Technische Universität Kaiserslautern

Safety and Reliability of Embedded Systems

(Sicherheit und Zuverlässigkeit eingebetteter Systeme)

Fault Tree Analysis
Obscurities and Open Issues

Technische Universität Kaiserslautern

Content

What are Events?

- In probability theory, everything that is true with a certain probability is called “event”
- In software / systems engineering (and in common language), an event is something occurring at a given point in time
- In FTA, events can be
  - Sudden events ("Bolt breaks")
  - States or conditions ("Valve is blocked")
  - (Informal) propositions ("Fire is not detected by supervisor")

☝ **All of them may be useful, but specify clearly what you mean**

- Note the differences regarding probabilities
  - States / propositions have a probability (at a given time)
  - Events have a probability density or rate
- (Out-dated) DIN 25424 features appropriate formulas for probability and probability density

Examples for Problematic Event Semantics

Priority AND: Output event occurs when all input events occur in a specific order

- If FT events are sudden events: When does output event occur?
  - E.g. simultaneously to the last of the input events
- If output event is a logical proposition: The proposition is true (all the time!), if the input events do occur in the right order
  - But can input events also be propositions then? How can then Input 1 occur before Input 2?
- If FT events are states / conditions / predicates that can be true or false at given times
  - Input 1 can become true before Input 2
  - Output condition is true upon the time when the last input condition is true

Inhibit, Enabler / Conditioning Events

Inhibit: Output event occurs when input event occurs and inhibit event is not true

- ☐ Inhibit event has state / condition semantics
- ☐ Sometimes enabler events (states) are distinguished from initiator events (trigger semantics)
- ☐ Separate symbols are available

Basic Event / Initiator

Conditioning Event / Enabler

The Use of NOT

MCS based probability calculation is not suitable for negated events!

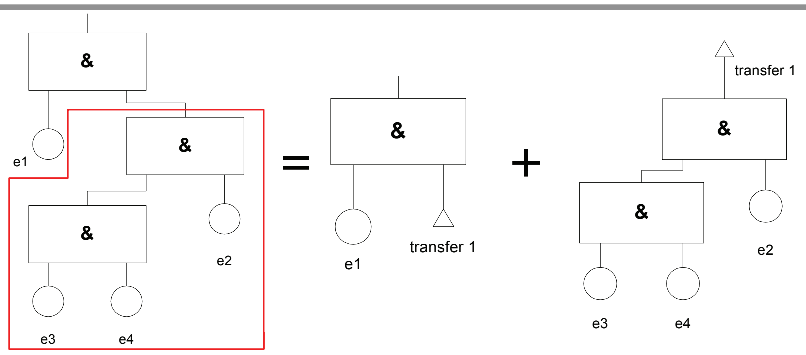
- ☐ A FT without NOT is called "coherent"
  - "A system can never get better if more components fail"
- ☐ Some NOTs are virtual (e.g. in transcriptions of voter, XOR, ...)
- ☐ If FT events are sudden events
  - How can their occurrence be negated?
  - In this case, NOT makes no sense
- ☐ If FT events are states / conditions / predicates / logical propositions
  - Negation makes sense
  - BDD algorithm can handle negated variables
  - Minimal cut sets → Prime implicants (may contain negated events)

cf. John Andrews: “To Not or not to Not”

Generalization vs. Causation Gates

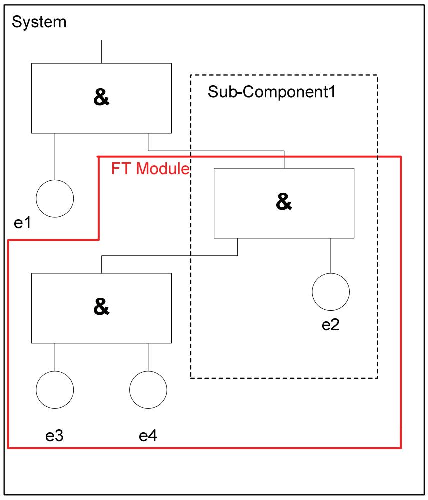
- Sometimes gates suggest causality
  - Electrical short circuit OR defective gas tube ⇒ fire
- Sometimes gates suggest generalization / decomposition
  - Engine defective OR tire defective = car defective
- In original FT standards no distinction, some researchers do distinction
  - Sometimes, two pairs of AND / OR are proposed [Gorski]
  - Some say that AND means causation and OR means decomposition [FT Handbook]
- Whether or not FTs express causality at all can be discussed...

Issues about Decomposition



- FTs are hierarchical by nature
- Traditionally, FTs are decomposed by modules (independent sub-trees)
- Each module is replaced by a single event with the same probability
- Each module can be analyzed independently
- Alternatively, partitioning into pages by transfer symbols

FT Decomposition by Modules



Modules are <u>independent</u> sub-trees

But:

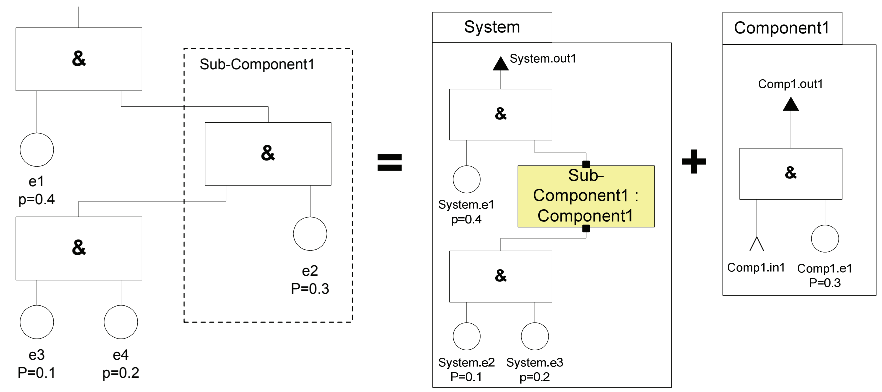
Technical components often influence each other

Technical components are not always modules!

Deficiencies of Classical Module Concept

- Module borders may be orthogonal to component borders
- Attachment of (partial) fault trees to components is not possible if components have external influences
- Division of labor (e.g. supplier/integrator) is not possible
- Modeling of some component by other models than fault trees is not possible
- Partitioning of fault trees into pages (using transfer ports) is a solution to some degree, but still no division of labor or reuse

Recent Approach: Component Fault Trees



Here, "component" means technical unit.
Components are connected by ports like in architectural models.
New paradigm: Components represent Boolean formulas

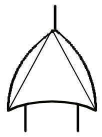
Temporal Functions

- Events can be understood as propositions that are true or false at each given point of time
- Supplying time functions instead of constant probabilities allows to plot reliability/availability function for the complete system
  - Events can be exponentially distributed, Weibull distributed etc.
  - Useful in combination with Markov analysis
  - Mission time for each event or sub-component specifies time
- Important special case: Exponential distribution
  - $P(t) = 1- e^{-\lambda t}$, λ: Failure rate; P: Probability, that component has failed
  - OR leads to an exponential function as gate output, but AND does not!

Attention when representing events by their occurrence rate:
Output function is not always exponential!

Temporal Relations between Events

- ☐ Standard FTA is based on Boolean logic and cannot handle temporal sequences
- ☐ Priority AND expresses probability that event 1 occurs before event 2
  - Only useful if time functions (and not static probabilities) are used
  - Probability of system failure before t is probability that E1 occurs before some intermediate point of time and E2 occurs after that point, accumulated for all points of time between 0 and t



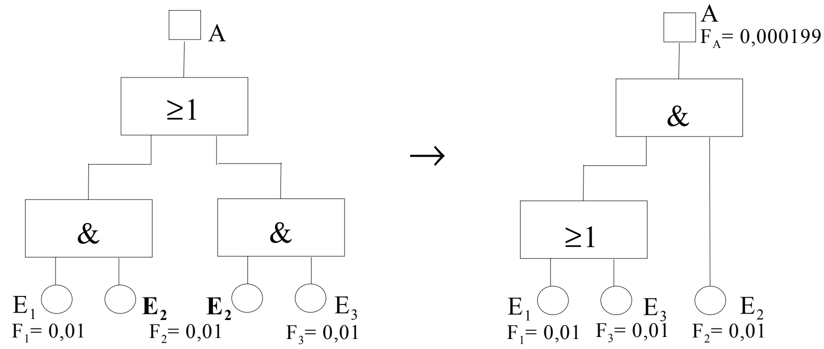
E1 E2

$$P_{failed}(t) = \int_0^t f_2(t_2)\left(\int_0^{t_2} f_1(t_1)\,dt_1\right)dt_2$$

Dependencies

- ☐ Stochastic independence is an important assumption for combinatorial approaches
- ☐ Repeated events as special case of dependency can be handled by restructuring the Boolean formula
- ☐ For special cases (e.g. spares), there are solutions
- ☐ When probabilities are small, errors may be negligible if dependencies are not taken into account
- ☐ Correct calculation in presence of arbitrary dependency is only possible by state-based models
- ☐ Functional dependency gate in DFT allows to express secondary faults

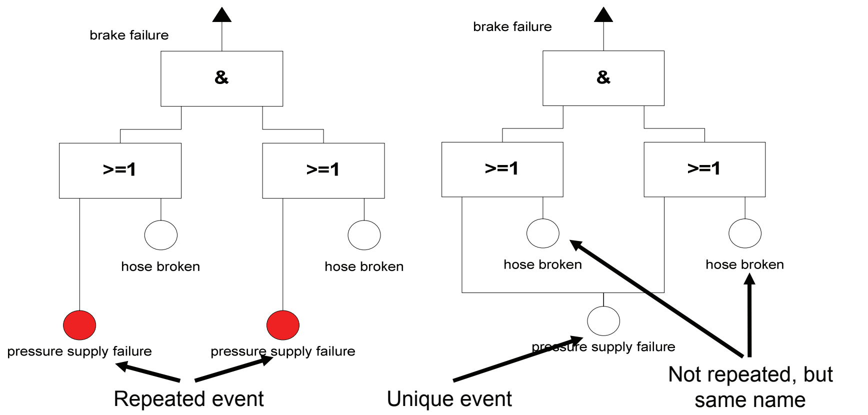
Elimination of Repeated Events



- By restructuring the Boolean formula, repeated events can be avoided

ENGINEERING SOFTWARE DEPENDABILITY

Repeated Events: Tree vs. DAG



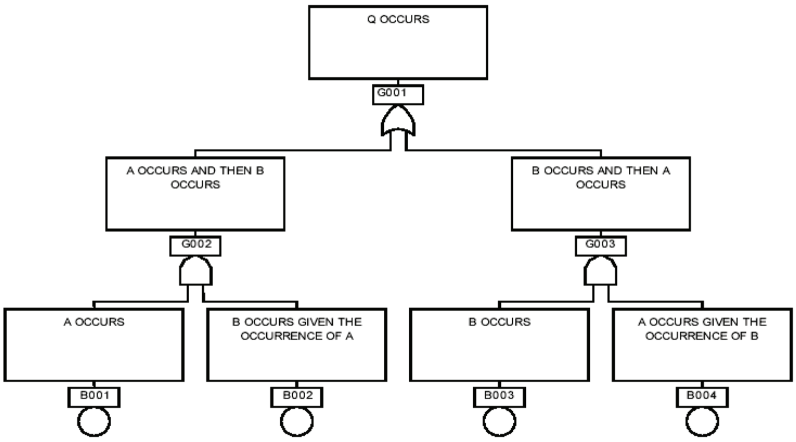
- Directed Acyclic Graphs (instead of trees) eliminate repeated events
- A namespace concept is desirable (implemented in Component Fault Trees)

ENGINEERING SOFTWARE DEPENDABILITY

Spares and Spare Pools

- ☐ A special case of dependency is the usage of spares
- ☐ System fails if primary unit fails and spare also fails or has already failed before
- ☐ While the primary unit is operating, spares fail
  - not at all → cold spare
  - at a reduced rate (specified by a factor) → warm spare
  - at the normal rate (hot spare)
- ☐ Special spare gates have been proposed
- ☐ Spare pool: Not only one spare unit, but n spare units with identical failure rate

Explicit Dependencies



Other ways to deal with dependencies

- ☐ Functional Dependency Gate (in some packages)
  - Output event occurs necessarily if input occurs
  - Output event can also occur spontaneously

☝ **Many attempts have been made to bring special cases of dependency into FTA. At a certain point, state-based models (e.g. Markov chains) are a better fit (but analysis is usually much slower!)**

Integration with other models

- ☐ Fault trees can and should be integrated with
  - Markov chains: to describe basic events in presence of dependencies
  - Event trees: to describe consequences of the top-event
- ☐ Fault trees can and should be used in conjunction with
  - FMEA
  - other hazard analysis models (Preliminary Hazard Analysis, Common Cause Analysis, ...)
  - general systems and software modeling techniques

☝ **Formal integration with software / systems modeling is desirable, but not yet achieved to a satisfactory degree**

FTA for Software

- ☐ FTA stems from an era when (at least) safety critical systems were purely electrical / mechanical
- ☐ They have "working/failed" semantics
- ☐ They cannot natively capture the dynamic nature of software
- ☐ There have been several attempts to apply FTA to software or to derive FTs from software
  - partly based on source code statements
  - partly based on statecharts or formal methods

☝ **Applying FTA to software controlled systems is still an issue**

Is there a formal semantics for FTA?

- ☐ FTs are intuitive, but unclear semantics is an issue when
  - constructing FTs automatically
  - integrating FTs with other kinds of models
  - using dynamic extensions
- ☐ Some recent research work about formalizing FTA
  - formalizing the meaning of gates
  - proving that FT is complete and consistent
- ☐ Different formalisms used
  - Z (algebraic specification)
  - Interval Temporal Logic, Duration Calculus (temporal/real time logic)
  - Translation into Markov chains and different kinds of probabilistic Petri nets

**Formalization is not agreed upon.
Different researchers use different approaches**

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Literature

- Component Fault Trees
  - Kaiser, B., Liggesmeyer, P., Mäckel, O.: A New Component Concept for Fault Trees. Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software (SCS'03), Canberra Conferences in Research and Practice in Information Technology, Vol. 33. P. Lindsay & T. Cant, Eds.
- Usage of Not
  - Andrews J.D. "The Use of not Logic in Fault Tree Analysis", proceedings of the 14th ARTS, Advances in Reliability Technology Symposium , Manchester, Nov 2000
- Priority AND, Dynamic Fault Trees, State-Event Fault Trees
  - On The Quantitative Analysis Of Priority AND Failure Logic, IEEE Transactions On Reliability (R-25 No 5), 1976, p324-326, JB Fussell & EF Aber & RG Rahl
  - Manian, R., Dugan, J. B., Coppit, D., and Sullivan, K. J. Combining various solution techniques for dynamic fault tree analysis of computer systems. In 3rd IEEE International High-Assurance Systems Engineering Symposium (1998), IEEE Computer Society, pp. 21-28.
  - Kaiser, B., Gramlich, C.: State-Event-Fault-Trees - A Safety Analysis Model for Software Controlled Systems. Safecomp 2004, September 21-24 2004, Potsdam, Germany