolean expressions, or implicitly as a set of discrete domain state variables (in the latter case, modes correspond to all the possible evaluations of the given variables). The user can either choose to compute *MTCS* for one pair of given modes m_1 and m_2 or for all pairs of distinct modes taken from a given set of modes.

For each event to be considered in the analysis, a corresponding parameter is created. For each mode m_1, $MCS(m_1)$ is computed using parameter synthesis and stored for the computation of *MTCS*(m_1, m_2), for all target modes m_2. For each pair of modes $\langle m_1, m_2 \rangle$, the LTL formula $\Psi(m_1, m_2)$ described in Section 5.5 is constructed and used for the parameter synthesis. The output of the parameter synthesis problem is a region, i.e. a Boolean formula over the set of parameters. Each parameter is replaced by the corresponding event and the corresponding minimal models are computed and printed. The command provides an option to print all modes and transitions in dot format.

6.2 Application to the EPS Case Study

We modeled the EPS system informally described in Section 3 in SMV language[1]. We separately defined the nominal model and the xSAP fault extension instructions. Then, using fault injection, we created the extended (faulty) model, on which we ran the *MTCS* computation routine. We created two variants of the EPS (nominal) models, a simple and a complex one. The simple model does not contain internal components of the channels. The behavior of the model is also simplified by ignoring the possibility of a channel reset. The complex model, on the other hand, also models cores, a comparator, a watchdog and the reset action of the channels. We first focus on the simple model to demonstrate the functionality of our approach. Then, we analyze its scalability using the complex model.

The simple model is composed of two channels *pd* and *sd*, the energy supply and vehicle bus for each channel *pdEnergy*, *sdEnergy*, *pdBus*, *sdBus*, and a redundant communication *com*. The modules representing these components interact as described in Section 3. In our analysis, we are interested in all events that can cause the system mode to change, namely the fault events described in Section 3 and the take-over of a channel in *slave* mode (i.e., when it fails to receive a torque request from the other channel and assumes that the other channel has failed). The list of all events for the simple model is shown in the middle column of Table 1.

We carried out the *MTCS* analysis on all system modes of the simple model. The modes, along with their criticality, are shown in Fig. 3. We ran the parameter synthesis routine for all pairs of distinct modes and obtained a set of minimal cut sets over

[1] Available at https://es-static.fbk.eu/people/vvozarova/TransitionAnalysis/

Table 1: The events of the EPS system.

	simple model	**complex model**
energy supply faults	*pdEnergy.fault* *sdEnergy.fault*	*pdEnergy.fault* *sdEnergy.fault*
vehicle bus faults	*pdBus.fault* *sdBus.fault*	*pdBus.fault* *sdBus.fault*
communication faults	*com.request_to_pd_fault* *com.request_to_sd_fault*	*com.can.request_to_pd_fault* *com.can.request_to_sd_fault* *com.uart.request_to_pd_fault* *com.uart.request_to_sd_fault*
channel faults	*pd.fault* *sd.fault*	*pd.core1.fault* *pd.core2.fault* *pd.comparator.compare_fault* *pd.comparator.forward_fault* *sd.core1.fault* *sd.core2.fault* *sd.comparator.compare_fault* *sd.comparator.forward_fault*
channel take-over recovery	*pd.takes_over* *sd.takes_over*	*pd.takes_over* *sd.takes_over*
channel reset recovery		*pd.reset* *sd.reset*

the events. For illustration purposes, Fig. 4a shows a few selected transitions between modes *master-slave*, *slave-master*, *master-master* and *slave-slave*. The edge labels correspond to the sets of events found by our analysis for the respective transitions. The graph shows that *pdBus.fault* is necessary to reach *slave-master* mode. In case no other fault occurs, the *sd* channel takes over in the following cycle. If the communication link to the *sd* channel fails at the same time as *pdBus*, the system reaches *slave-master* mode in one cycle. If the communication fails, the *sd* channel wrongly assumes that *pd* has failed and goes to the critical *master-master* mode.

Notice that if we used the formula presented in Sec. 5.2, that monitors only events that occur in m_1, different cut sets would be found. This is possible because some faults take one cycle to propagate. For example, there is a sequence of mode transitions containing transition from *master-slave* to *slave-slave* on which no fault occurs. The sequence is visualized in Fig. 4b. The effect of each fault is visible in the next cycle. The communication fault causes *sd* to go to *master*. The *sd* vehicle bus fault causes

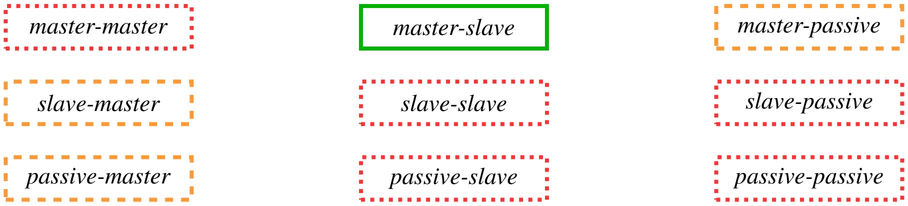
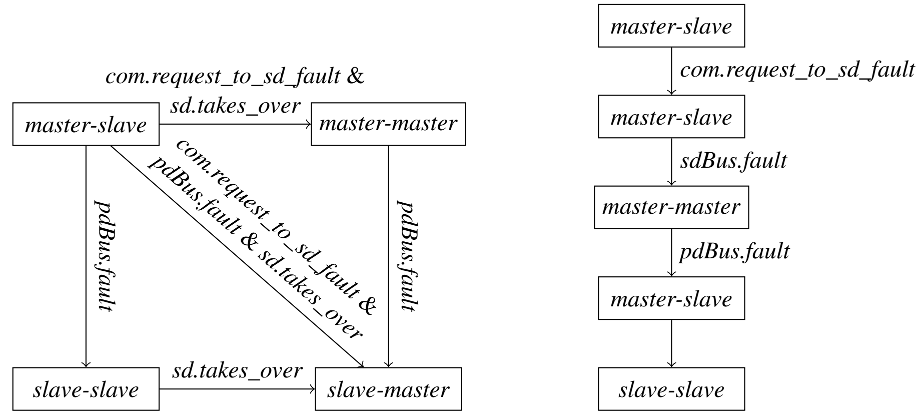


Figure 3: Possible combinations of channel modes. The green mode (solid border) is a nominal functional mode. The orange modes (dashed) are modes with degraded nominal function, but acceptable for a specific duration. The red modes (dotted) are critical and can lead to erroneous behaviour.



(a) Selected transitions between modes *master-slave*, *slave-master*, *master-master* and *slave-slave* with *MTCS* found by our procedure.

(b) Sequence of mode transitions, where in the last transition no fault occurs.

Figure 4: Found *MTCS* (left) and occured events (right) in selected mode transitions.

sd to go back to slave, analogously *pd* vehicle bus fault causes *pd* to go to slave. As a result, only one minimal cut set for *master-slave* to *slave-slave* transition is found, and that is an empty set.

To test the scalability of our procedure, we created a more complex model with more detailed communication and channel. Specifically, we modeled the redundancy of the communication by introducing two submodules *com.can* and *com.uart* with the same functionality as the original module. The communication fails only if both submodules fail. The channel is extended by adding two core modules *core1* and *core2*, *comparator* and *watchdog*. The comparator ensures that if a core fails, the channel goes to *passive* mode. However, if the comparator is faulty, the channel can either wrongly stay in the nominal mode or go to *passive* even when both cores are working correctly. If the watchdog recognizes that either a core or the comparator is faulty, it resets the channel to its initial mode. The list of all events is given in the last column of Table 1. The communication faults are replaced by faults in *com.can* and *com.uart*, the channel faults are replaced by core and comparator faults, and we additionally monitor the reset event of the channel.

6.3 Scalability Results

We tested the implemented procedure for both simple and complex model. The simple model contains 10 events and 7 nominal modules (more modules are introduced after the fault extension). The complex model contains 20 events and 17 nominal modules. We ran the experiments on a machine with Intel(R) Core(TM) i5 CPU and 16GB RAM. The results are in Table 2. The results show that the procedure is applicable for models with many events and complex behaviour. Table 3 shows numbers of found minimal cut sets and their cardinality for all mode transitions.

Table 2: Outcome of the *MTCS* analysis for both the simple and complex models. We report used memory and time, and the number of generated *MTCS* for all transitions between distinct modes (72 in total).

simple model			**complex model**		
time (s)	mem (MB)	MTCSs	time (s)	mem (MB)	MTCSs
56.56	392.1	63	621.84	1047.0	354

Table 3: Number of *MTCS* found for each transition from one mode (left) to another (top) for the EPS system. Cells with dash '-' are self loops on which the analysis was skipped. Cells with 'x' are transitions with no cut sets found (the transition is not feasible). The number of cut sets is followed by the cardinality of the sets in parentheses.

simple model

	MM	MS	MP	SM	SS	SP	PM	PS	PP
MM	-	1 (1)	2 (1)	1 (2)	2 (2)	2 (2)	2 (1)	2 (2)	4 (2)
MS	1 (2)	-	2 (1)	1 (3)	1 (1)	2 (2)	2 (3)	2 (1)	4 (2)
MP	x	x	-	x	x	1 (1)	x	x	2 (1)
SM	x	x	x	-	1 (1)	2 (1)	2 (1)	2 (2)	4 (2)
SS	x	x	x	1 (1)	-	2 (1)	2 (2)	2 (1)	4 (2)
SP	x	x	x	x	x	-	x	x	2 (1)
PM	x	x	x	x	x	x	-	1 (1)	2 (1)
PS	x	x	x	x	x	x	1 (1)	-	2 (1)
PP	x	x	x	x	x	x	x	x	-

complex model

	MM	MS	MP	SM	SS	SP	PM	PS	PP
MM	-	1 (1), 1 (2), 2 (3)	4 (1)	1 (0)	1 (1), 1 (2), 2 (3)	4 (1)	1 (0)	1 (1), 1 (2), 2 (3)	4 (1)
MS	1 (3), 4 (4)	-	4 (1)	2 (4), 2 (5)	1 (1)	4 (2)	5 (4), 2 (5)	4 (1)	16 (2)
MP	x	4 (2)	-	x	4 (3)	1 (1)	x	16 (3)	4 (1)
SM	1 (2), 2 (3)	1 (3), 3 (4), 4 (5), 4 (6)	4 (3), 8 (4)	-	1 (1), 1 (2), 2 (3)	4 (1)	4 (1)	4 (2), 4 (3), 8 (4)	16 (2)
SS	1 (3), 2 (4)	1 (2), 2 (3)	4 (3), 8 (4)	1 (1)	-	4 (1)	4 (2)	4 (1)	16 (2)
SP	x	4 (4), 8 (5)	1 (2), 2 (3)	x	4 (2)	-	x	16 (3)	4 (1)
PM	4 (1)	4 (2), 4 (3), 8 (4)	16 (2)	x	x	x	-	1 (1), 1 (2), 2 (3)	4 (1)
PS	4 (3)	4 (2)	16 (3)	x	x	x	1 (1)	-	4 (1)
PP	x	16 (4)	4 (2)	x	x	x	x	4 (2)	-

7 Conclusions

In this paper, we extended model-based safety analysis techniques to consider the transition between operational modes in complex systems. We propose new techniques based on parameter synthesis and symbolic model checking. We evaluated the approach in an industrial automotive case study describing the architecture of an ECU implementing multiple safety mechanisms for functional safety.

The directions for future development are manifold: 1. to investigate optimization techniques to increase the scalability; 2. to extend the method to consider the negation of events (when an event must not occur in the mode transition, thus going beyond the monotonic case and MCS); 3. to extend the method to consider multiple occurences of an event; 4. to extend the method to consider more general notions of causality; 5. to investigate how ordering of events influences mode transitions; 6. to embed the techniques in system and safety engineering processes involving the design of fault detection and recovery components and the specification of safety contracts on components.

References

1. Abele, A.: Transformation of a state description into a qualitative fault tree. In: Praxisforum Fehlerbaumanalyse & Co. (2019)
2. Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., Zampedri, G.: The xSAP Safety Analysis Platform. In: Proc. TACAS. LNCS, vol. 9636, pp. 533–539 (2016)
3. Bozzano, M., Cimatti, A., Mattarei, C., Griggio, A.: Efficient Anytime Techniques for Model-Based Safety Analysis. In: CAV. pp. 603–621 (2015)
4. Bozzano, M., Cimatti, A., Tapparo, F.: Symbolic Fault Tree Analysis for Reactive Systems. In: Proc. ATVA. LNCS, vol. 4762, pp. 162–176. Springer (2007)
5. Caltais, G., Leue, S., Mousavi, M.R.: (De-)Composing Causality in Labeled Transition Systems. In: Proc. CREST Workshop (2016)
6. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Parameter synthesis with IC3. In: Proceedings of FMCAD. pp. 165–168. IEEE (2013)
7. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Parameter synthesis with IC3. In: Formal Methods in Computer-Aided Design, FMCAD 2013. pp. 165–168. IEEE (2013)
8. Coudert, O., Madre, J.C.: Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions. In: Proc. Design Automation Conference (DAC 1992). pp. 36–39. IEEE Computer Society Press (1992)
9. ECSEL-JU: Multi-Annual Strategic Plan (MASP) (2020), Private Members Board of the ECSEL Joint Undertaking - ECSEL GB 2019.134
10. Halpern, J.: A modification of the Halpern-Pearl definition of causality. In: Proc. IJCAI. pp. 3022–3033 (2015)
11. Kabir, S., Papadopoulos, Y., Walker, M., Parker, D., Aizpurua, J.I., Lampe, J., Rüde, E.: A Model-Based Extension to HiP-HOPS for Dynamic Fault Propagation Studies. In: Proc. IMBSA. Lecture Notes in Computer Science, vol. 10437, pp. 163–178. Springer (2017)
12. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, New York, US (1992). https://doi.org/10.1007/978-1-4612-0931-7
13. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic (1993)
14. Ortmeier, F., Reif, W., Schellhorn, G.: Deductive cause-consequence analysis (dcca). IFAC Proceedings Volumes **38**(1), 62–67 (2005)
15. Pnueli, A.: The temporal logic of programs. In: Proc. SFCS. pp. 46–57 (1977)