

INCLUDING SYSTEMATIC FAULTS INTO FAULT TREE ANALYSIS

Israel BARRAGAN SANTIAGO^{* (1)}, Jean-Marc FAURE^{(1),(2)} and Yiannis PAPADOPOULOS⁽³⁾

⁽¹⁾ LURPA – ENS Cachan – 61, Avenue du President Wilson, 94230 Cachan, France
{barragan, faure}@lurpa.ens-cachan.fr

⁽²⁾ Institut Supérieur de Mécanique de Paris (SUPMECA) – 3 rue Fernand Hainaut, 93407 Saint-Ouen, France

⁽³⁾ Department of Computer Science, University of Hull – Hull HU6 7RX, UK

Abstract: Fault Tree Analysis (FTA) is a technique widely used for fault forecasting of physical systems. Although FTA is considered a well established safety analysis technique, paradoxically classical Fault Trees include only random faults. However, in modern automated systems, undesirable events arise not only from random hardware faults but also from defects in the logic of software controllers that control the physical system. Faults generated by these software controllers are systematic faults caused by coding errors or misinterpretations of control requirements. This paper proposes an extension to the basic Fault Trees construction process which takes into account this category of faults and advocates the use of dynamic and temporal gates to model it.

Keywords: Controller dependability, Event ordering, Fault tree analysis, Safety analysis, Temporal fault tree.

1. INTRODUCTION

Since its development in 1960 by Bell Labs, a large volume of technical and scientific work has been reported in the literature about FTA. Today, it is a well-known fault forecasting technique which is widely used in the design of safety critical systems. Fault Trees are commonly used to represent the effect that random hardware faults of components have on a system. One difficulty with applying this technique on modern automated systems is that such systems are the combination of logic controllers and controlled process (plants) where controllers receive and process inputs coming from the process and generate outputs to the process (see figure 1). Clearly, therefore, safety analysis of such systems must take into account not only the physical faults of components, including those of controllers, but also any faults caused by errors in the logic of those controllers.

In this paper, we focus on logic controller faults and we develop a method for their representation in FTA. Logic controller faults can be categorized in three classes depending on whether they are caused by:

- Hardware failures of the controller
- Unhanded deviations of controller inputs caused by failures of sensors connected to the controller
- *Design flaws* in the logic (software) of the controller, either a result of coding errors or misinterpretation of control requirements.

The first two classes of fault are currently considered in a classical FTA. Indeed in the course of such analysis, an erroneous output of a controller is typically attributed to *primary* and *secondary* hardware failures of the controller itself or to *command* failures typically deviations of controller inputs which are in turn caused by failures of connected sensors. The work developed here proposes to extend the FT method to integrate the analysis of the third class of faults in the above categorisation, i.e. those caused by *design flaws*.

* The Mexican Council of Technology CONACYT finances Israel Barragan.

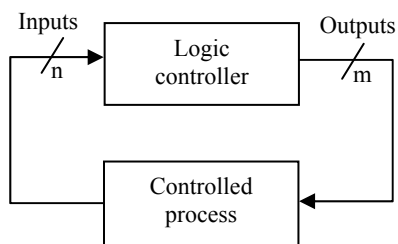


Fig. 1. Synthetic view of an automated system

Such faults fall in the general category of *systematic* faults because they can be reproduced every time the conditions that trigger the error in the control logic are present. These conditions are typically sets of correct inputs which by triggering the embedded error result to a fault manifested as an omission or commission of controller outputs or deviations of outputs from correct timing or value. Identifying these kinds of faults, therefore, requires from analysts to assume that even with correct input information the controller fails, delivering no output or erroneous outputs.

The integration, into the fault tree structure, of controller faults that can potentially be attributed to design flaws can assist the targeted investigation and eventual elimination of such flaws in relevant parts of the control logic. We should point out that although, in general, there may be large numbers of errors in a program, it is only a small portion of those errors that will trigger faults that can contribute to the hazard investigated as a top event in a particular fault tree. It is precisely those critical systematic faults that the proposed extension to FTA aims to identify.

The purpose of identifying faults caused by design errors is to remove those errors. FTA is a simple and widely applied method, familiar to most safety engineers. Extending its application on software controlled systems, therefore, will be beneficial in terms of improved fault forecasting and fault removal in such systems.

The inclusion of controller faults in fault trees requires an extended FTA vocabulary, in which the notions of time and event ordering exist and can be used to describe relationships among input conditions that trigger the fault and output conditions with which the fault is manifested. Managing that goal is an important point discussed here.

This paper is structured as follows. Section 2 recalls fundamentals and standard construction rules of fault trees. Section 3 deals with the extension of FTA to include systematic faults. The use of dynamic and temporal gates to represent temporal relationships between events is developed in section 4. The method is illustrated with a simple example in section 5. Conclusions and prospects are discussed in the last section.

2. FAULT TREE DESIGN

2.1 Fault Tree fundamentals

Fault Tree Analysis aims at identifying all sufficient and necessary combinations of basic events in a system that cause the top event of the fault tree which represents a hazardous system failure. These combinations of basic events are called Minimal Cut Sets. A basic event, typically a component fault, is a leaf node in the tree, i.e. an event that is not developed further in the analysis. The connections between the various identified basic events are carried out by means of logical gates. The two most commonly used gates are the AND-gate and the OR-gate.

Besides gates, several symbols are used to represent the fault events. Rectangles are used to describe intermediate events that result from the conjunction or disjunction of several basic events. Circles describe basic events that require no further development. Diamonds represent undeveloped events, which are conditions not further examined either because they are considered highly unlikely, and thus of no interest, or because information is unavailable. See figure 2.

Several commercial software tools support manual fault tree construction and automate qualitative analysis (i.e. calculation of minimal cut sets) as well as quantitative estimation of system unavailability from probabilities of basic events. Methods for automatic construction of fault trees are described in (Papadopoulos et al., 2001) and (Laengst et al., 2003).

2.2 Fault Tree construction

To provide a systematic way in which the construction of the fault tree may be approached is proposed in the fault tree handbook (US N.R. Commission, 1981). It is generally followed by analysts and has been included into other texts that provide guidance on construction such as (Andrews, 2002). The approach requires events in the fault tree to be classified as state-of-component faults or state-of-system faults. A state-of-component fault is one that can be caused by a single component failure. If a single component failure cannot cause the fault then it is classified as a state-of-system which behaves as an intermediate event.

State-of-component faults, are then developed using the fault tree structure illustrated in figure 2. A primary fault represents the failure of a component due to its internal defects. It occurs in an environment for which the component is qualified. A secondary fault is a fault of a component caused by excessive environmental or operational stress. In other words, a secondary fault represents a situation in which the component fails in conditions that exceed the conditions for which it was designed.

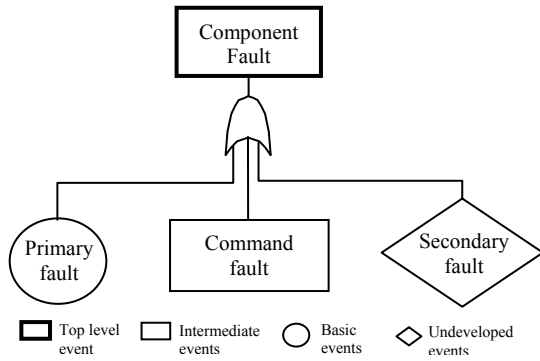


Fig. 2. Classical Fault Tree construction

Finally, a command fault describes a situation in which the component has not physically failed but operates in the wrong time or context. In such conditions, the component typically produces no output or incorrect output in response to inappropriate or misleading inputs received either from sensors or controllers that control its operation.

Primary faults represent basic events of the fault tree. On the other hand, secondary faults can be further investigated in which case the causes of any excessive environmental conditions are identified. For example if the condition is unacceptably high temperature, a failure of a cooling subsystem may be identified as a cause. Finally, command faults represent intermediate events in the fault tree which are always investigated in order to establish how incorrect inputs or commands are generated by other components further upstream in the system. The fault tree structure is progressively created as secondary and command faults are further investigated.

3. EXTENDING FTA TO ADDRESS SYSTEMATIC FAULTS

The scheme of figure 2 investigates physical faults and command faults caused by deviant inputs but omits any assessment of *design faults*, e.g. errors in control logic. Because such faults are common in software controlled systems, their identification and removal is extremely important and therefore, in our view, the scheme of figure 2 must be extended to account for these faults. Programmable logic controllers are typically programmed in one or more languages standardized by IEC 61131-3 (IEC, 1993). A representation of a sample program expressed in Ladder Logic is sketched in figure 3. The program decides the state of controller output O_1 by evaluating a logical combination of inputs I_1 and I_2 and previous output states.

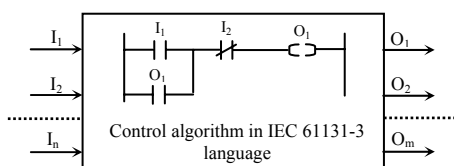


Fig. 3. Example of simple logic controller

In this system, a controller fault, i.e. an omitted or erroneous output, can be caused by a number of different failures. A hardware failure of the controller and its elements (e.g. input/output cards, processor, etc) is one possibility. A deviation on an input is a second possibility. If, for example, in the current state of the program of Fig.3, O_1 is set (i.e. $O_1=1$) and must be reset via I_2 , but the sensor which monitors I_2 is stuck at 0 then the result is an omission of reset output O_1 . This is a classic case of a command fault whereby a wrong input (WI) causes wrong output (WO). This can be symbolically represented as $(WI \rightarrow WO)$ and describes a situation covered by the scheme of Fig.2.

There is a third possibility to get an erroneous output in the example of Fig. 3 (and, indeed, in any control program). This is the situation where the controller produces an erroneous output in response to a set of correct inputs that trigger an error in the control logic. For instance, because a set dominant memory has been programmed instead of a reset dominant memory. This situation can be symbolically represented as $(RI \rightarrow WO)$, i.e. right inputs lead to wrong output) and is not covered by the scheme of Fig.2.

To address this deficiency of classical FTA, and enable assessment of systematic faults, we propose an extension which maintains the guidelines summarised in section 2.2 and the scheme of Fig.2, but in the case of command faults introduces a variant of the traditional technique. Our approach is to differentiate between classical command faults caused by undetected deviant inputs ($WI \rightarrow WO$) and command faults coming from control algorithms executed in the controller itself (systematic faults, or $RI \rightarrow WO$). Fig.4 depicts the proposed structure of the new general fault tree construction template.

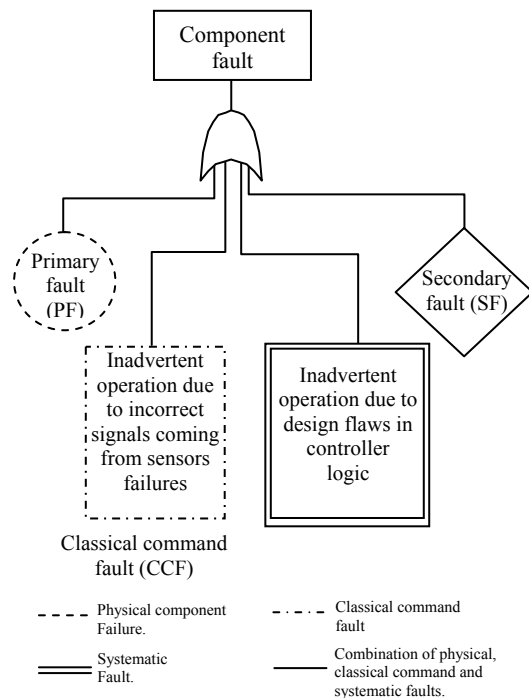


Fig. 4. General template

Overall, the result of applying the extended template in the course of fault tree construction, is fault trees which include five kinds of faults: physical faults (PF), classical command faults (CCF), systematic faults (SyF), secondary faults (SeF) and intermediate events that represent combinations of them (edged respectively by a dot line, a dot-dashed line, a double line, and a simple line for the last two kinds, see figure 4).

4. DESCRIBING SYSTEMATIC FAULTS USING A VOCABULARY OF GATES

Classical FTA assumes that the order in which the basic failures occur is irrelevant. However, especially in programmable systems, situations frequently arise in which the order of events is vital for the correct or faulty behavior of systems (Bozzano and Villaflorita, 2003). In general, the representation of controller faults in fault trees requires mechanisms for specifications of temporal relationships among events. This temporal information is essential for the description both of the causes and the effects of such faults. A fault tree is called dynamic (Cepin and Mavko, 2002), if it enables the description of such dynamic information about events and their temporal relationships. This requires the use of gates which have come to be known as “dynamic”. Some of them modeling functional dependencies and primary-spare behavior have been proposed by (Dugan, 1999).

In this work, we adopt the use of dynamic gates as a means of describing systematic faults but we focus on two dynamic gates original defined in the fault tree handbook: “Exclusive OR with condition” and “Priority AND”. The specification of these gates is given in Fig. 5.

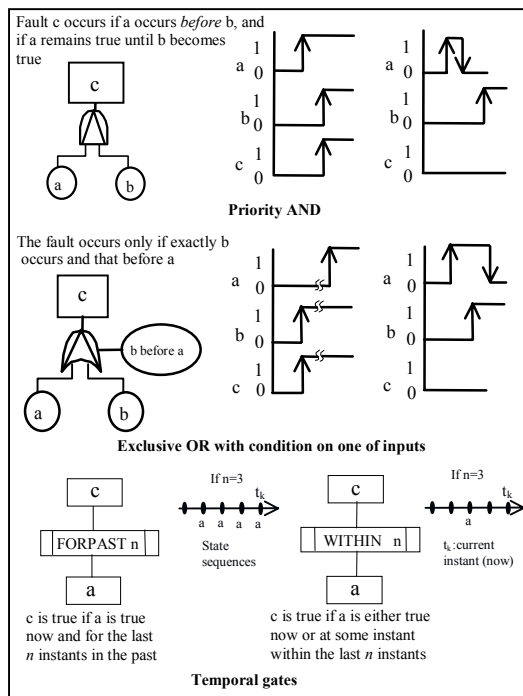


Fig. 5. Dynamic and temporal gates

In the case of the PAND gate, the chronograms show that the output fault happens only if *a* and *b* occur, with *a* occurring before *b*. Here we include the important notion of “event persistence”, i.e. *a* must still be persistent when *b* appears. That means if *a* disappears before the appearance of *b* no fault will be detected in the output of the gate.

Furthermore, the work developed by (Palshikar, 2003) proposes the addition to the fault tree notation of other special gates to describe temporal systems. The term Temporal Fault Trees (TFT) is coined for this kind of fault trees. TFT allow the user to easily specify physical time conditions between events. There are several temporal gates but in this paper, we deal only with WHITHIN *n* and FORPAST *n* gates. The representation of these two gates is also included in figure 5.

The extended vocabulary of gates illustrated in Fig.5 can be used for the representation of potential systematic faults in fault trees. Such faults are described using the gates of Fig.5 as logical and temporal relationships between correct conditions on controller inputs and failure conditions on controller outputs which only become true in the presence of the systematic fault in question. Conveniently, such relationships can then be derived from the tree and further model-checked to confirm or not the presence of the implied systematic faults that potentially contribute to the hazardous top event.

5. EXAMPLE

The proposed process is illustrated by an example derived from analysis of a pick-and-place manipulator (sketched in figure 6), which is part of an assembly line located at the Mechanical Engineering Department of the ENS Cachan. Two cases are analyzed: the first shows application of dynamic gates while the second focuses on temporal gates.

The goal of this manipulator is to pick up gearwheels with suction cups and to transfer the gearwheels to gear housings using two single-acting pneumatic cylinders. A logic controller commands the global process. Inputs and outputs of the control program are also given in figure 6. Only automatic operations are being considered.

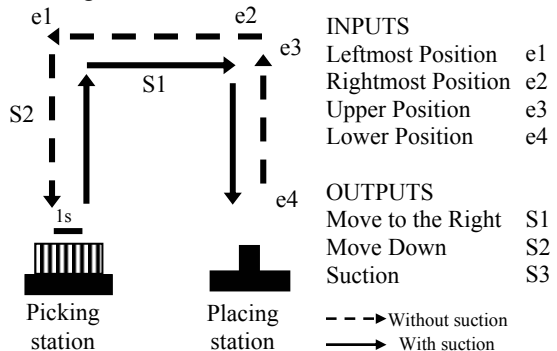


Fig. 6. Pick-and-place manipulator

5.1 First analysis: The part falls down during the movement from the picking to the placing station

The undesired event to analyse is “the part falls down during movement”. The first stage is shown in Fig.7. Four causes linked by an OR gate produce the fault. The first, “Suction device is broken” is a primary fault and, therefore, a basic event. The second, “Collision of the part with the environment”, is a secondary fault which for simplicity is not further developed here. The third, “Inadvertent commission of stop suction command”, represents a classical command fault caused by failures further upstream in the system. In this case the fault is produced if both position sensors (monitoring e2 and e4) are faulty, erroneously reporting to the controller that the arm is at the placing station. In such conditions, misled by sensors failures, the controller inadvertently stops the suction. These sensors failures are primary faults and therefore become basic events which are connected to the command fault by an AND gate.

Finally, the last cause of the top event represents the case of a systematic fault of the controller itself. In this case, although sensors are working correctly, the controller stops the suction before the manipulator reaches the correct place. This fault contradicts the specification which states that “suction must never be stopped before the manipulator is at the placing station”. The term “before” in the latter statement suggests a certain temporal order of events, and thus this portion of the tree is developed as a dynamic one (see Fig.8). The event ordering gate “Exclusive OR with condition” is used to model this fault.

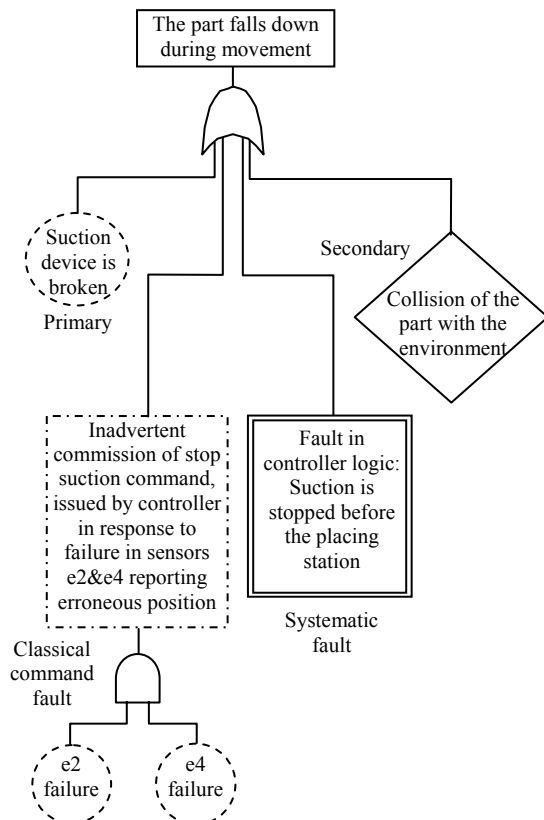


Fig. 7. Fault tree for “the part falls down”

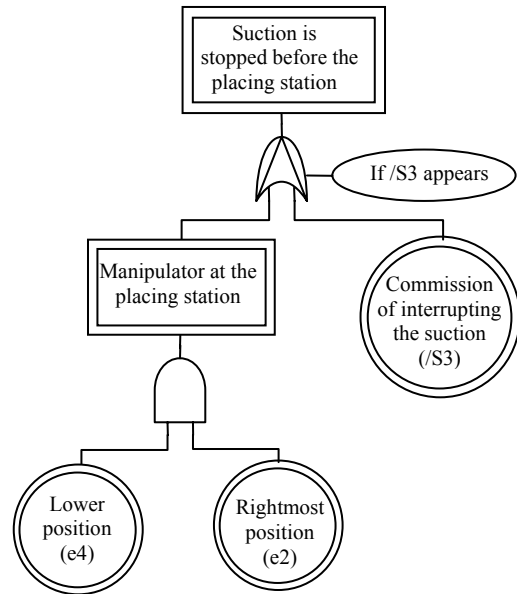


Fig. 8. Development of intermediate event “suction is stopped before the placing station”

The behavior of this gate (see Fig.5) defines that the output fault occurs if the conditioned input appears before the other input (that can be produced later or not produced at all). The left input of the gate shows that the manipulator is in position at placing station (lower position and rightmost position, indicated by the correspondent sensors). The other gate input indicates that the controller sends the command to interrupt the suction. This is the conditioned input. The fault is produced if this input is true before the other one. Notice that we have used the signals as inputs events of the gate. They are normal events and not really faults. The fault comes from the erroneous order of these signal changes.

5.2 Second analysis: Fault in vertical movement

A further undesired event in this system is a fault in the vertical movement of the manipulator. The specification imposes that the vertical cylinder must remain one second at the picking station before leaving to the other station so as to catch the gearwheel. Here, the vertical cylinder stays less than the required time of 1 second at the picking station. The fault tree is shown in Fig.9 and shows that a physical failure, e.g. a leak into the cylinder, can produce its retraction and cause the undesired event. That is considered as a primary fault. Secondary faults caused by the environment are also possible but for simplicity are not further developed here.

Instead we focus on the error in the operation of the directional valve that commutes and produces the retraction of the vertical cylinder. This fault must be seen as a systematic one because it can only be caused by an erroneous commission of the “commute command” issued by the control logic that operates the valve. In this case the “stop vertical movement command” is issued early and the cylinder retracts before the specified time.

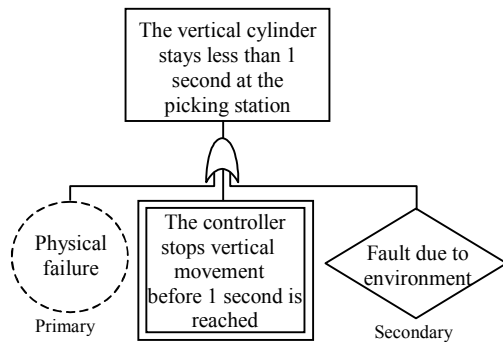


Fig. 9. Fault tree for the vertical cylinder

Once more this systematic fault can be modeled using the conditioned Exclusive OR (Fig.10). The left input of the gate defines that the manipulator is at the picking place (lower position and leftmost position) now and for the last second. To represent this timed condition the temporal gate FORPAST_n is used. The right part of the tree shows that the controller issues the “stops vertical movement” command early, i.e. before the condition described by the left branch is reached.

6. CONCLUSIONS

The work presented in this paper proposes an extension to classical fault trees and their construction process which enables inclusion and analysis of systematic faults. The approach contributes to improve safety analysis of systems by integrating into the examination of causes of failure the potential controller faults caused by design flaws in control algorithms. Such faults are systematically identified and recorded in the fault tree structure every time a controller is encountered in the course of a systematic traversal of the system model from system outputs to system inputs, in the course of which the causes of failure are progressively further investigated.

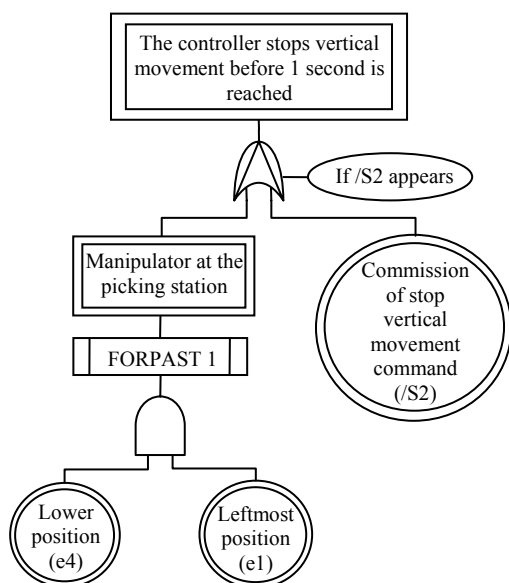


Fig. 10. Further development of the systematic fault

Once the potential systematic faults have been identified, the conditions that confirm their presence are then described using an extended fault tree vocabulary that contains classical, dynamic and temporal gates. In (Barragan and Faure, 2005), it has been shown that such fault trees can act as a useful tool for obtaining formal properties for timed model-checking. We are currently looking into the problem of qualitative analysis and reduction of fault trees which include dynamic and temporal gates. This analysis will enable the identification of minimal sequences of systematic faults. Moreover, it could be possible to semi-automatically generate such fault trees in the context of HiP-HOPS, a recently proposed technique for model based synthesis of fault trees (Papadopoulos and Maruhn, 2001). Once the fault tree is constructed, it would be possible to automatically check the presence or not of systematic failures in the control code via model-checking of the conditions specified in the fault tree.

REFERENCES

- Andrews, J. (2002). Fault Tree Analysis – Common Misconceptions. *Proceedings of the 20th International System Safety Conference*, pp. 401-410, August 5-9, Denver, Colorado, USA.
- Barragan, I. and J.M. Faure (2005). From Fault Tree Analysis to Model Checking of controllers. *Proceedings of the 16th IFAC WC 2005*, 6 pages, July 4-8, Prague, Czech Republic.
- Bozzano, M. and A. Villafiorita (2003). Integrating Fault Tree Analysis with Event Ordering Information. In: *Proceedings of ESREL 2003*, pp. 247-254, June 15-18, Maastricht, The Netherlands.
- Cepin, M. and B. Mavko (2002). A dynamic fault tree. *Reliability Engineering and System Safety*, N° 75, pp. 83-91.
- Dugan, J.B. and K.J. Sullivan (1999). Developing a low-cost, high-quality software tool for dynamic fault tree analysis. *Transactions on Reliability*, pp. 49-59.
- International Electrotechnical Committee (1993). IEC 61131-3, Programmable controllers, Programming languages.
- Laengst, W., A. Lapp, K. Stuebbe, J. Schirmer, D. Kraft and U. Kiencke (2003). Automated risk estimation based on fault trees and fuzzy probabilities. In: *Proceedings of SAFEPROCESS 2003*, pp. 51-56, June 9-11, Washington, D.C., USA.
- Palshikar, G.K. (2003). Temporal Fault Trees. *Information and Software Technology*, n° 44, pp. 137-150.
- Papadopoulos, Y. and M. Maruhn (2001). Model-based automated synthesis of fault trees from Matlab-Simulink models. *DSN'01, Int'l Conf. on Dependable Systems and Networks*. pp. 77-82. Göteborg.
- US Nuclear Regulatory Commission (1981). Fault Tree Handbook. *Technical Report NUREG-0492*, Washington, DC.