OBTAINING TEMPORAL AND TIMED PROPERTIES OF LOGIC CONTROLLERS FROM FAULT TREE ANALYSIS

Israel Santiago Barragan, Matthias Roth, Jean-Marc Faure

OBTAINING TEMPORAL AND TIMED PROPERTIES OF LOGIC CONTROLLERS FROM FAULT TREE ANALYSIS

Israel BARRAGAN SANTIAGO [*] [(1)] **, Matthias ROTH** [(2)] **and Jean-Marc FAURE** [(1), (3)]

[(1)] *LURPA – ENS Cachan – 61, Avenue du Président Wilson, 94230 Cachan, France*
{barragan, faure}@lurpa.ens-cachan.fr
[(2)] *Technische Universität Kaiserslautern, Erwin-Schrödinger-Str.12, 67653 Kaiserslautern, Germany*
[(3)] *Institut Supérieur de Mécanique de Paris (SUPMECA) – 3 rue Fernand Hainaut, 93407 Saint-Ouen, France*

Abstract: One of the prerequisites for formal verification of logic controllers using model-checking is the formalization of properties to verify. The work presented in this paper proposes a method to elaborate the formal properties of a logic controller from a Fault Tree Analysis (FTA). The method developed here extends the traditional FTA with event ordering and timed information by introducing specific gates which model logic and physical time constraints. The behavior of these gates is then formalized in the form of state automata; formal properties are derived from the set of automata obtained at the end of FTA. A simple case study exemplifies the method.

Keywords: Dependability, Fault tree analysis, Formal verification, Event ordering, Timed automata.

1. INTRODUCTION

Model-checking is a very popular formal verification technique relying on state automata theory (Bérard, et al., 2001). Its principle is to check whether formal properties hold (or do not hold) on a state model of the system. Hence, formal verification of logic controller using model-checking implies to build a state model of the controller as well as to write the formal properties that express in a formal way the application requirements that must be satisfied.

Unfortunately properties formalization is a difficult task because the application requirements are expressed in industry in a quite informal way, i.e. some sentences in natural language or drawings, but never with sound mathematical statements. Moreover the formalisms used to state formally properties in model-checking environments (timed or untimed temporal logics or timed automata) are totally unknown by automation engineers.

The work presented in this paper proposes a methodology to facilitate the identification and formalization of properties by means of an analysis technique commonly employed in industry for critical systems design: fault-tree analysis (FTA).

Our aim is to take benefit of the results of a fault tree analysis, developed from the application requirements, to elaborate formal properties that will be later inputs of a model-checker. Therefore, this work enables to bridge the gap between fault tree analysis, a widespread fault forecasting method, and model-checking, a promising fault removal method.

Since we deal with formal verification of controllers, the preliminary fault-tree analysis must take into account not only the random faults produced by failures of physical components of the process but also the faults issued from controllers. These latter faults come from designers' errors or misinterpretation of the control requirements and behave as systematic faults because they can be repeated all the time while the program is running.

The inclusion of controller faults in fault trees requires an extended FTA vocabulary, in which the notions of events ordering and physical time exist and can be used to describe relationships among input conditions that trigger the fault and output conditions with which the fault is manifested. This vocabulary is composed of temporal (untimed) and timed gates[1] whose behaviors are formalized using state automata.

This paper is structured as follows. The proposed method is outlined in section 2. Section 3 presents the extended FTA vocabulary, while section 4 deals with the formalization of the gates of this vocabulary. In section 5, a case study is developed so as to exemplify the overall method. Conclusions and prospects are discussed in the last section.

2. METHOD OVERVIEW

Figure 1 depicts the faults forecasting and removal approach that we advocate. Our contribution stands on the right side of this figure and can be split in two steps: design of a fault tree that includes systematic controller faults, and elaboration of the formal properties of the controller from the result of this FTA. This set of properties will be then input into a model-checking tool, once the controller will be really designed and implemented. FTA and model-checking are therefore performed at different phases of the design.

Obtaining formal properties from a fault tree requires to have at one's disposal a formal definition of each elementary gate as well as formal composition rules that permit to translate any set of connected gates into a formal model, provided that this set is consistent. Hence, formalization of each fault tree gate and gates set is one of the main objectives of this research.

Fault tree formalization has been addressed by several works (Schäfer, 2003), (Ortmeier, 2004) and (Reif, 2004). Nevertheless, it matters to highlight that the objective of these works is far different from that of ours. These researches indeed are aiming at proposing a modeling frame for safety analysis that combines FTA and model-checking; model-checking is then used to check completeness and correctness of a fault-tree from a formal model of the process. On the contrary, our approach uses sequentially these two techniques and focuses on the faults of the controller. Moreover, it will be shown in the next section that FTA taking into account controllers faults requires to introduce specific gates that are not taken into account in the works mentioned above.

---

[1] Even if the term 'gate' is commonly used to describe combinatory logic operators, such as AND, OR, …, it will be used in what follows to name each basic operator of a FT whatever the behavior of this operator (purely combinatory behavior or behavior depending on events order or on physical time).
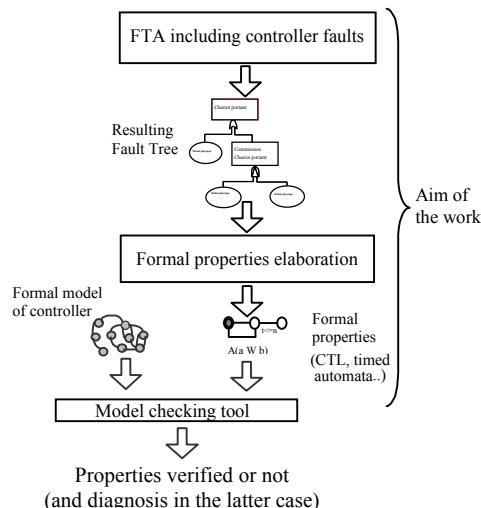

Fig. 1. From FTA to model-checking of controller

3. MODELLING CONTROLLER FAULTS USING TEMPORAL AND TIMED GATES

Fault Tree Analysis aims at finding all the associated sets of basic events that could cause that a given top event (a system failure of some kind) occurs. In classical FTA, the connections between the various identified basic events are carried out by means of logical gates, such as the AND-gate and the OR-gate, that express static relationships among their inputs.

Then 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 (Dugan, 1999), (Bozzano and Villafiorita, 2003). In general, the representation of controller faults in fault trees requires mechanisms for specifications of temporal relationships among events. This requires the use of gates which have come to be known as "temporal" for they express faults that depend on events order. In this work, we adopt the use of temporal gates as a means of describing systematic faults but we focus on two gates originally defined in an intuitive fashion in the fault tree handbook: "Priority AND" and "Exclusive OR with condition" (US N.R. Commission, 1981). The specification of these gates is given in Fig. 2.

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 at 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 timed systems, i.e. systems whose faulty behaviors are described using physical time. There are several timed 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 2.
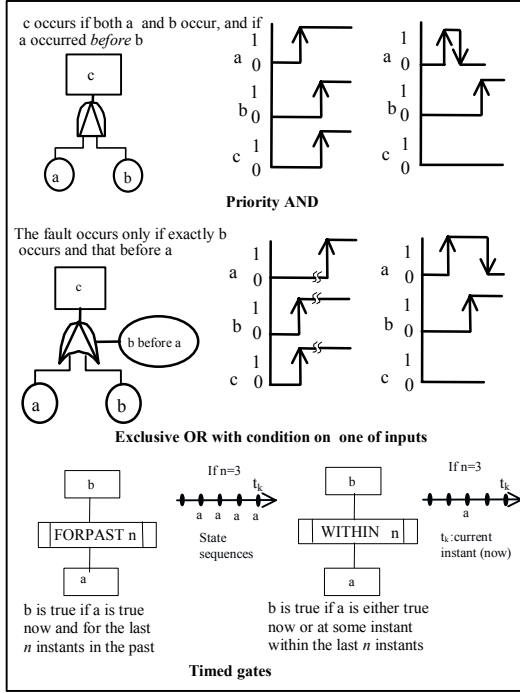

Fig. 2. Temporal and timed gates

Finally, the fault tree including systematic controller faults is constructed from static gates, temporal gates (if event-ordering description is necessary to describe a fault), and timed gates if timing constraints must be taken into account. The use of these gates preserves the simple, qualitative and visual nature of the fault trees. Nevertheless each temporal and timed gate must be endowed with a sound formal definition as shown in the next section.

4. FORMALIZATION OF TEMPORAL AND TIMED GATES

The aim of this section is to show that temporal and timed gates can be represented formally thanks to timed automata. Temporal gates can be also formalized using temporal logic statements as presented in (Barragan and Faure, 2005). Nevertheless using an only formalism to describe the behavior of these two types of gates is compulsory when addressing gates combination.

4.1 Timed automata

Timed automata is a modeling language that allows to describe the behavior of systems by means of finite state automata, extended with clocks and time constraints (Alur and Dill, 1994).

To give the basic definitions of timed automata the following notations are used (Behrmann, 2004) : C is a set of clocks. In a set of integer valued variables. $D(C,In)$ and $B(C)$ are conjunctions over simple conditions of the form $x \square c$ or $x - y \square c$, where $x,y \in C$ for $B(C)$ and $x,y \in C$ or In for $D(C,In)$. $c \in \aleph$ and $\square$

$\in \{<, \leq, =, \geq, >\}$. $F(C,In)$ represents the assignment of values to clocks C and variables In which happens on the transition when an automaton changes its state.

A timed automaton is a tuple (S, S_0, C, A, E, I), where S is a set of states, $S_o \in$ S is the initial state, C is the set of clocks, A is a set of actions and co-actions, $E \subseteq$ S x A x D(C,In) x F(C,In) x S is a set of edges between states with an action, a guard and a set of clocks to be reset and I: $S \rightarrow$ B (C) assigns invariants to locations. See figure 3.

A guard (firing condition) satisfies the following conditions: it evaluates to a Boolean; only clocks, integer variables, and constants are referenced; guards over clocks are essentially conjunctions. An assignment label is a comma-separated list of expressions only referred to clocks, integer variables, constants and only assigns integer values to clocks or integer variables.
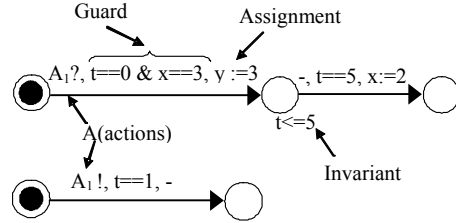

Fig. 3. Timed automata

4.2 Priority AND gate

Priority AND gate definition (according to figure 2) determines that the fault happens if its input basic events occur in a specific order. 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.

The state automaton modelling the gate is depicted in figure 4. The first edge shows that the action is the occurrence of the event a (represented by a rising edge). The guard is the condition that b is not true. The edge leading to the *Fault* state has the action "occurrence of the event b". Its guard is the constraint that a must be persistent. Hence, the automaton reaches the fault state only if the events occur in the given order. A third edge produces the return to the *Initial* state. It symbolizes the condition that event a must still be persistent when b appears. If this is not the case (falling edge of a), the fault is not produced. The property to verify, obtained from this gate, holds if the *Fault* state is never reached.

4.3 Exclusive OR gate with condition on one of inputs

In this gate, the fault is produced only if the conditioned input b appears before the event a that

can be produced later or not at all. The proposed automaton modeling this behavior is also shown in figure 4. The state *Fault* is reached if there is a rising edge of *b* (occurrence of the event) with *a* being false acting as a guard. As before, the property to verify, obtained from this gate, holds if the *Fault* state is never reached.
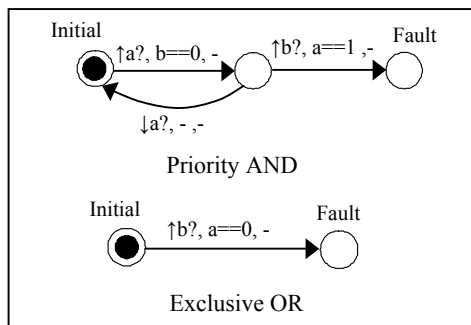

Fig. 4. Timed automata for temporal gates

4.4 Timed gates FORPAST n and WITHIN n

FORPAST n. The gate output is true if the event *a* is true now and for the last *n* time units in the past. The corresponding automaton is depicted in figure 5. The clock *t* is automatically increased with the physical time. As soon as *a* happens (rising edge) *t* is initialized. One transition leads from the initial state to an intermediate state.
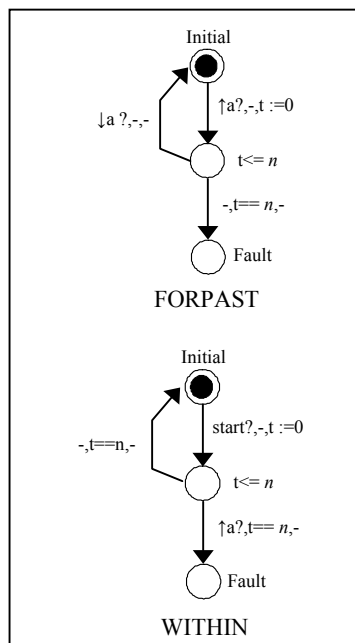

Fig. 5. Timed automata for timed gates

From this intermediate state, where the value of the clock must be smaller than or equal to *n*, two transitions are possible: one of them leading back to *Initial* depending whether the event *a* is transient and becomes false (falling edge) before the clock reaches the value *n*. The second edge leads to *Fault* if *t* equals *n*, i.e. *a* is always true during the *n* time units. The property to check holds if this *Fault* state is never reached.

WITHIN n. In this case, the fault happens if *a* is either true now or at some instant within the last *n* time units. The proposed automaton for this gate (shown also in figure 5) behaves as follows: in the first edge, a signal called *"start"* determines the beginning of the interval between *t=0* and *t=n* (at the same time *t* is initialized). The *Fault* state is reached if a rising edge of *a* is produced within this interval. In the opposite case, when *a* doesn't happen in the interval, the automata moves to the *Initial* state. Here again, the model checker must verify that the sequence leading to the *Fault* state is not produced.

5. CASE STUDY

The case study is developed on the third station of a Bosch mechatronics system (figure 6) whose aim is to assembly/disassembly gear wheels. Only automatic operations will be considered.

5.1 The process

The principal elements composing the station are a carriage, a linear conveyor, an insertion press (INPress) and its cylinder feeder (INFeeder), an extraction press (OutPress) and its cylinder feeder (OutFeeder). At the initial position (receiving position at the left) of the station the carriage waits for a gear from the previous station, which issues the signal to start the process and transmits a signal which indicates the presence or absence of a plain bearing. The linear conveyor brings the carriage either to the insertion or the extraction press device. Parts with plain bearings run to the extraction press and parts without a plain bearing to the insertion press. If the presence of a plain bearing was previously detected, the plain bearing is pressed out. If none is present, one is pressed in. The gear is conveyed by means of a feeder unit from the carriage to the working space of the press. In the case of the insertion press, a plain bearing is simultaneously fed from a drop magazine located behind the press. After the pressing step, the feeder unit takes back the piece into the carriage, which at its turn, runs to the transfer position (right) where the gear is transferred to the following station. Then the carriage runs back to the receiving position and the process starts again. A logic controller commands the global process.

5.2 First analysis: A part with bearing is transferred into the insertion press.

In the analysis, we consider events related to physical components, to the controller and faults that are combinations of these two kinds (edged respectively in fault trees by a dot line, a double line, and a simple line as is shown in figure 7). The undesirable event to analyze is "a part with bearing is transferred into the insertion press". This fault can produce the press breaking by trying to insert a bearing into a part already containing one. In a first stage, two faults linked by an OR gate produce the undesirable event. See figure 7.
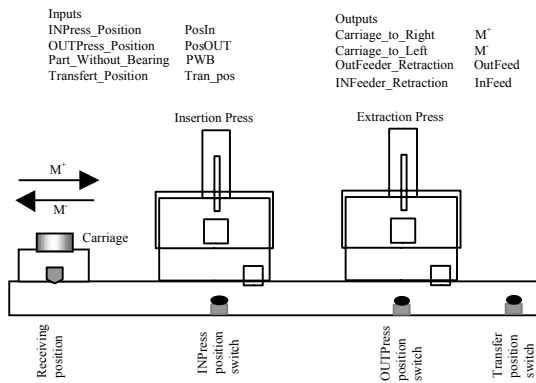
Inputs
INPress_Position   PosIn
OUTPress_Position   PosOUT
Part_Without_Bearing PWB
Transfer_Position   Tran_pos

Outputs
Carriage_to_Right   M⁺
Carriage_to_Left   M⁻
OutFeeder_Retraction OutFeed
INFeeder_Retraction InFeed

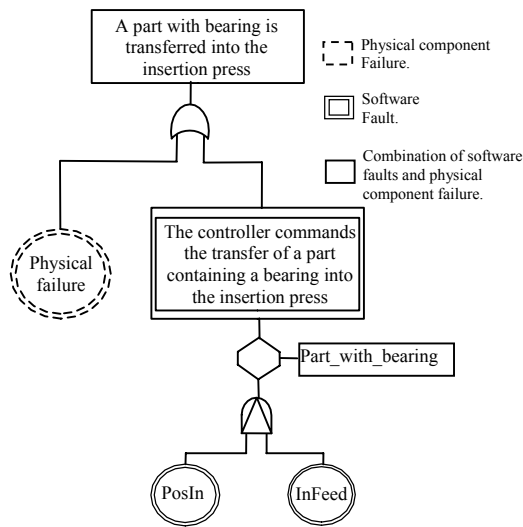Fig. 6. Diagram of the assembly/disassembly station

Fig. 7. Fault tree for "a part with bearing is transferred into the InPress"

The undesirable event is produced because of a physical failure or because the controller failed in the signal interpretation and delivers the erroneous command introducing the part into the INPress even if the condition signal Part_with_bearing is true. The controller fault (in double line box) is the result of a Priority AND gate where the fault occurs if at first the signal of the position switch in front of INPress is set, and then the controller commands the retraction of the INFeeder cylinder when the carriage is still in front of the press (PosIn). This sequence itself is not really a fault. It is only a fault if the condition signal Part_with_bearing is present (shown by the condition gate), i.e., it has been well transmitted from the previous station. If the carriage leaves the position PosIn before the cylinder is retracted the fault does not happen because PosIn is not persistent. Notice that we have included the normal events "PosIn" and "InFeed". This is essential here to derive the formal property that can model effectively the event of an erroneous commission of the controller output.

The state automaton modeling the controller fault is depicted in figure 8. The first transition contains the condition of the condition gate, the only variation from the generic model. The verification has to show that the state *Fault* is never reached.
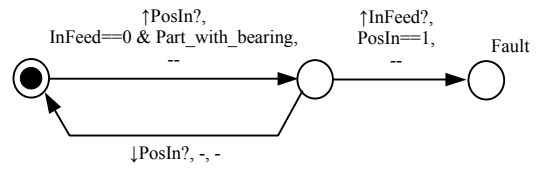

Fig. 8. Timed automaton for the first analysis

5.3 Second analysis: fault in carriage control

The second undesirable top-level event to analyze is "Erroneous Commission of carriage_to_left command before the transfer position is reached". The faults indicates an error of the controller that commands the return of the carriage (M⁻) even if the transfer position (Tran_pos) is not true. The gate Exclusive OR with condition on one of inputs is used to represent this fault (see figure 9.a). The faulty behavior is the occurrence of the event M⁻ (rising edge) before Tran_pos is true. It is important to remark that, to represent this fault, it is not possible to use a simple AND gate with inputs ¬Trans_pos and M-, like one could think at first sight. This would lead to an incorrect fault detection in the normal operation when the carriage has left the transfer position to go to the left. At that moment Tran_pos would be false and M- still true, what would lead to the incorrect fault detection. The associated timed automaton is shown in figure 9.b.
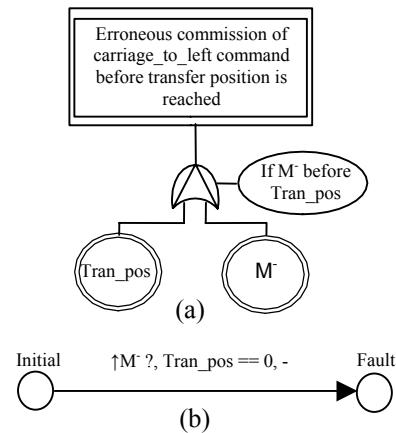

Fig. 9. Fault tree and timed automata for "Erroneous commission of carriage_to_left"

5.4 Third analysis: fault in extraction press feeder

The last case is "omission of the OutFeeder retraction when the carriage is in position". This fault means that there is no retraction movement of the feeder cylinder immediately after the carriage is in position in front of the extraction press. Here we want to treat a missing reactivity. The controller fails to set the retraction output in an interval of *n* time units starting from the event "PosOut is set". See figure 10(a). Two causes, linked by an AND gate, produce the undesired event: "signal of the position sensor is set" and "OutFeed command is omitted".
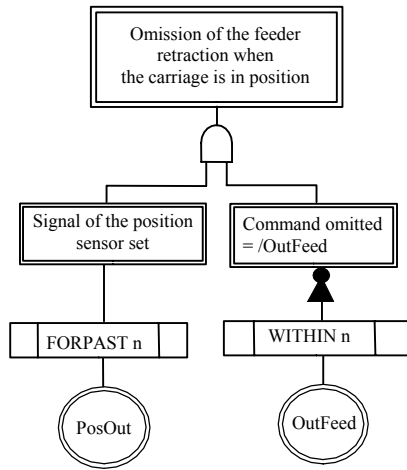
(a)
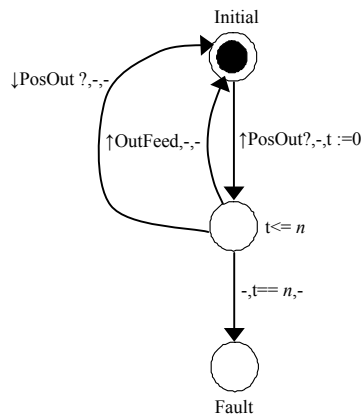
(b)

Fig. 10. Fault tree for "Omission of feeder retraction" and timed automaton

The first one is a condition and is the output of the temporal gate FORPASTn meaning that the sensor signal PosOut is true now and for the last *n* time units in the past. The cause in the right branch of the tree is a fault of the controller which omits to deliver the command OutFeed (NOT gate) either now or at some instant within the last same *n* instants started when PosOut is true. The temporal gate WITHINn describes this situation.

As it is shown in the figure 10.b, an automaton modelling the function of both temporal gates can be used to represent the faulty behaviour. For the integration the same clock *t* is used which is automatically increased with the physical time. As soon as PosOut happens (rising edge), *t* is initialized and the time starts counting. This signal is used for the synchronization of both automata. One transition leads to an intermediate state where the value of the clock must be smaller than or equal to *n*. Three transitions are then possible: one leading back to *Initial* depending whether the signal PosOut is transient and becomes false before the clock reaches the value *n*. The other one, also leading to *Initial*, if the signal OutFeed becomes true (the controller sets the command) before the clock reaches the value *n*, i.e. at some instant within *n*. Finally, the third

transition leads to *Fault* if *t* equals *n*, and the signal OutFeed is not produced at all .

6. CONCLUSIONS

The approach developed in this paper proposes to facilitate formal properties elaboration by means of a preliminary fault-tree analysis extended with systematic faults coming from logic controller. An important result of this work is the formal modelling of temporal and timed gates behaviour using timed automata. The global method presented in figure 1 has been used to verify logic controllers developed in standardised languages like ladder diagram. The model-checking tool employed in these experiments was UPPAAL. Several prospects can be drawn from this work. Consistency checking of fault trees involving the same variables and minimal cut sets computation including temporal and timed gates based in automata composition are challenging ones.

REFERENCES

Alur, R. and D.L. Dill (1994). A Theory of Timed Automata. *Theoretical Computer Science*, **Vol. 126**, pp. 183-235.

Barragan, I. And J.-M. Faure. From fault-tree analysis to model-checking of controllers. *Proceedings of 16th IFAC World Congress, CDROM paper n° 4596, Praha (CZ),* July 4-8, 2005.

Behrmann, G. et al (2004). *A Tutorial on UPPAAL*. Department of Computer Science, Aalborg University, Denmark.

Bérard, B. et al (2001). *Systems and Software Verification. Model-Checking Techniques and tools*. Springer.

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.

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.

Ortmeier, F. et al. (2004). Combining formal methods and safety analysis – The ForMoSA approach, LNCS **3147**, pp 474-493.

Palshikar, G.K. (2003). Temporal Fault Trees. *Information and Software Technology*, **n° 44**, p137-150.

Reif, W. (2004). Integrated formal methods for safety analysis of train systems, *Proceedings of IFIP WCC 2004,* August, 22-27, 2004, Toulouse, France.

Schäfer, A. (2003). Combining real-time model-checking and fault-tree analysis, LNCS **2805**, pp 522–541.

US Nuclear Regulatory Commission (1981). Fault Tree Handbook. *Technical Report NUREG-0492, Washington, DC*