Attack Trees vs. Fault Trees: two sides of the same coin from different currencies[$]

Carlos E. Budde[1], Christina Kolb[1], and Mariëlle Stoelinga[1,2]

[1] Formal Methods and Tools, University of Twente, Enschede, the Netherlands
[2] Department of Software Science, Radboud University, Nijmegen, the Netherlands

Abstract This work compares formal approaches to define and operate with attack trees and fault trees. We start by investigating similarities between the syntactic structure, semantics, and qualitative analysis, of static attack trees and fault trees. Then we point out differences of the analysis methods and metrics between the two formalisms, providing a deeper insight for their dynamic variants. Finally, we overview several extensions and categorise them using the new concept of dimension, which allows us to compare these extensions and point out research gaps.

1 Introduction

Attack trees (ATs) and fault trees (FTs) are popular formalisms that support the identification, documentation, and analysis of security (resp. safety) risks. They are part of system-engineering frameworks such as SysML-Sec [28], and count with commercial tools such as Isograph's AttackTree and FaultTree+ [14, 15].

The popularity of these formalisms in industry is due to their capacity to represent complex processes succinctly and to the desired level of detail. In AT (resp. FT) analysis, a hierarchical diagram is designed to systematically map security (resp. safety) hazards. The resulting model gives insight into the vulnerabilities of the system, which can then be countered cost-efficiently [18, 34].

The origins. This analogous procedural approach is no coincidence: ATs were inspired on FTs. The latter were introduced in 1961 at Bell Labs to study ballistic missiles [33, 30]. In 1990 FT analysis was "about 39 years old, and has become a well-recognized tool worldwide" [9]. In contrast, Weiss introduced threat logic trees—the origin of ATs—in 1991, and its "similarity. . . to fault trees suggests that graph-based security modelling has its roots in safety modelling" [21].

Ever since, these formalisms increased their modelling and analysis power to best satisfy the needs of the safety or security application domain. This has separated the syntax and semantics of new FT- and AT-based formalisms, in spite of their sharing the modelling principle of top-down hierarchical decomposition. In this work we study this disjoint evolution from the perspective of formal methods. We first show in Sec. 2 that the syntax of their so-called static

[$] This work was partially funded by NWO project 15474 (*SEQUOIA*) and ERC Consolidator Grant 864075 (*CAESAR*).

versions, as wells as their corresponding semantics and qualitative analysis, are mathematically equivalent. The only distinction between static FTs and ATs as a formalism is their domain ontology: safety vs. security. This is the root of their subsequent differentiation, which we study in Sec. 3. To compare their extensions in a systematic manner we introduce the notion of dimension, which allows us to contrast parallel (even symmetrical) evolutions. The work concludes in Sec. 4.

Formalism. A *formalism* is defined by **1.** an (unambiguous) *syntax* to represent its elements, called *models*; **2.** a *semantics* that maps each model to a (unique) mathematical object; **3.** a *domain ontology* in which the models are interpreted. The three parts of this preliminary definition are formalised in the sequel.

Related work. Surveys on FTs are [18] and [30]. The latter covers modelling and analysis tools. The former pinpoints limitations of FTs to assess the reliability of static systems (only), and mentions extensions that overcome them, e.g. dynamic FTs [8], state-event FTs [19], and Stochastic Hybrid FT Automata [6]. Standard FT analysis and its extensions are also extensively discussed in [30], including technical details of different FT models and analyses.

Surveys on attack trees, [21] and [34], present the state of the art in graphical-hierarchical attack/defense modelling. The latter is a modern and comprehensive summary on the use of formal methods to enhance security evaluation. It references software tools, and discusses steps for industrial technology transfer.

2 Similarities between fault and attack trees

ATs and FTs follow the same modelling principle: an expert panel identifies a main event of interest—one *top element*—and refine it logically to the level of well-understood basic components or actions—the set of *basic elements*—[21, 30]. This analogous model-building process results in identical syntactic structures.

2.1 Syntactic structure: static FTs and ATs

The vanilla version of FTs and ATs, so-called *static* fault or attack trees, have the same syntax. We unify them under the concept of *logical-decomposition tree*.

Definition 1 (LDT). *A* logical-decomposition tree *(LDT for short) is a tuple* $T = \langle N, t, ch\rangle$ *where:* (i) *N is a finite set of* nodes*;* (ii) $t\colon N \to \{AND, OR, LEAF\}$ *gives the* type *of each node;* (iii) $ch\colon N \to 2^N$ *gives the (possibly empty) set of* children *of a node. Moreover, T satisfies the following constraints:* (a) $\langle N, E\rangle$ *is a connected directed acyclic graph (DAG), where* $E = \big\{(v,u) \in N^2 \mid u \in ch(v)\big\}$*;* (b) *$T$ has a unique root, denoted R_T:* $\exists!\, R_T \in N.\, \forall v \in N.\, R_T \notin ch(v)$*;* (c) *$LEAF$ nodes $N_L \subseteq N$ are the leaves of T:* $\forall v \in N. t(v) = LEAF \Leftrightarrow v \in N_L \Leftrightarrow ch(v) = \varnothing$.

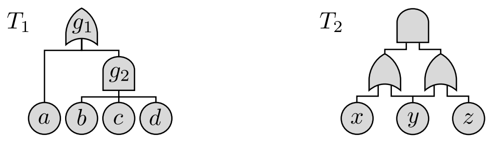
So, syntactically, static FTs and ATs are LDTs (VOT gates in static FTs are syntactic sugar of AND and OR gates). LDTs can be proper trees or DAGs: the difference is that in proper trees, each child node has exactly one parent. Node $v \in N$ is *the parent* of $u \in N$ iff u is a child of v, i.e. T has the edge $v \to u$. By definition, *LEAF* nodes are not parents: parent nodes are called *gates*.

Logical gates. An LDT represents the logical decomposition of events via disjunction and conjunction, which can be interpreted as a Boolean function. Consider e.g. a wooden gate that can break due to rotten wooden planks OR rusty hinges (or both); and the hinges rust if they are made of iron AND the environment is humid AND sufficient time elapses. This decomposition is safety-oriented. From an analogous security perspective, the wooden gate is breached by dislodging the hinges OR cracking the wooden planks, and hinges can be dislodged if the alloy is fragile AND the attacker has a crowbar AND enough strength.

Both cases yield the LDT $\langle\{a,b,c,d,g_1,g_2\},t,ch\rangle$ with leaves $N_L=\{a,b,c,d\}$ and gates $t(g_1)=AND$, $t(g_2)=OR$, s.t. $ch(g_1)=\{a,g_2\}$, $ch(g_2)=\{b,c,d\}$. This represents the Boolean function $\lambda\, abcd\,.\, a\vee(b\wedge c\wedge d)$ whose atoms take safety or security meaning: this is denoted $\mathrm{OR}(a,\mathrm{AND}(b,c,d))$ and visualised as Fig. 1a.

Visualisation. FTs and ATs are graphical formalisms, drawn as in Fig. 1 with the root on top, and every child connected to an (upper) parent by a line. Leaves are circles, and logical gates resemble their electronic-circuit counterparts.

Figure 1: T_1 has tree structure; the OR gate g_1 is the root; g_2 is an AND gate. T_2 has DAG structure: y has two parents.



(a) $\mathrm{OR}(a,\mathrm{AND}(b,c,d))$ **(b)** $\mathrm{AND}(\mathrm{OR}(x,y),\mathrm{OR}(y,z))$

2.2 Semantics & analysis

Once an LDT model T has been created, it is studied to find safety/security vulnerabilities of the system. For this, T is bestowed with formal semantics.

Structure function. These semantics can be given via a function $f_T\colon 2^{N_L}\to\mathbb{B}$ that indicates whether a set of basic elements trigger the top element of T. That is, $f_T(A)=\top$ iff the Boolean function represented by T is satisfied by the mapping $\big(A\mapsto\top\big)\cup\big((N_L\setminus A)\mapsto\bot\big)$, where $\setminus$ denotes set difference. For instance for T_1 in Fig. 1a, to evaluate $f_{T_1}(\{a,c\})$ one maps a and c to $\top$, b and d to $\bot$, and evaluates the Boolean function represented by T_1—$\lambda\, abcd\,.\, a\vee(b\wedge c\wedge d)$—which returns $\top$. This so-called *structure function* f_T is given by the syntactic structure of T, and hence it is analogous for static FTs and ATs [17, 30].

Evidence. The set $A\subseteq N_L$ on which f_T is evaluated is called *evidence*: for ATs it represents the steps carried out by an attacker; for FTs it represents elements that have failed. If $f_T(A)=\top$ then A is called *valid evidence*; else it is *invalid.* Valid evidence A is called *minimal* if no proper subset of A is valid. For instance in Fig. 1, $\{a\}$ and $\{x,z\}$ are minimal evidence of T_1 and T_2 resp., $\{a,b\}$ is also valid (but not minimal) evidence, and $\{x\}$ is invalid. In FT analysis, minimal evidence is also called “minimal cut set” or “prime implicant.”

Formal semantics. Static FTs and ATs are *coherent*: adding basic elements to evidence preserves its validity [3]. That is, if $f_T(A)=\top$, then $f_T(A\cup\{a\})=\top$ for every $a\in N_L$. Thus, all valid evidence of T—i.e. that can trigger its top element—is characterised by the collection of minimal evidence. This gives rise to the formal semantics of T: $[\![T]\!] = \{A \subseteq N_L \mid f_T(A) = \top \wedge A \text{ is minimal}\}$.[†]

Qualitative analysis. Since $[\![T]\!]$ subsumes all ways to trigger the top element of T, its computation provides key information on the vulnerability of the system. For FTs, any $A \in [\![T]\!]$ with few elements pinpoints a safety hazard—where a few basic failures can trigger a system-level failure—and likewise for ATs. The amount of subsets in $[\![T]\!]$ can be exponential on the number of nodes in T [24]. Since this hinders computations, and given the interest in small sets from $[\![T]\!]$, FT analysis sometimes bounds the size of the minimal evidence to compute [32].

Thus, static FTs and ATs are mathematically equivalent: their syntax is given by an LDT, T, and their semantics by the set of minimal evidence, $[\![T]\!]$. What sets them apart as formalisms is their *domain ontology*, i.e. their application domain: safety and security have different goals, fulfilled by enriching LDTs with (a) attributes on the leaves, and (b) new types of gates. In Sec. 3 we show how this is the root of several differences between FTs and ATs.

3 Differences between fault and attack trees

The aforementioned similarities apply only to *static* FTs and ATs. Later extensions to these formalisms, e.g. to include notions of complement and dynamic behaviour, have caused them to grow in different directions. We discuss this in Sec. 3.2 but first we show, in Sec. 3.1, that the different goals of the safety/security domains break the analogies even for static FTs and ATs.

3.1 Analyses that differ for static FTs and ATs

Quantitative analysis. Beyond the constitution of each set $A \in [\![T]\!]$, it is useful to quantify their relevance. For example, if every basic element a_i has a probability $p_i \in [0,1]$ of occurrence, one can compute the total probability of some evidence A [26]. Similarly, values $\lambda_i \in \mathbb{R}_{>0}$ can describe *the rate* at which these basic probabilities increase with time. Then one can measure the system unreliability, i.e. the probability of triggering the top element at various mission times. Rates also enable time-only measurements, such as the mean time it takes for some evidence $A \in [\![T]\!]$ to be observed [30].

All these quantitative queries, that deal with the probability and frequency of occurrence of events, are characteristic of FT analysis [9, 24, 30, 32]. The reason is that it is feasible and useful to estimate e.g. the mean time to failure (MTTF) of machine components: this allows engineers to compute safe, cost-optimal policies for inspection, maintenance, and replacement of company assets [29].

In contrast, the probability of basic attacks in ATs are very hard to know [11]. Unknown vulnerabilities may increase it, also its frequency, and the conditional probability tables are usually not-knowable. Therefore, it is typical to query the max (rather than total) attack probability [34]. This is also cost-driven: rather than try everything, an attacker may only choose the most promising attack.

[†]There are other equivalent ways to define static AT/FT semantics, e.g. bundles [25].

The time for an attack is also described differently than for a failure: whereas failures typically have MTTF or rate values, basic attacks steps can be given [min, max] intervals, and further differentiate activation from execution time [23].

Finally, quantitative analyses in ATs can be richer than in FTs, exploring attribute domains beyond time and probability. These include the cost to carry out certain attacks, the skill or psychological profile required to do it, the max damaged caused, and Pareto analyses thereof [2, 11, 23].

Propagation of values through logical gates. When the AT or FT is a proper tree, quantitative queries can be computed bottom-up directly on its syntactic structure. For this, the values of the basic elements are propagated upwards in the tree [25]. For instance if basic elements a and b cost resp. €3 and €7, then the cost of $\mathrm{OR}(a, b)$ is the min, €3, and the cost of $\mathrm{AND}(a, b)$ is the sum, €10.

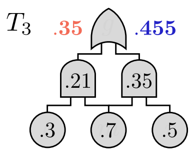


Figure 2: Probability computation: AT (left, red) vs. FT (right, blue)

However, here too we find a remarkable difference between static FTs and ATs, that is overlooked by many reviews in spite of its apparent impact in quantitative analyses. In FTs, the "probability of failure" asks for *total probability*, so the (probability) value of an OR gate is the sum of the values of its children, minus the value of their intersection. Instead and as indicated above, attacks are characterised by their *max probability*, so the value of an OR gate in an AT is the max value among its children. This is illustrated in Fig. 2: the values in the basic elements are given; the probability of an AND gate is the product of its children; but if T_3 is an AT, its (max) attack probability is .35; and if it is an FT, its (total) failure probability is .455.

This different propagation of values also affects conjunctive gates (AND), most notably with time attributes. The basic elements in static FTs refer to failures in components, which are under simultaneous use and therefore whose degradation is concurrent. Thus the MTTF of an AND gate is the max across the MTTF of its children. ATs, in contrast, have more ways to describe a conjunction of activities. In particular they could require time-exclusion: consider one attacker who must perform multiple actions, e.g. deactivating an alarm and silencing the dog. Here, the time to execute all attacks is not the max, but the sum of the times [1]. ATs can indicate this with a new gate: *sequential-AND* (SAND).

Generally speaking, static ATs and FTs have begotten independent extensions that introduce new gates. For instance, SEQ enforcers in FTs can be seen as analogous to SAND gates in ATs. In general, however, these extensions further differentiate the AT and FT formalisms, as we discuss in Sec. 3.2.

3.2 Extensions of the formalisms

So far we considered (only) static FTs and ATs, pointing out their similarities and differences. In this section we revise several extensions that grow beyond them. We first overview some prominent formalisms in Table 1, and then refine the comparison by defining and making use of the concept of dimension.

Form.	Extensions			Main features
FTs	**DFT**	[8]	Dynamic FTs	FTs + PAND + SPARE + FDEP
	RFT	[4, 7]	Repairable FTs	FTs + repair boxes
	E-DFT	[10]	Extended DFTs	DFTs + gen. SPAREs + triggers
	SE-FT	[27]	State/Event FTs	FTs + Petri nets in leaves
ATs	**SAND-AT**	[16]	SAND attack trees	ATs with sequential AND
	ADTree	[20]	Attack–defense trees	ATs + defenses
FTs & ATs	**BDMP**	[5]	Boolean Markov proc.	FTs + ATs + triggers + repairs
	CFT	[31]	Component FTs	ATs + FTs with modular structure
	AFT	[22]	Attack-Fault Trees	SAND-ATs + DFTs
	FT-AT	[12]	FTs integrated to ATs	FTs whose BEs are refined as ATs

Table 1: Overview of extensions to the FT and AT formalisms

Safety extensions of FTs. The first formalisms in Table 1 are important extensions of FTs: *DFTs* are static FTs plus PAND gates (that fail if all children fail in left-to-right order), SPARE gates (for spare parts), and FDEPs (that model common-cause failures); *RFTs* are static FTs with repair boxes, that can repair failed BEs; and *E-DFTs* are generalised DFTs with triggers, which allow arbitrary subtrees as spares, and whose FDEPs can trigger gates as well as BEs.

FTs + security. While DFTs, RFTs, and E-DFTs, improve safety modelling in fault trees, other extensions can cover security aspects. For instance, *SE-FTs* have Petri nets as basic elements. These are more versatile than the usual two-state BEs, allowing state changes that can model safety *and security* hazards.

Security extension of ATs. There also exist extensions to improve security modelling of attack trees: *SAND-ATs* add dynamic behaviour to ATs, by forcing attacks to occur in a specific order via SAND gates[‡]; and *ADTrees* model protections against attacks via special defense nodes.

Combinations of FTs and ATs. All those formalisms extend either FTs or ATs, but there also exist formalisms that combine them. *BDMPs* can have ATs as subtrees of FTs and vice versa, and allow propagations of failures/attacks (and repairs) via triggers among gates. *CFTs* add modular FTs to ATs, to foster large-system analysis via decoupled studies of smaller components. In contrast, *AFTs* trade scalability for versatility, by merging DFTs (with all its dynamic gates) with ATs plus SAND gates. Finally, *FT-ATs* refine the BEs in fault trees via attack trees, modelling attackers that try to force a system failure.

Note that, interestingly and to the best of our knowledge, no formalism that combines FTs with ATs includes defenses. More importantly, we find independent extensions that overlap in some modelling goals, e.g. RFTs and the repairs of BDMPs. We compare these (partial) overlaps via dimensions.

Dimensions. An extension augments the modelling power of FTs/ATs. Some extensions reach to each other, e.g. AFTs and BDMPs are in both domain ontologies (safety and security). But other extensions are parallel: compare DFTs to

[‡]SANDs in ATs *force a sequence of events*, similarly to SEQ enforcers in certain flavours of FTs; this differs from PAND gates in DFTs, which *observe the order of events.*

SAND-ATs, both of which make the order of events relevant but without crossing the safety/security line. We thus identify different ways to classify the space of formalisms, where a *dimension* d defines (not necessarily exclusive) classes that are orthogonal to those defined by another dimension d'. For example, the domain ontology can be seen as a *domain* dimension: it defines the classes `safety` and `security` s.t. the formalisms {FT,DFT} are in `safety`, {AT,SAND-AT} are in `security`, and {AFT,BDMP} are in both. Further dimensions to classify formalisms include *dynamics*—the order of events matters or not—and *complement*—there is a single type of event (e.g. attacks), or complementary types (also defenses). Fig. 3 shows a scheme of this 3D classification.

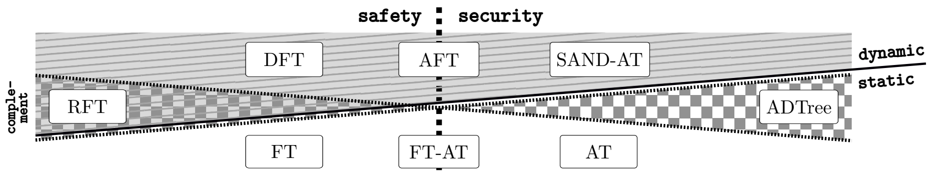


Figure 3: Dimensional split of formalisms: *domain*, *dynamics*, and *complement*.

Such concept of dimension resembles that of an *ontology* in information science [13]. For us, different dimensions yield orthogonal classifications of the same set of individuals. These individuals are the formalisms within scope: we use Table 1 as the scope, but Def. 2 generalises to any FT/AT extension.

Definition 2 (Dimension). *A* dimension *is an ontology with two or more non-empty classes, whose individuals are the formalisms from Table 1. A* dimensional base $\mathbb{D} = \{d_i\}_{i=1}^{n}$ *is a finite set of orthogonal* dimensions*.*

So far we have compared formalisms exclusively from the perspective of the *domain* dimension: we now turn our attention to *dynamics* and *complement*.

Note, however, that these three dimensions—that Table 2 defines in our full scope—are not exhaustive. We identify at least an extra *complexity* dimension, sensitive to the number of states of the basic elements. In terms of *complexity*, FTs and ATs are `simple` (binary states), while SE-FTs and Fault Maintenance Trees [29] are `complex` (its basic elements are resp. Petri nets and Erlang chains).

Dynamics. This dimension classifies formalisms based on whether their semantics caters for order. The broadest possible classes are `static` and `dynamic`. The success of the top element in a `static` formalism does not depend on the order in which the basic elements occur. This includes FTs, ATs, CFTs, AT-FTs, and ADTrees. Other formalisms in Table 1 are `dynamic`: they either enforce an order, e.g. SAND gates and SEQ enforcers; or the propagation of success in some gates depends on it, e.g. PAND gates in DFTs. Besides a richer semantics (that affects qualitative analyses), `dynamic` formalisms have more complex quantitative analyses. In a static AT, the attack time of a conjunctive gate is the max time among its children. Instead, in a SAND-AT, it is the max *or* the sum of the times, depending on whether the gate is a "parallel" AND *or* a sequential-AND.

Dimension		FT	DFT	RFT	E-DFT	SE-FT	BDMP	CFT	AFT	FT-AT	SAND-AT	ADTree	AT
dom.	`safety`	✓	✓	✓	✓	✓	✓	✓	✓	✓			
	`security`					✓	✓	✓	✓	✓	✓	✓	✓
dyn.	`static`	✓						✓		✓		✓	✓
	`dynamic`		✓	✓	✓	✓	✓		✓		✓		
cmp.	`single`	✓	✓		✓	✓		✓	✓	✓	✓		✓
	`dual`			✓			✓					✓	

Table 2: Dimensional split of formalisms in Table 1

Complement. This dimension has two classes: `dual` formalisms have two complementary type of events; `single` formalisms have one. By *event* we mean a change of state, whose multiplicity can have syntactic support via a type system, or it can reside entirely at semantic level. An example of the latter are repairs in RFTs: their (single-typed) basic elements can transit in both directions between their active and failed states. An example of dual events via types are attack- vs. defense-nodes in ADTrees: given an attack, if the counter-defense occurs, then the state of the corresponding gate changes first to "attacked" and then to "not attacked." This differs from the absence of an attack for quantitative queries, e.g. to compute attack cost. In Table 1, the only formalisms in the `dual` class of this *complement* dimension are ADTree, RFT, and BDMP. All the rest are `single`: only one change of state can happen, namely a failure (resp. an attack) that involves a transition from an active to a failed (resp. attacked) state.

Finally, we note that comparing the classification of different dimensions helps to spot research gaps. For instance, from the five formalisms in both classes of the *domain* dimension, only BDMPs are `dual` as per *complement*. Since that comes from repairs of failed basic elements, we know that no formalism in Table 1 that combines `safety` and `security` includes defenses, as pointed out earlier.

4 Conclusions and future work

We have compared FTs against ATs, showing how they model system vulnerabilities in the same mathematical *static way*. However, their different domain ontologies—safety for FTs, security for ATs—gives rise to different quantitative analyses. This shows in the algebra used to propagate values through gates, e.g. to compute the probability of a failure vs. that of an attack. Moreover, new gates have been added to FTs and ATs, extending these formalisms in directions that sometimes cross each other. We introduced the concept of dimension to classify these extensions, thus generalising the safety/security dichotomy.

These studies can be deepened by finding new dimensions to compare formalisms. Our dimensional split offers a high-level view that helps to spot research gaps. In particular, we found no formalism that merges ATs and FTs, that also includes defenses against attacks. Neither have we found formalisms with clearly-differentiated AT/FT submodules, such as FT-ATs, that also offer dynamic gates and repairs, such as BDMPs. The industrial relevance of model visualisation, plus the need for versatile modelling, makes this gap a promising line of research.

References

1. Arnold, F., Hermanns, H., Pulungan, R., Stoelinga, M.: Time-dependent analysis of attacks. In: POST. LNCS, vol. 8414, pp. 285–305. Springer Berlin Heidelberg (2014). https://doi.org/10.1007/978-3-642-54792-8_16
2. Aslanyan, Z., Nielson, F.: Pareto efficient solutions of attack-defence trees. In: POST. LNCS, vol. 9036, pp. 95–114. Springer Berlin Heidelberg (2015). https://doi.org/10.1007/978-3-662-46666-7_6
3. Barlow, R.E., Proschan, F.: Statistical theory of reliability and life testing: probability models. Intl. series in decision processes, Holt, Rinehart and Winston (1975)
4. Bobbio, A., Codetta-Raiteri, D.: Parametric fault trees with dynamic gates and repair boxes. In: RAMS. pp. 459–465. IEEE (2004). https://doi.org/10.1109/RAMS.2004.1285491
5. Bouissou, M.: BDMP (Boolean logic Driven Markov Processes) as an alternative to Event Trees. ESREL 2008 (2008)
6. Chiacchio, F., D'Urso, D., Compagno, L., Pennisi, M., Pappalardo, F., Manno, G.: SHyFTA, a stochastic hybrid fault tree automaton for the modelling and simulation of dynamic reliability problems. Expert Systems with Applications **47**, 42–57 (2016). https://doi.org/10.1016/j.eswa.2015.10.046
7. Codetta-Raiteri, D., Iacono, M., Franceschinis, G., Vittorini, V.: Repairable fault tree for the automatic evaluation of repair policies. In: DSN. pp. 659–668. IEEE Computer Society (2004). https://doi.org/10.1109/DSN.2004.1311936
8. Dugan, J., Bavuso, S., Boyd, M.: Fault trees and sequence dependencies. In: ARMS. pp. 286–293. IEEE (1990). https://doi.org/10.1109/ARMS.1990.67971
9. Ericson, C.A.: Fault tree analysis – A history. In: 17^{th} International System Safety Conference. pp. 1–9 (1999)
10. F., A., A., B., der Berg F., V., M., G.D.S.: DFTCalc: A Tool for Efficient Fault Tree Analysis. Computer Safety, Reliability, and Security. SAFECOMP 2013 **8153**(1), 55–87 (06 2013)
11. Fila, B., Wideł, W.: Attack–defense trees for abusing optical power meters: A case study and the osead tool experience report. In: GraMSec. LNCS, vol. 10510, pp. 95–125. Springer International Publishing (2019). https://doi.org/10.1007%2F978-3-319-66845-1_22
12. Fovino, I.N., Masera, M., De Cian, A.: Integrating cyber attacks within fault trees. Reliability Engineering & System Safety **94**(9), 1394–1402 (2009). https://doi.org/10.1016/j.ress.2009.02.020
13. Guarino, N.: Formal ontology, conceptual analysis and knowledge representation. International Journal of Human-Computer Studies **43**(5), 625–640 (1995). https://doi.org/https://doi.org/10.1006/ijhc.1995.1066
14. Isograph: AttackTree, https://www.isograph.com/software/attacktree/
15. Isograph: FaultTree+, https://www.isograph.com/software/reliability-workbench/fault-tree-analysis-software/fault-tree-analysis/
16. Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R.: Attack trees with sequential conjunction. In: Federrath, H., Gollmann, D. (eds.) ICT Systems Security and Privacy Protection. pp. 339–353. Springer International Publishing, Cham (2015)
17. Jürgenson, A., Willemson, J.: Computing exact outcomes of multi-parameter attack trees. In: OTM. LNCS, vol. 5332, pp. 1036–1051. Springer Berlin Heidelberg (2008). https://doi.org/10.1007/978-3-540-88873-4_8

18. Kabir, S.: An overview of fault tree analysis and its application in model based dependability analysis. Expert Systems with Applications **77**, 114–135 (2017). https://doi.org/10.1016/j.eswa.2017.01.058
19. Kaiser, B., Gramlich, C., Förster, M.: State/event fault trees—A safety analysis model for software-controlled systems. Reliability Engineering & System Safety **92**(11), 1521–1537 (2007). https://doi.org/10.1016/j.ress.2006.10.010
20. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: FAST. LNCS, vol. 6561, pp. 80–95. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-19751-2_6
21. Kordy, B., Piètre-Cambacédès, L., Schweitzer, P.: DAG-based attack and defense modeling: Don't miss the forest for the attack trees. Computer Science Review **13–14**, 1–38 (2014). https://doi.org/10.1016/j.cosrev.2014.07.001
22. Kumar, R., Stoelinga, M.: Quantitative security and safety analysis with Attack-Fault Trees. In: 18th International Symposium on HASE. pp. 25–32 (2017)
23. Kumar, R., Ruijters, E., Stoelinga, M.: Quantitative attack tree analysis via priced timed automata. In: FORMATS. LNCS, vol. 9268, pp. 156–171. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-22975-1_11
24. Lee, W., Grosh, D., Tillman, F., Lie, C.: Fault tree analysis, methods, and applications — A review. IEEE Transactions on Reliability **R-34**(3), 194–203 (1985). https://doi.org/10.1109/TR.1985.5222114
25. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: ICISC. LNCS, vol. 3935, pp. 186–198. Springer Berlin Heidelberg (2006). https://doi.org/10.1007/11734727_17
26. Rauzy, A.: New algorithms for fault trees analysis. Reliability Engineering & System Safety **40**(3), 203–211 (1993). https://doi.org/10.1016/0951-8320(93)90060-C
27. Roth, M., Liggesmeyer, P.: Modeling and Analysis of Safety-Critical Cyber Physical Systems using State/Event Fault Trees. In: SAFECOMP (2013)
28. Roudier, Y., Apvrille, L.: SysML-Sec: A model driven approach for designing safe and secure systems. In: MODELSWARD. pp. 655–664. IEEE (2015)
29. Ruijters, E., Guck, D., Drolenga, P., Peters, M., Stoelinga, M.: Maintenance analysis and optimization via statistical model checking. In: QEST. LNCS, vol. 9826, pp. 331–347. Springer (2016). https://doi.org/10.1007/978-3-319-43425-4_22
30. Ruijters, E., Stoelinga, M.: Fault Tree Analysis: A survey of the state-of-the-art in modeling, analysis and tools. Computer Science Review **15–16**, 29–62 (2015). https://doi.org/10.1016/j.cosrev.2015.03.001
31. Steiner, M., Liggesmeyer, P.: Combination of safety and security analysis - finding security problems that threaten the safety of a system (2016)
32. Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault tree handbook with aerospace applications. NASA Office of Safety and Mission Assurance (2002), version 1.1
33. Watson, H.: Launch control safety study. Tech. Rep. Section VII, Vol. 1, Bell Labs (1961)
34. Wideł, W., Audinot, M., Fila, B., Pinchinat, S.: Beyond 2014: Formal methods for attack tree–based security modeling. ACM Comput. Surv. **52**(4) (2019). https://doi.org/10.1145/3331524