A Hybrid Model Checking and Theorem Proving based Approach for Fault Tree Analysis

Shahid Khan[1,2], Waqar Ahmad[1] and Osman Hasan[1]
[1]*Sch. of Elect. Engg. and Computer Sc., National University of Sciences and Technology (NUST), Islamabad, Pakistan*
[2]*Software Modeling and Verification, RWTH Aachen UniversityAachen, Germany*
{*shahid.khan1,waqar.ahmad, osman.hasan*}*@seecs.nust.edu.pk*

***Abstract*—Static fault trees (SFT)s are used for conducting dependability analysis and thus are extensively utilized in various functional safety standards defined by the IEC and ISO for automotive applications. Traditionally, SFTs are manually developed by domain experts through a very cumbersome and error prone process. Once the SFT is available, quantitative fault tree analysis (FTA) is carried out using analytical approaches, e.g., probabilistic model checking and theorem proving. While the former is limited to exponential distributions, the latter can analyze fault trees (FT)s with arbitrary probability distributions. This paper proposes to combine model checking-based automatic FT generation and theorem proving based FTA to ensure a more rigorous and complete approach for FTA. The proposed approach particularly utilizes 1) the XSAP safety assessment platform to automatically generate SFTs from a functionally verified system model and 2) the HOL4 theorem prover to analyze the XSAP-generated SFTs. The usefulness of the approach is illustrated using the case study of an automated vehicle system.**

***Index Terms*—Reliability, Dependability, Formal Verification.**

I. Introduction

Static fault tree (SFT) can be categorized as a modeling language to carry out dependability studies of safety critical systems [21]. It is a top-down approach where a fault tree (FT) is developed for an undesired top level event (TLE). Traditionally, SFTs are manually developed by reliability engineers in consultation with domain experts. This process is cumbersome because a new FT is developed for each TLE, and a modification in system design leads to the repetition of the whole process. Moreover, the process becomes more error prone with the increase in the size of the FTs. SFTs are either developed on paper or using computer-based tools, e.g.,ReliaSoft and Isograph's FaultTree+.

Fault tree analysis (FTA) is performed through simulation-based or analytical methods. Simulation-based methods can analyze FTs with arbitrary probability distributions. However, simulation results lack hard guarantees and proving the fact that simulation tools are bug free is nontrivial [11]. Alternatively, formal methods, like binary decision diagrams (BDD)s, probabilistic model checking and theorem proving [21] have been used for FTA. While BDDs are quite efficient for SFT analysis, they cannot compute exact times to failure, e.g., mean time to first failure (MTTFF). Probabilistic model checking-based analysis is limited to SFTs with exponential distributions associated to their basic events [17].

Theorem proving-based FTA, on the other hand, leverages upon the high expressiveness of higher-order logic (HOL) and the inherent soundness of theorem proving. A HOL formalization of probability theory [18] has been used to formalize the foundational FT gates, e.g., AND, OR, VOT, NAND, NOR, XOR, NOT [1]. This framework is used for the analysis of many real-world applications, including satellite solar arrays [2] and air traffic communication gateways [1]. This framework is further extended to dynamic fault trees by formalizing dynamic gates including PAND, SPARE and FDEP [15].

Automatic fault tree generation from system models is also considered in the literature [5]. Classical approaches exist to generate FTs from system descriptions captured through the system modeling language (SysML) [19] or unified modeling language (UML) [13]. Another prominent approach is to use model checking [3] and satisfiability modulo theories [4] to generate FTs from functionally verified system models. One such approach is implemented in the XSAP tool [7] that is a part of the COMPASS toolset [8]. This framework supports quantitative analysis through probabilistic model checkers, e.g., Storm [16]. It is used to solve many real-world applications including a wheel braking system [9].

Combined model checking and theorem proving-based FTA approach for dynamic fault trees (DFTs) has also been considered in [14]. Higher-order logic is used to prove the equivalence between DFTs and their reduced forms. The reduced DFTs are then analyzed using the Storm model checker. Although limited to SFTs at the moment, our proposed approach extends the approach of [14] as it also caters for the formal FT generation from a formally verified model.

Contribution. To the best of our knowledge, the literature on theorem proving-based FTA does not consider the automatic FT generation process from system description. This paper addresses this gap through the following key contributions:

- presenting a novel approach to combine model checking and theorem proving-based formal FTA approaches while automating the process of FT generation,
- exemplifying the approach using a tool chain consisting of the XSAP safety assessment platform and the HOL theorem prover, and
- demonstrating our methodology on an automated vehicle.

978-1-6654-6703-2/22/$31.00 ©2022 IEEE

II. Proposed Approach

The proposed approach requires 1) detailed system design documents, 2) functional requirements and 3) a list of failure modes of each system component. It mainly consists of four major steps as depicted in Fig. 1.

Step 1: Model development. We utilize the aforementioned system information, i.e., design and functional requirements, to develop a formal model using the symbolic model verifier (SMV) language. This is the input language of the nuXmv model checker [10]. We also define the failure modes of each component using the fault extension instructions (FEI) language. This is the input language of the xSAP tool [7]. The SMV language provides a flexible modeling platform to express both synchronous and asynchronous behaviors. The SMV model is organized as *modules* while offering non-deterministic variable assignments. We also consider the input from domain experts in this step as they can provide insights in the modeling process and facilitate in fault identification.

Step 2: Model validation. We validate the functionality of the SMV model by verifying various temporal properties using the nuXmv model checker [10]. The properties are described either in linear temporal logic (LTL) or computational tree logic (CTL). The property verification process also generates *counterexamples* to debug the SMV model, as shown in Fig. 1 by a feedback dashed line back to the first step.

Step 3: FT generation. This step is performed using xSAP [7], which has a rich fault library containing a comprehensive set of predefined failure modes, including different variants of stuck at, random and conditional faults. xSAP enables the FT generation by extending the SMV model with the failure mode information described in the FEI.

Step 4: FT analysis. We construct a formal FT model using the formalization of fault trees developed in the HOL4 theorem prover. At this step, we also annotate the FT with the actual probability distributions provided in the system design documents. The HOL formalization provides a library of formally verified expressions of commonly used gates. The library contains the formally verified probabilistic inclusion exclusion (PIE) principle to verify the safety properties of the given system at a desired time t.

III. Case study: automated vehicle system

We illustrate the effectiveness of our proposed approach on an automated vehicle (AV) system [6], [20]. The AV is divided into six blocks: 1) *Hardware*, 2) *Communication*, 3) *Mechanical*, 4) *Software*, 5) *Human Interaction* and 6) *Controller*, depicted in Fig. 2 and described below:

Hardware models the electric and electronic system that provides surrounding information to the vehicle. It consist of two sub-blocks: 1) *Hardware Integration* and 2) *Sensors*. The former receives the commands from the main controller and negotiates these commands with the *Sensors*. The *Sensors* block consists of reliability-wise redundant assemblies: 1) *Primary Sensors* and 2) *Backup Sensors*. The AV *Hardware* block is equipped with many sensors, e.g., light detection and ranging (LIDAR), radio detection and ranging (RADAR), global positioning system (GPS), camera, wheel encoder, infrared sensors and ultrasound. The *Communication* block interacts with the neighboring vehicles (*V2V*) and other infrastructure (*V2X*). The *Mechanical* block consists of three subsystems, i.e., *Steering* system, *Braking* system and *Conveyor* system. The *Software* block models the control algorithm and the data processing unit that is responsible for autonomous navigation [12]. The *Human Interaction* block models the human-computer interface of the vehicle. Finally, the *Controller* block is responsible for issuing commands to other subsystems of the AV. *Failure modes* of the hardware sensors are adapted from [12], [22] and, along with some failure models of other blocks, are summarized in Table I.

TABLE I: Few failure modes of automated vehicle system

Component	Failure modes
LIDAR	Laser malfunction. Mirror motor malfunction. Position encoder failure. Over voltage. Short circuit. Optical receiver damages.
Radar	Electrical component failure. Induced noise. Clutter effect.
Camera	Foreign particles. Shock wave. Over voltage. Short circuit. Vibration from rough terrain.
Communication	*V2V*. *V2X*. Database. Degraded.
Mechanical	Steering failure. Braking failure. Conveyor.
Software	Data processing unit failure. Control algorithm failure.
Human Interaction	Wrong command generation Wrong command interpretation.

IV. Experimental Results

Modeling. We develop the SMV model by making a one-to-one correspondence between SMV modules and the AV architecture of Fig. 2. The SMV model also contains a module named *Monitor*. This module is responsible for tracking the commands and behavior of the components, i.e., whether the component has complied to the command or not. According to the formal model, the system proceeds in discrete steps and at each instant, the system controller can query the health of any component from the *Monitor*. The controller and *Monitor* are assumed to be fault-free. Hence, they do not appear in the resulting FTs. We specify the failure modes of our case study in the FEI language. The resulting SMV model and all results are available online [1]

Validation. We verify the functional requirements on the SMV model. One property that we considered is that the AV can always communicate with other vehicles and the central network. This property is formalized as `:G(V2V & V2X)`. Another interesting property checks the functioning status of *Hardware*. In our SMV model, we have defined predicates to indicate the status of HV components. Once such predicate is `is_HW_functional`. *Monitor* module checks the status of all hardware sensors and *Hardware Integration*; it resets the flag to zero upon the failure of any sensor. Our SMV model satisfies all these properties.

[1] https://github.com/shahid-khan/dft-bdmp/tree/master/2022-SMACD

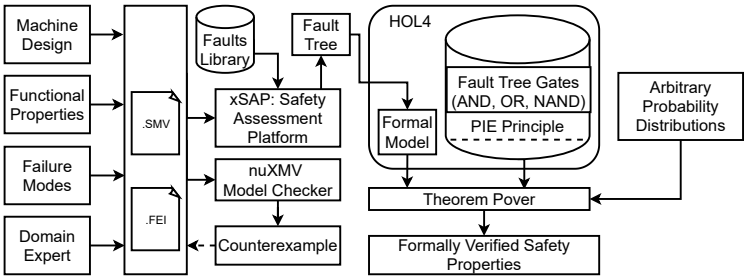

Fig. 1: Proposed approach for formal FTA

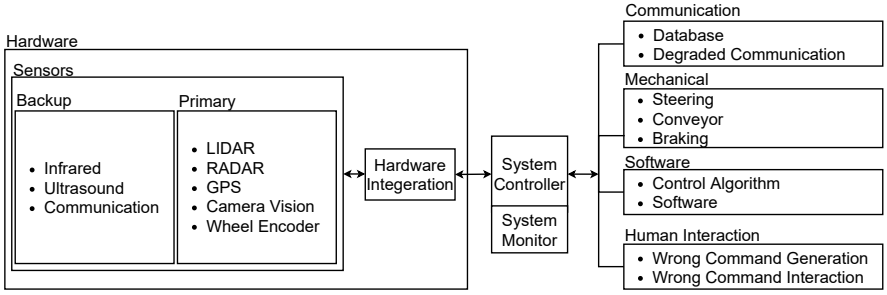

Fig. 2: Automated vehicle case study

FT generation. After validation, we use the SMV model and the failure modes information of the FEI file to generate an extended SMV model in xSAP. Once the extended model is available, we generate FTs for different TLEs. For illustration purpose, we provide a FT generated by xSAP for the TLE as shown in Fig. 3 `!VC.is_HW_functional`. (We negate the desired behavior to obtain the TLE.) The FT of Fig. 3 depicts that hardware integration failure is a single point failure mode to make the *Hardware* non-functional. Moreover, at least one primary and one backup sensor has to fail for the occurrence of the TLE. We did not include all failure modes in Fig. 2 for the sake of simplification. For instance, `E2: LIDAR` failure can occur due to six failure modes, cf. Table I.

FT analysis. We analyze the FT of Fig. 3 in HOL4 using the available FT formalization [1]. As theorem proving requires manual interaction, we provide details of each step. The FT gates are modeled as `Hol_datatype 'gate` as:

```
'gate = {AND, OR} of gate list | NOT of gate
| atomic of 'a event'
```

The formal definition of the FT in HOL4 is:

Definition 1: `⊢ AV_HW_fail_FT p E1 E2 E3 E4 E5 E6 E7 E8 E9 t = FTree p (OR [E1; AND [OR (`χ_f `p [E2;E3;E4;E5;E6] t)); OR (`χ_f `p [E7;E8;E9] t))])`

Where the function χ_f takes a probability space p, a list of random variables, a time index t and returns a failure event at time index t.

The minimal cutsets for the FT obtained from the MOCUS algorithm are:

mcs_0=['E1'], mcs_1 =['E2', 'E7'], mcs_2=['E2', 'E8']
mcs_3=['E2', 'E9'], mcs_4 =['E3', 'E7'], mcs_5=['E3', 'E8']
mcs_6=['E3', 'E9'], mcs_7 =['E4', 'E7'], mcs_8=['E4', 'E8']
mcs_9=['E4', 'E9'], mcs_10=['E5', 'E7'], mcs_11=['E5', 'E8']
mcs_12=['E5', 'E9'], mcs_13=['E6', 'E7'], mcs_14=['E6', 'E8']
mcs_15=['E6', 'E9']

Attributing the exponential distribution to the failure of components, the failure probability of AV *Hardware* FT ($F_{TLE}(t)$)is mathematically expressed as:

$$1 - e^{-\lambda_{E1} \cdot t} \cdot \prod_{i=2}^{6} \prod_{j=7}^{9} (1 - (1 - e^{-\lambda_{Ei} \cdot t})(1 - e^{-\lambda_{Ej} \cdot t})) \quad (1)$$

Equation (1) is formally verified as a theorem in HOL4, using the PIE [1] principle, while containing an exhaustive list of all the required assumptions:

Theorem 1: `⊢ (A1): 0 ≤ t ∧ prob_space p ∧ (A2): (∀`x'`. MEM` x' `(fail_event_list p [E1;E2;E3;E4;E5;E6;E7;E8;E9] t) ⇒` x' `∈ events p) ∧ (A3): mutual_indep p (FLAT (list_fail_event_list p [[E1]; [E2;E7]; [E2;E8]; [E2;E9]; ··· [E6;E7]; [E6;E8]; [E6;E9]] t)) ∧ (A4): list_exp p [C_E1;C_E2; ··· ;C_E8;C_E9] [E1;E2;··· ;E8;E9] ⇒ prob p (FTree p (AV_HW_fail p E1 E2 ··· E8 E9 t)) = 1 - list_prod (one_minus_exp_prod t [[C_E1]; [C_E2;C_E7]; [C_E2;C_E8]; [C_E2;C_E9]; ··· ; [C_E6;C_E7]; [C_E6;C_E8]; [C_E6;C_E9]])`

The first assumption (A1) ensures that the variable t models time as $t \in \mathbb{R}_{\geq 0}$. The next assumption (A2) ensures that p is a valid probability space based on the probability theory in

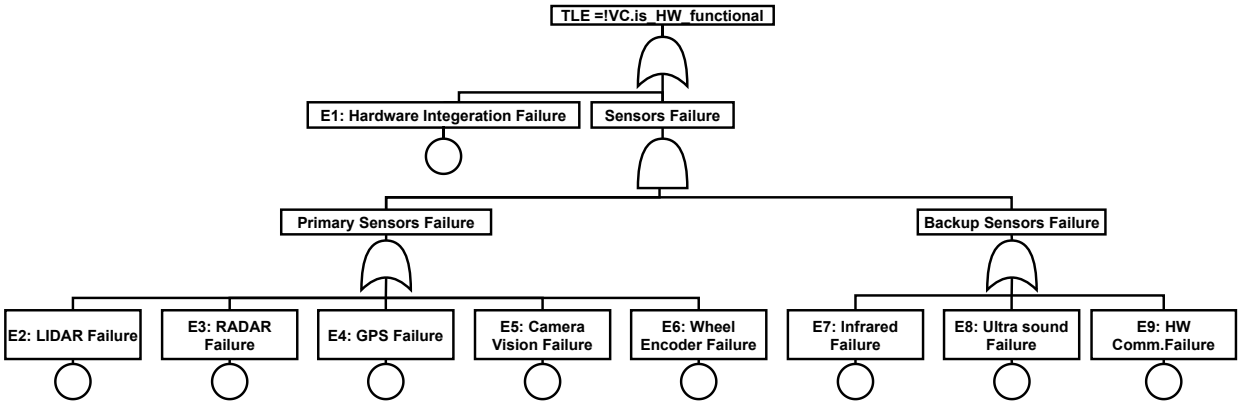

Fig. 3: Fault tree for *Hardware* block as generated using xSAP

HOL [18]. The next assumption (A3) ensures that the events corresponding to the failures modeled by the random variables $E1, \cdots E9$ are valid events from the probability space p and they are mutually independent. Finally, the last assumption (A4) characterizes the random variables E1, $\cdots$ E9 as exponential random variables with failure rates $C\_E1 \cdots, C\_E9$, respectively. The function `one_minus_exp_prod` accepts a two-dimensional list of failure rates and returns a list that corresponds to Equation (1). The proof of Theorem 1 is also provided in our on line results. The distinguishing feature of this result is that all variables are universally quantified, i.e., the failure probability of the AV *Hardware* can be calculated for any value of failure rates.

V. Conclusions

This paper presents a novel hybrid approach for formal FTA that leverages upon the complementary strengths of both model checking and theorem proving, i.e., automatic functional analysis as well as fault tree generation along with the theorem proving based complete FTA. Successful results on an automated vehicle case study are presented. We plan to extend this work to develop a comprehensive formal safety analysis framework for automated vehicle systems.

References

[1] W. Ahmad and O.Hasan. Formalization of fault trees in higher-order logic: A deep embedding approach. In *Dependable Software Engineering Theories, Tools and Applications*, volume 9984 of *LNCS*, pages 264–279. Springer, 2016.

[2] Waqar Ahmad and Osman Hasan. Towards Formal Fault Tree Analysis Using Theorem Proving. In *Conferences on Intelligent Computer Mathematics*, volume 9150 of *LNCS*, pages 39–54. Springer, 2015.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[4] Clark Barrett and Cesare Tinelli. *Satisfiability Modulo Theories*, pages 305–343. Springer, 2018.

[5] Axel Berres and Holger Schumann. Automatic Generation of Fault Trees: A survey on methods and approaches. In *ESREL 2016*, 2016.

[6] Parth Bhavsar, Plaban Das, Matthew Paugh, Kakan Dey, and Mashrur Chowdhury. Risk analysis of autonomous vehicles in mixed traffic streams. *Transportation Research Record: Journal of the Transportation Research Board*, (2625):51–61, 2017.

[7] Benjamin Bittner, Marco Bozzano, Roberto Cavada, Alessandro Cimatti, Marco Gario, Alberto Griggio, Cristian Mattarei, Andrea Micheli, and Gianni Zampedri. The xSAP Safety Analysis Platform. In *TACAS*, volume 9636 of *LNCS*, pages 533–539. Springer, 2016.

[8] Marco Bozzano, Harold Bruintjes, Alessandro Cimatti, Joost-Pieter Katoen, Thomas Noll, and Stefano Tonetta. COMPASS 3.0. In *TACAS (1)*, volume 11427 of *LNCS*, pages 379–385. Springer, 2019.

[9] Marco Bozzano, Alessandro Cimatti, A Fernandes Pires, D Jones, G Kimberly, T Petri, R Robinson, and Stefano Tonetta. Formal Design and Safety Analysis of AIR6110 Wheel Brake System. In *Computer Aided Verification*, volume 9206 of *LNCS*, pages 518–535. Springer, 2015.

[10] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv Symbolic Model Checker. In *International Conference on Computer Aided Verification*, volume 8559 of *LNCS*, pages 334–342. Springer, 2014.

[11] Mirko Conrad, Guido Sandmann, and Patrick Munier. Software tool qualification according to iso 26262. In *SAE 2011 World Congress & Exhibition*. SAE International, 2011.

[12] Plaban Das. Risk analysis of autonomous vehicle and its safety impact on mixed traffic stream. Master's thesis, Dept. of Civil and Environmental Engineering, Rowan University, 2018.

[13] Charles E. Dickerson, Rosmira Roslan, and Siyuan Ji. A formal transformation method for automated fault tree generation from a uml activity model. *IEEE Transactions on Reliability*, 67(3):1219–1236, 2018.

[14] Yassmeen Elderhalli, Osman Hasan, Waqar Ahmad, and Sofiène Tahar. Formal dynamic fault trees analysis using an integration of theorem proving and model checking. In *NASA Formal Methods*, volume 10811 of *LNCS*, pages 139–156. Springer, 2018.

[15] Yassmeen Elderhalli, Osman Hasan, and Sofiène Tahar. A Methodology for the Formal Verification of Dynamic Fault Trees Using HOL Theorem Proving. *IEEE Access*, 7:136176–136192, 2019.

[16] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The Probabilistic Model Checker Storm. *CoRR*, abs/2002.07080, 2020.

[17] Joost-Pieter Katoen. The Probabilistic Model Checking Landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 31–45. ACM, 2016.

[18] T. Mhamdi, O. Hasan, and S. Tahar. On the Formalization of the Lebesgue Integration Theory in HOL. In *Interactive Theorem Proving*, volume 6172 of *LNCS*, pages 387–402. Springer, 2011.

[19] Faida Mhenni, Nga Nguyen, and Jean-Yves Choley. Automatic fault tree generation from sysml system models. In *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 715–720, 2014.

[20] National Highway Transportation and Safety Administration NHTSA. Preparing for the Future of Transportation: Automated Vehicle 3.0, 2018.

[21] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15-16:29–62, 2015.

[22] M Swarup and M Srinivasa Rao. Safety Analysis of Adaptive Cruise Control System Using FMEA and FTA. *International Journal of Advanced Research in Computer Science and Software Engineering Research Paper*, 4(6), 2014.