

Fault Tree Analysis with Moby/FT

Johannes Faber

Department of Computing Science, University of Oldenburg, Germany
johannes.faber@informatik.uni-oldenburg.de

1 Introduction

Fault tree analysis (FTA) is a standardised technique used by engineers to explore possible failure states of safety-critical systems. Since the FTA is generally practised informally and thus, it cannot be guaranteed that it is applied correctly, several formal approaches have been introduced in the past (e.g. [STR02]) for allowing engineers to verify that they built a fault tree considering the right events (*correctness*) and that they did not forget a relevant event (*completeness*). But without automated proof support the acceptance of such formal approaches in industry is insufficient.

In this paper we present the tool Moby/FT, which offers a graphical editor for specifying and verifying fault trees. The aim of Moby/FT is to increase the reliability of real-time systems, which are often profoundly safety-critical. The tool allows the user to specify fault trees together with the semantics of their events in terms of an interval logic, Duration Calculus [ZH04], and to automatically verify the correctness and completeness of the fault trees relatively to the system model.

Hence, we introduce a novel method for the automatic verification of fault trees using the continuous real-time model-checker Uppaal [UUP05] for timed automata [AD94]. Therefore, fault trees with an DC semantics have to be transformed into timed automata, which can be verified by Uppaal. The connection to Uppaal, i.e. the generation of the timed automata and the appropriate requests, is automatically performed by Moby/FT.

2 Fault Tree Analysis.

The semi-formal fault tree analysis (FTA) technique is described in the IEC standard [IEC90]. The aim of the FTA is to analyse the conditions and factors causing an undesired, so-called *top event*. The fault tree itself is a graphical representation of these causes as a tree where the top event is stated as the root node. The other nodes represent either events that are direct or indirect causes of the top event or gates. Gates link two or more *cause events* causing one *fault event* in the suitable manner. For example, an *or*-gate is used to express that one cause event suffices to enforce the fault. Whereas an *and*-gate is chosen to express that all cause events are needed to enforce the fault. Figure 1 gives an example of a fault tree (taken from [VGRH81]) in Moby/FT for a pressure tank

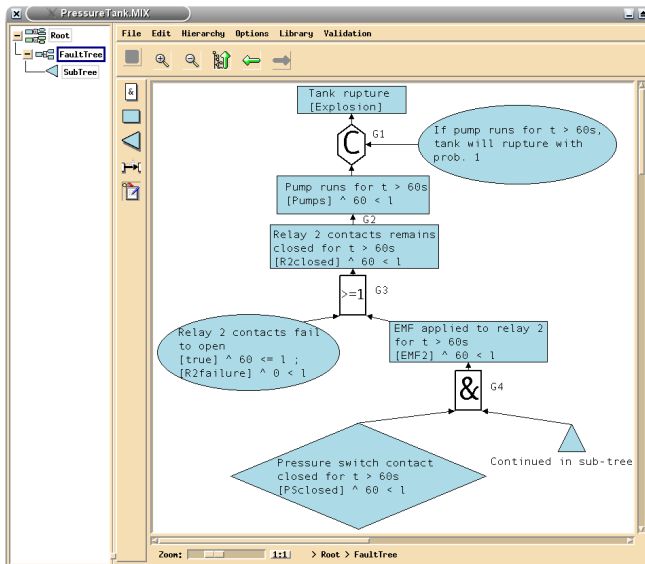


Fig. 1. Moby/FT with a fault tree from the pressure tank example.

system, where a tank is filled by a pump. It will rupture if filled for more than 60 sec. So, the top event of the fault tree is **Tank rupture**. The direct cause event for this fault is **Pump runs for $t > 60$ sec**. The used gate is an **inhibit-gate**, where an additional constraint can be declared that must also be valid to cause the fault event.

We apply the terms *correctness* and *completeness* to define the semantics of the gates. Correctness intuitively means that the cause events of one gate always imply the fault event of this gate. Completeness means that the occurrence of the fault event always implies the occurrence of the causes. For example, the **and-gate** G4 in Fig. 1 is complete, because the fault event implies the *simultaneous* occurrence of the cause events. For the completeness of an **or-gate** (gate G3) the fault event has to imply only one of the cause events. It is important to remark that in practice the completeness is more important for a system, since engineers use the FTA to discover all possible reasons for a top event. The completeness of trees ensures that no fault was forgotten.

Like in [Sch03] we use an interval-based logic for describing the formal semantics of gates, but according to [STR02] we extend this approach with additional gates, e.g. the **inhibit-gate**. The *Duration Calculus with Liveness* (DCL), introduced in [Ska94], is an extension of the well-known Duration Calculus [ZHR91, ZH04]. It introduces special modalities to express liveness properties. In the DCL, like in the DC, time-dependent variables called *observables* are used to describe system states. The formal semantics of the gates is formulated in [STR02] for every gate as an implication in an DCL formula. The semantics of the fault events is given by general DCL formulas.

3 The Verification Concept in Moby/FT

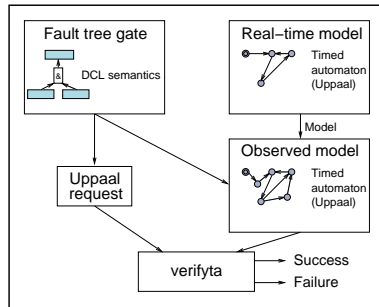


Fig. 2. The verification concept.

For enabling the verification of the completeness and correctness of fault trees with Uppaal we have to bring together the different worlds of the DCL formalism on the one hand and the Uppaal timed automata on the other hand. Since DCL is more expressive than timed automata, we curtail the possible event formulas to some event patterns. Following [Gór94] three patterns occur in fault trees. These are *simple events* with a defined duration, *irreversible events*, and sequences of events. In our example the event **Tank rupture** is an irreversible event and **Pump runs for $t > 60$ sec** is a simple event. Figure 2 outlines the concept used for the verification of fault trees in Moby/FT. We assume a real-time system modelled as a timed automaton in Uppaal and a fault tree belonging to this system. Since the Uppaal logic can just formulate simple reachability requests in a subset of TCTL and thus is too weak to express the event formulas directly, we have to find a way to ‘observe’ the behaviour, specified by a fault tree, in the timed automata. In our approach, we observe the automata by extending them with additional states and using test automata to detect the occurrences of events. By this means, event formulas, which cannot be formulated in the Uppaal logic directly, can be verified by reducing the DCL formulas to simple reachability requests. The *observed model* (Fig. 2) comprises the parallel composition of the extended Uppaal model and the test automata. Simultaneously, the Uppaal request is generated, depending on the gate type and the appropriate event patterns. Afterwards, Moby/FT invokes the Uppaal stand-alone model-checker **verifyta**.

Transformations and Test Automata. Since there is no implicit connection between the DCL formulas of the fault tree and the locations of the timed automata, we associate the DCL observables with the locations of the same name. For example the observable **Explosion** describing the event **Tank rupture** is associated with the location **Explosion** (cf. Fig. 3) such that the observable is evaluated to true while the automaton remains in the associated location.

For detecting the occurrence of a fault event specified by an DCL formula we use test automata that synchronise with transitions entering locations that belong to the respective observables. Hence, the test automata *observe* the behaviour of the observables. The verification process can be split into three parallel steps:

1. The locations belonging to observables have to be observed. This is achieved by inserting additional locations, which are always entered directly before the observed locations. The inserted transition between the new location and the observed one is labelled with a new synchronisation label (cf. Fig. 3, the top right automaton).
2. Test automata are added that synchronise with the new transitions of step 1. These automata have *acknowledgement locations* entered when an event

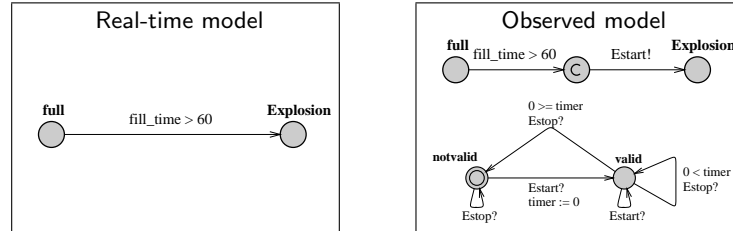


Fig. 3. Observation of the location **Explosion**.

in the original system model occurs. The test automata depend on the types of the gates and the formulas of the respective fault events. Figure 3 gives a simple example for an observation detecting whether an event like **Tank rupture** has ever occurred. The bottom right automaton is a test automaton, which enters the acknowledgement location **valid** at the same time that the observed location **Explosion** is entered. [Fab04] contains a more comprehensive list of test automata for DC patterns, e.g. for sequences of events.

3. The Uppaal request has to be generated. It consists of a direct transformation of the DCL formulas for the completeness and correctness to the Uppaal logic applying the acknowledgement locations of the test automata.

4 Conclusions and Related Work

Moby/FT provides the fully automatic verification of fault trees using the novel verification technique. By this means, it assists engineers with identifying potential faults and with analysing the conditions causing an undesired event. The users do not have to be familiar with the DC formalism but with the intuitively comprehensible event patterns. Moby/FT and the verification process are described in detail in [Fab04], where in particular, the correctness of the underlying transformations is proven. Moreover, there we present the entire pressure tank example and were able to show the completeness of its fault tree with Moby/FT. Future work will support additional model-checkers and Moby/FT will be integrated into the Moby/RT environment, which already comprises different real-time applications like Moby/PLC and Moby/CD. Moby/FT can be downloaded for free in a version for Linux and Solaris on the Moby home page [Mob05].

Another work treating the real-time model-checking of fault trees can be found in [Sch03], where a similar fault tree semantics is used, permitting less events, and the verification is realised by transforming the fault trees into phase automata and applying the model-checker Moby/DC. But fully-automated tool support is lacking. It is intended to integrate a Moby/DC connection in Moby/FT for allowing the user to choose between different model-checkers and to compare the different verification techniques. In [OST⁺02] and [Thu04] discrete-time model-checking for the verification of fault trees is used, in the former paper with the model-checker RAVEN and in the latter with the interactive verifier KIV. Test automata expressing real-time properties are also applied in [DL02].

References

- [AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
- [DL02] H. Dierks and M. Lettrari. Constructing test automata from graphical real-time requirements. In W. Damm and E.-R. Olderog, editors, *FTRTFT '02: Formal Techniques in Real-Time and Fault-Tolerant Systems: 7th International Symposium*, volume 2469 of *LNCS*, pages 433–453, Oldenburg, June 2002. Springer-Verlag.
- [Fab04] J. Faber. Fehlerbaumverifikation durch Modelchecking mit Uppaal. Master's thesis, University of Oldenburg, 2004. In German.
- [Gór94] J. Górski. Extending safety analysis techniques with formal semantics. In Felix Redmill, editor, *Technology and assessment of safety-critical systems: Proceedings of the Second Safety-Critical Systems Symposium*, pages 147–163. Springer Verlag Berlin, 1994.
- [IEC90] IEC 61025: Fault tree analysis (FTA), 1990.
- [Mob05] Moby product home page. Department of computing science, University of Oldenburg, <http://csd.informatik.uni-oldenburg.de/~moby>, 1998-2005.
- [OST⁺02] F. Ortmeier, G. Schellhorn, A. Thums, W. Reif, B. Hering, and H. Trappschuh. Safety analysis of the height control system for the Elbtunnel. volume 2434 / 2002 of *LNCS*, pages 296 – 308. SAFECOMP 2002, Springer-Verlag Heidelberg, January 2002.
- [Sch03] A. Schäfer. Combining real-time model-checking and fault tree analysis. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME '03: Formal Methods*, volume 2805 of *LNCS*, pages 522–541, Pisa, Italy, September 2003. Springer-Verlag.
- [Ska94] J. U. Skakkebæk. Liveness and fairness in Duration Calculus. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR '94: Concurrency Theory, 5th International Conference*, volume 836 of *LNCS*, pages 283–298, Uppsala, Sweden, August 1994. Springer-Verlag.
- [STR02] G. Schellhorn, A. Thums, and W. Reif. Formal fault tree semantics. In *IDPT '02: Proceedings of The Sixth World Conference on Integrated Design & Process Technology*, Pasadena, CA(USA), 2002.
- [Thu04] A. Thums. *Formale Fehlerbaumanalyse*. PhD thesis, Universität Augsburg, 2004.
<http://www.opus-bayern.de/uni-augsburg/volltexte/2004/38/>. In German.
- [UUP05] Uppaal home page. Basic Research in Computer Science (University of Aalborg) and Department of Computer Systems (University of Uppsala), <http://www.uppaal.com>, 1995-2005.
- [VGRH81] W.E. Vesely, F.F. Goldberg, N.H. Roberts, and D.F. Haasl. *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission, 1981.
- [ZH04] Zhou Chaochen and M. R. Hansen. *Duration Calculus*. Springer-Verlag, Berlin Heidelberg, 2004.
- [ZHR91] Zhou Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, December 1991.