Safety Conflict Analysis in Medical Cyber-Physical Systems using an SMT-Solver

Jan Kühn[1,*], Pierre Schoonbrood[1], André Stollenwerk[1], Christian Brendle[2], Nabil Wardeh[3], Marian Walter[2], Rolf Rossaint[3], Steffen Leonhardt[2], Stefan Kowalewski[1], Rüdger Kopp[3]

[1]Informatik 11 - Embedded Software, [2]Philips Chair for Medical Information Technology
both RWTH Aachen University, 52056 Aachen
[3]Clinic for Anesthesiology, RWTH Aachen University Clinic, 52056 Aachen
*corresponding author: kuehn@embedded.rwth-aachen.de

Abstract

This paper presents a method to include safety system conflicts into a fault tree analysis (FTA) with semantic extensions of fault events. The verification of the incoherent fault tree is done with an SMT-Solver. As an example a networked setup of medical devices for extracorporeal lung assist was analyzed. The method is developed as a basis for improved safety analysis of networked systems.

1 Introduction

The number of networked medical devices in clinical environments is increasing rather fast. This is motivated by trends in health care like data exchange in clinical setups for control and safety tasks and the extended use of electronic health records, which can be updated automatically. The design and engineering of medical devices is rather extensive, since the demand for safety and reliability is high. According to DIN EN 60601 the safety of a medical device has to be assured after every possible single fault. Our focus is the combination of basic techniques like the fault tree analysis (FTA) with more accurate but elaborate methods. The project ECLA-Vent focuses on the improvement of ARDS[1]-treatment by control of the therapy parameters depending on the state of the patient. The setup was used as worked example [?]. The clinical setup involves several actuators, sensors and passive components which should interact with the patient according to seven safety goals. All medical devices used in this setup, i.e. a patient monitor or a blood pump, are connected with an embedded safety network over CAN.

2 Safety Measures and Dependencies

2.1 Background

Single fault protection for an FTA is given, if every possible chain of faults, caused by a single basic event, additionally requires the malfunction of a safety system to result in a valid top event. Safety related dependencies are used to describe the interaction of safety measures. Focus of our interest are competing measures, which prevent the correct operation. Our example are the goals in a networked system for extracorporeal lung assist,

[1]acute respiratory distress syndrome

where the blood is oxygenated outside of the body. Safety systems can include the detection of a leak or supervision of a sufficient oxygenation. These can conflict due to their measures which influence the same control value limitation of the blood flow to minimize the loss of blood or ensure sufficient lung assist. A simplified example is given in Fig. 1, with the types of events and existing dependencies discussed below.

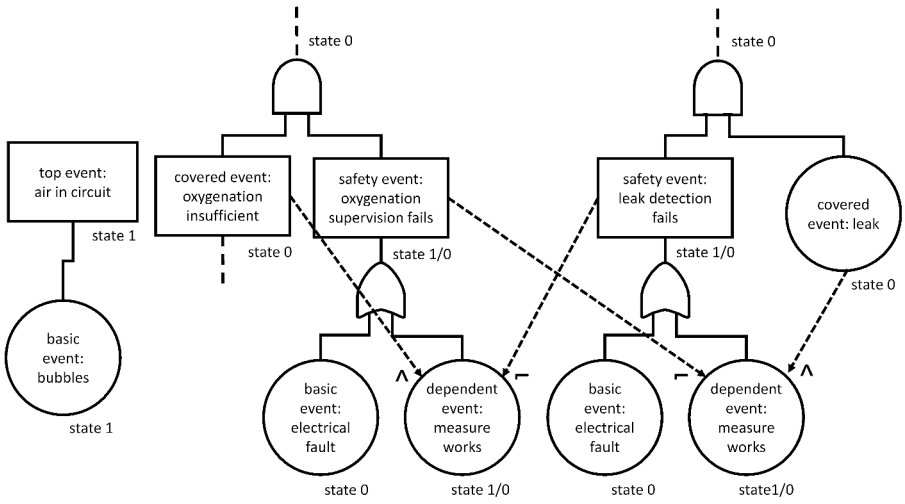

Figure 1: A reduced example of a fault tree with dependencies between safety events.

2.2 Safety Measure Dependencies

We adopt the syntax of the Faulttree Handbook [**?**]. The semantic was extended to allow the modeling of dependencies of two or more safety measures. Three types of events are used, one independent, at least one dependent and exactly one covered event for each dependent event. The formal description of a safety measure dependency is $S = (i, D, C)$ where i is the independent event, D the nonempty set of dependent events and C the nonempty set of covered events.

Independent Events

An independent event represents the fault of a safety system. Its correct operation interrupts another conflicting safety measure. An independent event as safety measure is modeled by an intermediate event. In our case it could be the failing safety measure for a leak in the cardiopulmonary bypass.

Dependent Events

The dependent event is a basic event which results in the fault of a safety measure and is caused by the successful operation of another safety measure represented by the corresponding independent event. If the independent event is false, all related dependent events are true. The dependent event can be seen as negation of the independent event which results in a non-coherent fault-tree [**?**].

Covered Event

If the fault tree is analyzed for violation of the single fault safety, a dependency generates duplicates of fault chains due to equivalent assignments. The single fault condition allows a single basic event without dependency to be true. The state of basic events which are dependent events is the negation of the related independent (safety) event. Valid assignments which create the same fault chain, unrelated to dependent events, differ in the possible states of the dependent events. The events, semantically extended to be covered events, are the events directly associated with a safety event. This coverage can be violated by a dependency of the covering safety event. Any corresponding dependent event as part of the covering safety event is only active, if the covered event is true. In the following we show which dependencies have to be considered. Dependencies which create duplicates of fault chains will be blocked. The dependency between safety measures is now assumed to be bidirectional.

- Φ set of all covered events
- Ψ set of all safety events
- Ψ_{rel} set of all relevant safety events

- $\Omega \subseteq \Psi$ set of all covering safety events
- $\Theta \subseteq \Psi$ set of safety events in conflict with a safety event $\omega \in \Omega$
- $\Phi_{\text{sel}} \subseteq \Phi$ set of all covered and fulfilled events
- $\theta \in \Theta$ a safety event which interrupts a safety event $\omega \in \Omega$
- Ψ_{elim} set of all irrelevant safety events

For every covered event $\varphi \in \Phi_{\text{sel}}$ the corresponding safety event $\psi \in \Psi$ is added to Ω. For each $\omega \in \Omega$ all interrupting dependent events are identified. For each of the dependent events independent event $\psi \in \Psi$ is added to Θ. The safety events which are further handled as irrelevant, because they interrupt with Θ, are given by $\Psi_{\text{elim}} = \Psi \setminus (\Omega \cup \Theta)$. Three cases of interactions can be distinguished for dependencies between safety measures:

1. An interaction between irrelevant safety events in Ψ_{elim} and has to be blocked. There exists no dependency allowing $\psi \in \Psi$ causing a fault for a $\psi_{\text{elim}} \in \Psi_{\text{elim}}$ and for a $\omega \in \Omega$. Otherwise ψ would be element of Θ. The dependency of an independent event $\psi \in \Psi_{elim}$ is irrelevant and blocked, because it does not contain covered events in Φ_{sel}. Otherwise Θ would contain its independent event, due to its dependency on $\omega \in \Omega$ which covers a $\varphi \in \Phi_{\text{sel}}$.

2. An interaction between safety events in the set of covered events Θ. Two variable assignments exist, which cause a fault in $\psi \in \Psi$. The dependency between the safety events $\theta_1 \in \Theta$ and $\theta_2 \in \Theta$ create these two assignments which only differ in the state of θ_1 or θ_2 (Fig. 1), which can be combined by blocking.

3. An interaction between safety events in $\Psi_{\text{rel}} = \Omega \cup \Theta$ and $\Psi_{\text{elim}} = \Psi \setminus (\Omega \cup \Theta)$. $\theta \in \Theta$ is influenced by the dependency on $\psi_{\text{elim}} \in \Psi_{\text{elim}}$, or vice versa. Only the active θ is relevant, because it is able to create a fault in $\psi \in \Omega$ and could influence the fulfillment of the top event. Because no independent event $\psi_{\text{elim}} \in \Psi_{\text{elim}}$ exists in Φ_{sel} the dependencies causing a fault in θ are blocked.

There exists a dependency for a $\theta \in \Theta$ to a $\psi \in \Omega$ for every ψ which has a dependency with θ because of the assumption of bidirectional dependencies. On the one hand $\psi \in \Omega$ can be interrupted which allows the fulfillment of the top event. On the other hand θ can be interrupted and therefore not influence any other event. If $\psi \in \Omega$ is not fulfilled, all ψ interrupting $\theta \in \Theta$ have to be fulfilled. As a result the top event cannot be fulfilled and the covering safety event becomes irrelevant.

3 Fault Chain Generation

3.1 Parsing the Fault Tree

The fault tree is parsed according to [**?**], with subsumed soundness and completeness. The AND- and OR-gates conditions are $\psi \leftrightarrow (\varphi_1 \wedge \varphi_2)$ and $\Psi \leftrightarrow (\varphi_1 \vee \varphi_2)$, respectively. The tree is parsed top down by checking the successors of every intermediate events, until an event is found, which is not an intermediate event. The result is a logical equivalence with a successor event or with an logical term for a gate and its successors.

3.2 Parsing Dependencies

All relevant information of an event, such as its dependencies, is stored in a global list. Based on this list, an additional list with dependencies is generated. The dependency is relevant, if the covered event is fulfilled. This leads to

$$K = c_1 \vee c_2 \ldots c_n,\ n = |C|,\ C \in S$$

as a formal condition for activation of a dependency S, with the covered event $c_i \in C$. The value v of a dependent event is given by $v = \neg i \wedge K$, with independent event i and activation condition K. $d \leftrightarrow v$ is generated for every dependent event $d \in D$ of a safety measure dependency S.

3.3 Single Fault Safety

For Φ as a set of basic events in a tree we define $\Phi_b \subseteq \Phi$ as the subset of all basic events according to the semantics of [**?**] and $\Phi_d \subset \Phi$ the set of all dependent basic events caused by a interrupting safety measure with $\Phi_b \cup \Phi_d = \emptyset$. For every pair of $b_1, \ldots b_n \in \Phi_b$, with $n = |\Phi_b|$ is $\neg(b_i \wedge b_j)$ created, where $i, j \in \{1, \ldots, n\}$ and $i \neq j$. This is only satisfiable if no pair exists where both are valid the same time.

4 Evaluation

An existing open source tool with XML support was used [**?**]. The modeled fault trees were analyzed for dependencies. As result minimal fault trees were identified. These trees represent the fault chain from a top event, that means the violation of a safety goal, to the responsible basic event. The minimal trees include only these events, that are true for the occurring basic event. Since a fault tree modeled with dependencies is incoherent, the analysis is more complex. This reason and the option to use arithmetic information motivated the use of an SMT[2]-Solver. The fault tree was processed according to the parsing methods above to generate input in form of Boolean expressions which were checked for satisfiability with the SMT-solver. Because it is supported by several well known solvers the parsed input is in SMT-Lib format. Here SMTInterpol was used [**?**]. The solver finds a valid variable assignment which fulfills a top event according to the single fault condition. This is done iteratively with exclusion of the models found by including their negation until no assignment can be found by the solver. A description in form of the relevant tree is given by comparing the set of logical equivalents (cf. Sect. 3) with an identified assignment. Further a report about the events which violate the single fault condition and their fault chain is generated, which includes information describing the events. Minimal fault trees which were identified are reported in this form: "safety goal 4: loss of blood in extracorporeal circuit" $\leftarrow$ ... "leak detection fails" $\leftarrow$ "leak" $\wedge(\neg$ "oxygenation supervision fails") (Fig. 1). In case of the worked example 7 safety goals were defined. 13 dependencies between safety measures were identified in the existing FTA and therefore 42 dependent events were added to include the conflicts. The valid assignments which violate the single fault condition were increased from 21 to 52. It has to be noted, that the safety dependencies are over-approximated, since detailed behavior is not modeled yet.

5 Related Work

The problem of undesired equivalent results as described in Sect. 2.2 could be handled by quantifier elimination [**?**], which is a more general approach, but in case of this problem less efficient. The SAT-Solver has to be used at least once for each time the elimination algorithm is used for an equivalence class. Other examples of safety analysis methods with SMT-Solvers can be found in the work of Bozzano et al. [**?**], but do not handle conflicts in FTAs and therefore differ in usage of the solver.

6 Conclusion and Outlook

A method to increase the accuracy of an FTA by modeling dependencies of existing safety measures was presented. It can be used to prove the single fault safety of a system based on the FTA. The fault tree syntax was not changed and the semantic only extended. Detailed reports about violations of the single fault safety were generated with the use of an SMT-Solver. They include the violating fault chain, from responsible basic event to the violated top event. The computation time needed for the example mentioned above on a consumer notebook (i.e. Intel i5-3320M, 4GB DDR2) was always below 20 seconds. The presented method allows to include possible conflicts of networked cyber-medical systems and verification of these fault trees with increased complexity.
Further our method is thought as basis to connect high- and low-level methods to support safety and reliability during the design of networked systems with significant safety demands. Currently the computational efficiency of this method is tested more detailed and estimated for increasing tree size. To improve the FTA the types of supported events can be extended according to [**?**] to allow the modeling of priorities, which decreases the number of false positives. Over-approximation can be further reduced by including arithmetic information for the SMT-Solver, provided by detailed models.

Acknowledgements
This work was supported by the German Research Foundation DFG (DFG - Grant PAK 138/2). The authors gratefully acknowledge this allowance.

[2]Satisfiability modulo theories

References

[BCM13] M. Bozzano, A. Cimatti, and C. Mattarei. Automated analysis of reliability architectures. In *Engineering of Complex Computer Systems (ICECCS), 2013 18th International Conference on*, pages 198–207. IEEE, 2013.

[BHSØ03] M. Burgess, E. Haugvaldstad, D. Steinnes, and H. Øystein. Faultcat, http://www.iu.hio.no/faultcat/, Version: August 2003.

[BKK11] J. Brauer, A. King, and J. Kriener. Existential quantification as incremental sat. In *Computer Aided Verification*, pages 191–207. Springer, 2011.

[CHN12] J. Christ, J. Hoenicke, and A. Nutz. Smtinterpol: An interpolating smt solver. In *Model Checking Software*, pages 248–254. Springer, 2012.

[HRVG81] D. F. Haasl, N. H. Roberts, W. E. Vesely, and F. F. Goldberg. Fault tree handbook. Technical report, Nuclear Regulatory Commission, Washington, DC (USA). Office of Nuclear Regulatory Research, 1981.

[RPA08] R. Remenyte-Prescott and J. Andrews. Analysis of non-coherent fault trees using ternary decision diagrams. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 222(2):127–138, 2008.

[SKB+14] A. Stollenwerk, J. Kühn, C. Brendle, M. Walter, J. Arens, M. N. Wardeh, S. Kowalewski, and R. Kopp. Model-based supervision of a blood pump. In *19th World Congress of the International Federation of Automatic Control*, pages 6593–6598, 2014.

[STR02] G. Schellhorn, A. Thums, and W. Reif. Formal fault tree semantics. In *Proceedings of The Sixth World Conference on Integrated Design & Process Technology, Pasadena, CA*, 2002.