Compositionality of Component Fault Trees*

Simon Greiner, Peter Munk, and Arne Nordmann

Robert Bosch GmbH, Corporate Sector Research, Renningen, Germany
{firstname.lastname}@bosch.com

Abstract In order to deal with the rising complexity of safety-critical systems, model-based systems engineering (MBSE) approaches are becoming popular due to their promise to improve consistency between different views of the system model. Component Fault Trees (CFTs) are one particular technique to integrate the well-known Fault Tree Analysis (FTA) with a model of the system. CFTs decompose the specification of fault propagation on component level, which results in smaller, easier to manage models and leads to a safety analysis view that is consistent with the system model. However, although CFTs gain more and more popularity, their semantics is not well defined and the compositionality of CFTs is not formally proven to the best of our knowledge.
In this paper, we provide a formal basis for CFTs, formalize semantics of CFTs and formally prove compositionality of CFTs by mapping them to information flow semantics, which is well-researched in the security analysis domain. Our results allow insights in the compositionality of CFTs, showing a high potential for validation techniques of CFTs and discuss these consequences in detail. We claim that this proof is crucial for the use of CFTs in assurance cases for safety-critical systems and one fundamental approach to integrate safety and security engineering.

1 Introduction

With systems becoming increasingly complex, analyzing and assuring their dependability becomes more and more challenging, e. g., in the domain of highly-automated driving [2]. Since safety is a system property, component tests are not sufficient. Hence, analysis has to be done on system level, requiring potentially large safety artifacts to be reviewed. With growing complexity, the number of test cases and size of analysis results to be reviewed grows exponentially.

The challenge of the growing size of a particular safety artifact, the fault tree, led to the proposal of Component Fault Trees (CFT) by Kaiser et al. [17,18]. CFTs decompose the fault tree of a system and link its parts to system elements. This allows specification and review of fault propagation on component level and analysis on system level after automatically generating the system's fault tree based on the CFTs and the system architecture. This automation speeds up the

* This work was partially funded within the project SecForCARs by the German Federal Ministry for Education and Research with the funding ID 16KIS0792. The responsibility for the content remains with the authors.

impact analysis of a system's safety properties after changes. However, to the best of the authors' knowledge, the composability property of CFTs has not been formally proven so far.

We argue that a proof of the composability of CFTs is key to allow their usage in assurance cases of safety-critical systems. This paper presents such a formal proof, resulting in two main contributions: First, we provide formal semantics for CFTs and a formal proof that the correctness of CFTs is compositional. This is shown by mapping CFTs to the formalization of non-interference [8], a well-known property from the security engineering domain. We discuss consequences of our formalization and compositionality of CFTs for event types, component reuse, and validation. Second, we show that this mapping is one fundamental approach to integrate safety and security engineering.

2 Components and Component Fault Trees

CFTs are a compositional way to describe the propagation of faults through a system in Model-based Systems Engineering (MBSE). Depending on the point of view of the user of a CFT, it either represents the description of the actual behavior of the component in case of a fault, or it represents the specification of the fault behavior the component is supposed to implement. While both use cases are valid, the intended meaning of when a component is consistent with a CFT is the same. In the remainder, we consider CFTs to be the description of the actual fault propagation of a component.

In this section, we formally define the semantics of CFTs. In Sect. 2.1 we introduce the formal computational model of components. In Sect. 2.2 we formally define CFTs and what it means for a CFT to correctly describe the fault propagation of a component.

2.1 Components

In the remainder of this work we take the formalization of components from Greiner and Grahl [8] and reuse their notation for better comparability of further results in this paper.

A component has an internal state, input ports, and output ports. A component can receive messages via input ports and send messages via its output ports. Received messages can trigger the component to change its internal state. Formally, we consider components as Input-Output Labeled Transition Systems (see [26] for a formal definition of IOLTS). A port has a name and a signature, i. e. names and types of variables that can be communicated via the port. For a message m communicated via an input port with name p with value v, we write $m = p?v$, for a message n communicated via an output port with name q with value w, we write $n = q!w$. We refer to the set of all messages communicated via an input port as inputs, and the set of all messages communicated via an output port as outputs. If it does not matter whether a message is an input or an output, we write $m = p.v$ or $n = q.w$ respectively. We write $c \xrightarrow{m} c'$ for a

component c communicating message m and transitioning to a component c'. You can consider c' to have a changed internal state. If c' is irrelevant, we write $c \xrightarrow{m}$, if there exists some c' such that $c \xrightarrow{m} c'$.

The behavioral definition of a component limits the sequence of messages a component can communicate. We refer to a sequence of messages as a trace. We use $\frown$ as the concatenation operator for traces and $\varnothing$ to refer to the empty trace. The length of a trace is defined as the amount of messages in a trace. We write $c \xrightarrow{t} c'$ if a component c transitions to component c' while communicating the trace t. A component c can communicate a trace $t \frown m$ while transitioning to component c', if there exists a component c'' such that $c \xrightarrow{t} c''$ and $c'' \xrightarrow{m} c'$. We again write $c \xrightarrow{t}$, if there exists some c' such that $c \xrightarrow{t} c'$. We refer to all possible traces as $\mathcal{T}$. Finally, we explicitly define environments in which components can run. Environments model the entities providing inputs to a component after observing the behavior of the component.

Definition 1 (Environment). *An environment ω is a function $\mathcal{T} \mapsto \mathcal{P}(\mathcal{I})$, where $\mathcal{P}(\mathcal{I})$ is the powerset of all inputs.*

Environments limit the traces components can communicate while running in environments to those traces, where the environment provides necessary inputs.

Definition 2 (Communication under Environment). *A component c can communicate a trace t under an environment ω, written $\omega \models c \xrightarrow{t}$, iff $c \xrightarrow{t}$ and for all $t_1, t_2, p?v$ with $t = t_1 \frown p?v \frown t_2$, it holds that $p?v \in \omega(t_1)$.*

2.2 Component Fault Trees (CFTs)

Following the definition of Kaiser et al. [18,17], we consider a CFT as a visual description for a given component that tells which output events are caused by which combinations of input events and basic events.

An event, as used in a CFT, belongs to a port of a component, over which a message is communicated in a erroneous way. The type of an event states in which way the communicated message deviates from a correct one. We introduce the examples *ex* and *val* as types of events later in this section.

Definition 3 (Event). *An event E is a tuple (p, t), where p is port and t is a type. If p is an input (output) port, E is an input (output) event.*

Herein, we distinguish between input events (or input errors), output events (or output errors), and basic events (or internal faults). Basic events are caused by internals of the component, e. g., the breaking of hardware, glitches in a clock, and similar. Hence, basic events are happening independent from the modeled interfaces. For a concise presentation in this paper, we consider basic events in a CFT to be communicated by the environment as a message via an implicit special port into the component. In the remainder, we thus treat basic events analogously to input events.

For each output event, the CFT describes the logical combination of input events and basic events that lead to the output event by means of chained AND

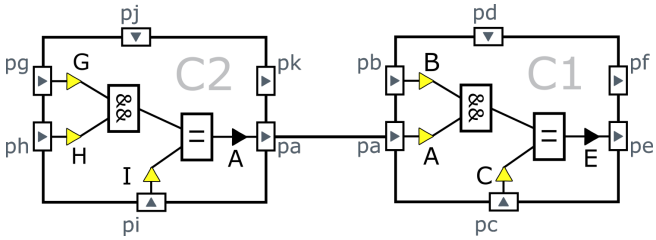

Figure 1. Two simple CFTs, where output events (black triangle) depend on input events (yellow triangles).

and OR gates. Together, input events, basic events, AND gates, and OR gates describe a formula in propositional logic where the events are the literals which only appear non-negated in the formula.

Please note that general fault tree standards such as the IEC 61025 [14] define more complex gates such as a voter gate. We neglect these in this work since they can be transformed into combinations of AND and OR gates. Additionally, the IEC 61025 [14] defines a NOT gate (and a derived XOR gate) together with the hint that it "is advised that this gate be used carefully by an experienced analyst to avoid unwanted results". This hint and the lack of use cases might be the reason why we are unaware of industrial fault trees that use NOT gates, except for mimicking behavior that is inherent to CFTs, namely exchanging subtrees depending on variants used. For this reason, we neglect NOT gates and derived gates as well, assuming literals to occur non-negated in the resulting propositional formula. We can thus formally define a CFT as a tuple of a propositional formula of input (and basic) events and an output event.

Definition 4 (CFT). *A CFT is a tuple (P, E), where P is a propositional logic formula, where each literal in P is an input event, and each input event only appears non-negated in P. E is an output event.*

A CFT describes which combinations of input and basic events lead to which output events. Figure 1 (right) shows the exemplary component $C1$ with input ports p_a, p_b, p_c, and p_d, and the output ports p_e and p_f. The CFT shows the events A, B, and C on the respective ports and the output event E on port p_e.

Example 1. The CFT (P, E) shown in Figure 1 (right) defines $P = (A \sqcap B) \sqcup C$. The semantics of the specification given by the CFT is that E may happen, if either an event A and an event B happen or the event C happens. In other words, the CFT states: if E happened, then previously either the events A and B happened, or the event C happened (or both). In our example, the complete propositional logic formula of the CFT is $E \Rightarrow P$, i. e. $E \Rightarrow ((A \sqcap B) \sqcup C)$.

Please note that a CFT does not state that event E has to happen (equivalence instead of implication), if the other events happen, i. e. a CFT describes a worst-case fault propagation.

We consider two types of events:

1. A timing event (p, ex) describes for a correct message communicated over the port p that in the erroneous case, this message is not communicated (i. e., commission error); or in the correct case a message is not communicated over p, while it is communicated in the erroneous case (omission error).
2. A value event (p, val) describes that in the correct case a message is communicated over p with value v, while in the erroneous case a message on the port is communicated with a value v', different from v.

Example 2. Reconsidering our example in Figure 1, we assume the events (in the sense of Def. 3 describing a deviation from correct behaviour) to be defined as $A = (p_a, ex)$, $B = (p_b, val)$, $C = (p_c, val)$, and $E = (p_e, val)$. Assume the component for which the CFT provides a specification can communicate the following traces:

$$t_c = p_a?1, p_b?2, p_c?3, p_d?4, p_e!5 \qquad t_2 = p_b?2, p_c?3, p_d?4, p_e!6$$
$$t_1 = p_b?2, p_c?3, p_d?4, p_e!5 \qquad t_3 = p_b?3, p_c?3, p_d?4, p_e!6$$

Let t_c above be a correct execution without any input events occurring. For t_1, the CFT correctly describes the behavior of the component: Here, only a timing event happens on port p_a, i. e. the message is not received. The CFT states that the component still sends the correct output message ($E = (p_e, val)$), since no value event occurred at port b (p_b, val) and $P = (A \sqcap B) \sqcup C$.

However, the behavior in t_2 is a counterexample for the correctness of the CFT w.r.t. the component's behavior, because the timing event A happened in the form of not communicating $p_a?1$, while the value event B does not happen. Yet E occurred and the component sends the wrong output value.

The component's behavior in trace t_3 would again be correctly described by the CFT, since additionally to A the value event B happened, and thus the value event E happens.

We will see in the following that it is easier to provide a formal definition of the semantics of a CFT in terms of when an event must not happen. So, instead of expressing when a output event may happen, we rephrase the semantics of a CFT such that it describes when an output event must not happen. For a CFT describing $E \Rightarrow P$, we consider the contrapositive of the formula and gain $\overline{P} \Rightarrow \overline{E}$. The resulting formula states that if P is not satisfied, i.e., if a respective combination of input events does not happen, then the event E must not happen.

Example 3. For our example, the reformulation is as follows:

$$\overline{((A \sqcap B) \sqcup C)} \Rightarrow \overline{E} \;\equiv\; ((\overline{A} \sqcup \overline{B}) \sqcap \overline{C}) \Rightarrow \overline{E} \;\equiv\; ((\overline{A} \sqcap \overline{C}) \sqcup (\overline{B} \sqcap \overline{C})) \Rightarrow \overline{E}$$

In other words: If neither event A nor event C happens, then event E must not happen. Also, if neither event B nor C happens, then event E must not happen.

For every propositional formula P, with literals only appearing non-negated, the formula $\overline{P}$ is a propositional formula with literals only appearing negated. For $\overline{P}$ we can find a disjunctive normal form with clauses $\overline{P_1}, ...\overline{P_n}$, such that $\overline{P} = \overline{P_1} \sqcup ... \sqcup \overline{P_n}$. $\overline{P} \Rightarrow \overline{E}$ then holds, iff for all clauses $\overline{P}_i$ it holds that $\overline{P}_i \Rightarrow \overline{E}$.

Definition 5 (Clause). *A clause $\overline{P_i}$ is a propositional formula, where each literal only appears negated and $\sqcap$ is the only logical operator in the formula.*

A clause $\overline{P}_i$ only considers events on particular ports. Let for a given CFT: $\overline{P}_i = \overline{A_1} \sqcap \ldots \sqcap \overline{A_n}$ and $\overline{F}$ be the negated output event, where $A_i = (q_i, t_i)$ and $F = (q_f, t_f)$. So in Example 3, $\overline{A} \sqcap \overline{C}$ and $\overline{B} \sqcap \overline{C}$ are two separate clauses.

For this specification, messages on ports other than q_i and q_f are irrelevant to the specification. We can now define when messages at most differ from each other according to an event or a clause.

Definition 6 (Message Event-Equivalence). *A message $m = q.v$ is irrelevant w.r.t. an event $A_i = (q_i, t_i)$, if $q \neq q_i$. For a message m that is irrelevant, we write $m \approx_{\overline{A_i}} \square$.*

Two messages $m_1 = q_1.v_1$ and $m_2 = q_2.v_2$ are event-equivalent w.r.t. an event $A_i = (q_i, t_i)$, written $m_1 \approx_{\overline{A_i}} m_2$, if

$$
\begin{aligned}
&m_1 \approx_{\overline{A_i}} \square \text{ and } m_2 \approx_{\overline{A_i}} \square \text{ or}\\
&q_i = q_1 = q_2 \text{ and } t_i = ex \text{ or}\\
&q_i = q_1 = q_2 \text{ and } t_i = val \text{ and } v_1 = v_2
\end{aligned}
$$

Two messages $m_1 = q_1.v_1$ and $m_2 = q_2.v_2$ are event-equivalent w.r.t. a clause $\overline{P_i} = \overline{A_1} \sqcap \ldots \sqcap \overline{A_n}$, written $m_1 \approx_{\overline{P_i}} m_2$, if $m_1 \approx_{\overline{A_i}} m_2$ for all $0 < i \leq n$

Two messages $m_1 = q_1.v_1$ and $m_2 = q_2.v_2$ are event-equivalent w.r.t. a clause $\overline{P_i}$ and an event E, written $m_1 \approx_{\overline{P_i}, \overline{E}} m_2$, if $m_1 \approx_{\overline{P_i}} m_2$ and $m_1 \approx_E m_2$

Example 4. Revisiting Example 2, $\approx_A$ is defined as

$$
\begin{aligned}
&q_1.v_1 \approx_{\overline{A}} \square && \text{if } q_1 \neq q_a \text{ and}\\
&q_1.v_1 \approx_{\overline{A}} q_2.v_2 && \text{if } q_1 = q_2 = q_a \text{ or } q_1.v_1 \approx_{\overline{A}} \square \text{ and } q_2.v_2 \approx_{\overline{A}} \square
\end{aligned}
$$

Analogously $\approx_{\overline{C}}$ is definied as

$$
\begin{aligned}
&q_1.v_1 \approx_{\overline{C}} \square && \text{if } q_1 \neq q_c \text{ and}\\
&q_1.v_1 \approx_{\overline{C}} q_2.v_2 && \text{if } q_1 = q_2 = q_c \text{ and } v_1 = v_2 \text{ or } q_1.v_1 \approx_{\overline{C}} \square \text{ and } q_2.v_2 \approx_{\overline{C}} \square
\end{aligned}
$$

Given event equivalence of messages, we can canonically define equivalence of traces. Two traces are equivalent w.r.t. an event, if both traces at most differ on irrelevant messages and other messages in the traces are equivalent.

Definition 7 (Trace Event-Equivalence). *Two traces t_1, t_2 are event-equivalent w.r.t. an event A_i, written $t_1 \approx_{\overline{A_i}} t_2$, iff*

$$
\begin{aligned}
&t_1 = \varnothing \text{ and } t_2 = \varnothing \text{ or}\\
&t_1 = m_1 \frown t_1' \text{ and } m_1 \approx_{\overline{A_i}} \square \text{ and } t_1' \approx_{\overline{A_i}} t_2 \text{ or}\\
&t_2 = m_2 \frown t_2' \text{ and } m_2 \approx_{\overline{A_i}} \square \text{ and } t_1 \approx_{\overline{A_i}} t_2' \text{ or}\\
&t_1 = m_1 \frown t_1' \text{ and } t_2 = m_2 \frown t_2' \text{ and } m_1 \approx_{\overline{A_i}} m_2 \text{ and } t_1' \approx_{\overline{A_i}} t_2'
\end{aligned}
$$

Event-equivalence of traces w.r.t. a clause is defined analogously.

Example 5. For the clause $\overline{B} \sqcap \overline{C}$ the trace t_c from Example 2 is equivalent to traces t_1 and t_2. However, t_c is not equivalent to t_3, since $p_b?2 \approx_{\overline{B} \sqcap \overline{C}} p_b?3$ does not hold. For the clause $\overline{A} \sqcap \overline{C}$, t_c is not equivalent to any of the traces t_1, t_2, t_3, since $p_a?1 \approx_{\overline{A}} \square$ does not hold, but no message on p_a is communicated in one of the traces t_1, t_2, or t_3.

To provide formal semantics of the specification of a CFT for a component, we compare correct runs of the component with runs where erroneous inputs are provided to the component. Given an environment ω, we define erroneous environments ω_f w.r.t. a clause $\overline{P_i}$ and an output event E such that ω_f can provide input messages that deviate from correct messages according to $\overline{P_i}$. ω_f, however, may provide arbitrary input messages after observing an output from the component, which is not specified by E. In that case, the CFT is not correct w.r.t. the component.

Definition 8 (Erroneous Environment). *ω_f is an erroneous environment for ω w.r.t. a clause $\overline{P_i}$ and an output event E, written $\omega_f \approx_{\overline{P_i}, \overline{E}} \omega$, iff for all $t_1 \approx_{\overline{P_i}, \overline{E}} t_2$ it holds that*

$$\forall p?v \in \omega(t_1) \quad \bullet (p?v \approx_{\overline{P_i}, \overline{E}} \square \;\; or \;\; \exists q?u \in \omega_f(t_2) \bullet p?v \approx_{\overline{P_i}} q?u) \;\; and \tag{1}$$

$$\forall q?u \in \omega_f(t_2) \quad \bullet (q?u \approx_{\overline{P_i}, \overline{E}} \square \;\; or \;\; \exists p?v \in \omega(t_1) \bullet p?v \approx_{\overline{P_i}} q?u) \tag{2}$$

Definition 8 limits how the inputs provided by a correct and an erroneous environment may differ, after observing behaviors of the component which differ in the correct and the erroneous run at most according to the specification provided by P_i and E. Line 1 defines that the erroneous environment must not omit a message, which is provided by the correct environment, unless the correct message is irrelevant. However, the messages may differ according to the clause $\overline{P_i}$ and the output event E. Line 2, states that the erroneous environment must not provide messages, which are not provided by the correct environment, except irrelevant messages. Again, the messages may, however, differ according to the fault specification.

Erroneous environments describe all possible environments, which a component can run in, such that *at most* input events according to a clause are provided. Therefore, an environment should be an erroneous environment to itself (i.e. no input events happen at all). We extend the definition of environments from above.

Definition 9 (Environment (extd.)). *A function ω is an environment w.r.t. a clause $\overline{P_i}$ and an event E, iff it is an environment according to Definition 1, and ω is an erroneous environment to itself according to Definition 8 (i. e., $\omega \approx_{\overline{P_i}, \overline{E}} \omega$).*

We can now define correctness of a CFT w.r.t. a component. A clause $\overline{P}$ and an output event E are correct w.r.t. a component, if for every execution in an erroneous environment, there is also an execution in the respective correct environment, such that the correct and the erroneous execution at most differ on input messages allowed by $\overline{P}$ and output messages allowed by $\overline{E}$.

Definition 10 (Clause correctness w.r.t. a component). *Given the relation $\approx_{\overline{P},\overline{E}}$ for the clause $\overline{P}$ and the output event E. $\overline{P}$ and E are correct w.r.t. a component c, if*

$$\forall \omega_f, \omega_c \forall t_f \bullet \omega_f \approx_{\overline{P},\overline{E}} \omega_c \ \wedge \ \omega_f \models c \xrightarrow{t_f} \implies \exists t_c \bullet \omega_c \models c \xrightarrow{t_c} \wedge \, t_f \approx_{\overline{P},\overline{E}} t_c$$

Finally, we can define for the complete CFT when it is correct w.r.t. a component c, if all clauses defined by the CFT are correct w.r.t c.

Definition 11 (CFT correctness w.r.t. a component). *Given a component c, a CFT (P, E) with $\overline{P} = \overline{P_1} \sqcup ... \sqcup \overline{P_n}$. (P, E) is correct w.r.t. c, iff $\overline{P_i}$ and E are correct w.r.t. c for all $0 < i \leq n$.*

In this section we have formalized CFTs and CFT correctness w.r.t. a component. In the following section we formally discuss CFT composition and show that CFT correctness is compositional.

3 Compositionality of Component Fault Tree Correctness

The core idea of components is to compose them to larger, typically more complex systems. When composing components, we also have to compose their CFTs to a CFT for the composition. In this section, we formally prove that the correctness of CFTs is compositional. The formalization of the semantics of CFTs in Sect. 2 is equal to the formalization of non-interference, a well-known security property that describes information flow through a system. For details on non-interference, we refer to Sect. 5. For proving compositionality of CFT correctness, we re-use results compositionality of non-interference from [8].

In Sect. 3.1 we formally define CFT composition and discuss in Sect. 3.2 compositionality of correctness of composed CFTs w.r.t. composed components. In Sect. 3.3 we show that some restrictions we made in this paper for presentation purposes can be relaxed without violating the core of our results.

3.1 CFT composition

The composition of components in a model is defined by connectors between one output port of one component and an input port of the other component, see Figure 1. These connectors are implicitly directed, with the direction from an output port to an input port. We assume that the composition of components is acyclic. This means that we assume that if an output port of component a is connected to an input port of component b, then there is no output port of component b, which is connected to an input port of component a (and analogously for compositions). For sake of simplicity of the presentation in this paper, we assume that connectors only connect ports with the same name. For two components c and d each providing the ports p_a, we write $c.p_a$ and $d.p_a$ to distinguish them. Also, for simplicity of the presentation, we assume that an output event is at most connected to one input port.

For the CFTs of composed components, we assume, for a simpler presentation, that two ports connected by a connector define the same events, i.e. if a connector connects $c.p_a$ and $d.p_a$, and for $c.p_a$ an event (p_a, t) is defined, then an event (p_a, t) is also defined for $d.p_a$ and vice versa.

Two CFTs can be composed, if the output event of one CFT is the same as an input event of the other CFT. If (P_c, E) is a CFT of component c and (P_d, A) is a CFT of component d, and c and d are composed via a connector on the port which has defined event A, then we can also compose the CFTs. The CFT of the composition *comp* is (P'_c, E), where P'_c is the formula P_c where each occurrence of A is replaced by the formula P_d.

Definition 12 (CFT composition). *Let c and d be components, $p_1 \dots p_n$ ports, which are input ports of c and outputs ports of d, $A^1_1 \dots A^1_{m_1}, \dots A^n_1 \dots A^n_{m_n}$ events with A^i_j being an event on port p_i, i.e. an input event of c and an output event of d. Let further be (P_c, E) be a CFT of c and (P^i_j, A^i_j) the CFTs of d for the output events A^i_j.*

We define the CFT of the composition of c and d for output event E as $(P_{c,d}, E)$, where $P_{c,d}$ is formula P_c with each occurrence of A^i_j is replaced by P^i_j.

Note that by assumption, compositions of components are acyclic. Therefore for all compositions of c and d it is clear in which direction messages are passed. The composition of two CFTs thus itself is a CFT according to Definition 4. As such, we can discuss the correctness of a composed CFT w.r.t. a component.

3.2 Compositionality of CFT correctness

If two CFTs (P_c, E) and (P_d, A) are correct w.r.t. the components c and d respectively, it is not obvious that their composition $(P_{c,d}, E)$ is correct w.r.t. the composition of c and d. We consider here the composition of two components as the interleaving composition of their LTS.

For the following proofs, we assume components to 1) accept inputs only depending on the port, not the communicated value (no discrimination on inputs over the same port), 2) not to produce indeterministic output and 3) not to have indeterministic internal behavior.

Definition 13 (Deterministic components). *A component c is deterministic, if*

$$
\begin{aligned}
&c \xrightarrow{p?v} && \implies c \xrightarrow{p?v'} \text{ for all } v \text{ and } v' \text{ and}\\
&c \xrightarrow{m_1} \text{ and } c \xrightarrow{m_2} \text{ and } m_1 \neq m_2 && \implies m_1 \text{ and } m_2 \text{ are inputs, and}\\
&c \xrightarrow{m} c_1 \text{ and } c \xrightarrow{m} c_2 && \implies c_1 = c_2
\end{aligned}
$$

In the remainder, we assume all components to be deterministic.

We can now show that the composed CFT is also correct w.r.t. the composition of the components.

Theorem 1 (Composed CFT correctness w.r.t. composition). *Given components c and d, CFT (P_c, E) of c and (P_j^i, A_j^i) of d as in Definition 12, such that the CFTs are correct w.r.t. the respective components. Then the composed CFT $(P_{c,d}, E)$ is correct w.r.t. the composition of c and d.*

Proof. The full formal proof for this theorem can be found in an accompanying technical report[1]. □

3.3 Discussion of restrictions

In the previous sections we made several restrictions on components, their ports, allowed event types and others. Several of these restrictions were made in order to allow a compact description of our results in this paper. In the following, we lift several of these restrictions without invalidating our core compositionality result. Please note that all proofs were made such that they also hold in the less restricted case without changes.

In Sect. 3.1 we assume that connectors only connect ports with the same name. This assumption is typically not satisfied in a real model, however, it can easily be achieved with a simple renaming of ports.

We further assume that an output port is at most connected to one input port. Practically, an output port is often connected to several input ports, modeling the property that a message sent via this port is read by several other components. Again, a 1:1 relation between ports can be achieved by duplicating the output port, renaming it, and connecting one input port to one of the duplicated output ports. Similarly, multiple events on a single port are not restricted.

Concerning event types, we only considered timing and value events in Sect. 2.2. Our results hold for more complex type systems, as long as the type system defines an equivalence relation $\approx$ over messages, such that $m_1 \approx m_2$ implies that either m_1 and m_2 are irrelevant, or m_1 and m_2 are messages over the same port. For detailed examples, see [8].

For example we allow event types which state that messages may differ on the last bit, or that messages may differ on the last bit, if the first bit of the value is 1 (e. g., encoding a break signal), or even that a message is irrelevant iff the first bit of the value is 0 (e. g., encoding a log message).

Also in Sect. 3.1 we assume that if two ports are connected, the event types defined on the ports are equal. This assumption is made for presentational purposes. We do require that the event types defined on the ports satisfy a subtype relation. A similar, but informal, subtype definition for CFTs is provided by [21]. If p_i is the input port, and p_o is the output port with events A_i and A_o respectively, it has to be satisfied that for all messages m_1 and m_2 it holds that $m_1 \approx_{\overline{A_i}} m_2$ implies $m_1 \approx_{\overline{A_o}} m_2$. For a detailed discussion of this subtype property, the interested reader is referred to [10] and [9, Sect. 7.3].

[1] Greiner, S., Munk, P., and Nordmann, A.: Compositionality of Component Fault Trees - Definitions and Proofs. (2019). `http://arxiv.org/pdf/1907.09920`

4 Consequences

Apart from the central result of this paper, the compositionality of correctness of CFTs, our work has some fundamental consequences for safety engineering concerning fault propagation as well as security engineering. For one, compositionality allows easier re-using of components and their CFTs in different contexts since analysis results can be re-used and only have to be acquired once. Additionally, our formalization of CFTs connects the well-known security property *non-interference* with the safety method concerning CFTs. Thus the results in the respective domains can be re-used to a certain extent in the other domain.

4.1 Consequences for Safety Engineering

Limits of Compositionality. In this paper, we use equivalence relations over messages as a basis for event specifications. The main reason is that equivalence relations allow for a general and rather simple-to-use compositionality result. Other approaches from non-interference research are not that strict on the specification. In particular [4,7] discuss specifications, where the secrecy (or criticality in our case) of some output information depends on previously communicated inputs, e. g., some information is secret if the user did not previously log in with a password. They show that in general these specifications can also be compositional. However, this result is not general, but heavily depending on internal properties of the program, in particular invariants of the program state and properties connecting those invariants to the external communication history. The proofs of these properties are specific to a concrete specification and the system, complicated, complex, time-consuming, and hence not practicable for real-world programs.

Event Types. Different work on CFTs is concerned with type systems and hierarchies for event types. Typically, this work also provides sub-type relations, often w.r.t. semantic sub types, i.e., a type is a subtype of another, if from common understanding of the expressed fault, it is more specific. Our equivalence relations in combination with their effect on compositionality (see Sect. 3.3) provide a formal condition for subtypes relations.

CFT validation. Fault propagation descriptions are typically validated against the implementation of the system using time-consuming and inherently incomplete fault injection tests. Our formalization shows that CFTs in essence describe an information flow property, thus we can re-use validation methods originally designed for information flow analysis. Most of these methods are focused on software components.

For example, a very common method for analyzing information flow are type systems. See [12] for an overview. Here inputs and outputs of a software component are typed with security types (similar to our event types) and by automatically inferring types of statements and local variables, it can be shown that the

information flow of an implementation is consistent with the specification, i.e., the CFT in our case.

Another technique is taint analysis, e. g., [3], where inputs are tainted to be secret (faulty in our case), and taints are propagated through the program. It can now be checked that public outputs (in our case outputs over critical ports) are not tainted. Taint analysis can be performed statically and dynamically.

Other techniques build on program dependency graphs (e. g., [11]), where through program slicing the flow of information (propagation of faults in our case) is analyzed. Dependency graph based analysis are in particular interesting in terms of scalability. These techniques can be useful for automatically generating information flow specifications (CFTs) from a given program.

Finally, non-interference is a well-defined property, which allows for theorem proving approaches for the verification of information flow (fault propagation). Different approaches have been developed building on different theoretical backgrounds, e. g., [27]. Since non-interference (CFT correctness) is compositional, a combination of different analyses can be relatively easily achieved by using different analyses either for different component or even different partial specifications (Clauses in our case).

4.2 Consequences for Security Analysis

While methods for the analysis of information flow properties is well-researched, it is an open problem how to gain the necessary specifications. For a non-interference specification for security-critical programs, inputs and outputs have to be marked as secret or public. Practically, those specifications do not exist for real-world programs.

Non-interference is often used for modeling confidentiality properties, i.e., properties stating that some secret information must not leak to publicly available outputs. In particular in safety-relevant systems, e. g., automotive systems, a more interesting security property is integrity, i. e. the property that an attacker is not able to influence safety-relevant outputs. Safety norms, e. g., ISO 26262 [15], recommend fault propagation analysis for those outputs in form of FTA and FMEA anyway. Thus, a CFT in essence defines an integrity specification for safety-critical outputs. This specification could generally be re-used in a threat and risk analysis in security engineering, e. g., SAE J3061 [16], to decide whether an attacker can indirectly influence a particular safety-relevant output.

5 Related Work

Extensive overviews of MBSA methods, including CFTs, are given by Aizpurua and Muxika [1], Sharvia et al. [28], and Lisagor et al. [19]. CFTs have been used in different industrial domains, such as railway [13] and automotive [22,24]. The underlying principle of all CFT approaches and implementation is to stitch together the fault tree for a given top event based on the individual CFTs and the components of the system model [17,18].

Thums and Schellhorn [31] present an FTA semantics in Computational Tree Logic (CTL). Later, Thums [30] also introduces an FTA semantics in Interval Temporal Logic (ITL) and compares it with previous formalization, e. g., in Duration Calculus (DC). However, Thums did not consider CFTs.

Bozzano et al. [5] present a trace-based formalization of hierarchical components and their contracts. Extending this formalization with contract-based fault injection, they show how fault trees can automatically be generated. While this is a very powerful approach, it requires components, their contracts, and the refinement of these contracts to be specified. As opposed to this work, we directly prove the composability of CFTs.

Mahmud et al. [20] generate Pandora temporal fault trees (TFT) based on the behavior of components defined by state machines. The approach is to generate a TFT from each state machine and combine these to a TFT of the entire system. However, the authors do not formally prove the compositionality property. To the best of our knowledge, the correctness of this composability has not been formally proven so far. Our formalization of components as Labeled Transition Systems, and formalizing the semantics of CFTs using equivalence relations is an extension of previous work in [8].

Formalizing information-flow properties with an explicit environment was pioneered by Wittbold and Jonson[32], and Rafnsson et al. [26] show that non-interference is also compositional when the presence of messages is secret.

In [8], compositionality is shown for specifications using the more general notion of equivalence relations, and the theory is extended to components which offer their functionality in the form of services. In [10], the authors show that non-interference for service components directly follows from non-interference of services, which allows a combination of analysis methods on a more fine-grained level, hence with increased precision. Bauereis et al. [4] show that compositionality of non-interference for specifications with a dependency on the history, e. g., access to information after logging in, is possible, however complicated and not generalizable.

As mentioned in Sect. 4, the analysis of information flow properties is well-known in the security domain. Several analysis methods from the security domain or the safety domain have been adopted and applied in the respective other domain [25]. One prominent example is the attack tree analysis that is conceptually based on the fault tree analysis [25]. A survey of both techniques is given by Nagaraju [23]. Fovino et al. [6] integrate attack trees and fault trees. While the authors propose a sound mathematical basis for the quantitative security risk assessment, they do not base their analysis on the system model. Steiner and Liggesmeyer [29] propose to extend CFTs with attack trees. The authors propose to leverage the data flow in the system to create new security events besides the safety events in CFTs. For cut sets that contain both, security and safety events, the rating of combined security events and the probability of combined safety events are calculated separately. Steiner and Liggesmeyer do not leverage the information flow that is already modeled in the system model and do not prove the composability of their approach.

6 Conclusion and Future Work

In this paper, we present a formalization of Component Fault Trees (CFTs) by mapping their semantics to the information-flow property non-interference for Input-Output Labeled Transition Systems. We re-use results from security research to formally prove that the correctness of CFTs for components is compositional. By bringing together a well-known safety engineering approach (CFTs) and a well-known security property (non-interference), we enable to check the validity of CFTs against their implementation, leveraging existing validation methods from security engineering such as information flow analysis using type systems, taint analysis, or program dependency graphs and program slicing. Hence, we argue that CFTs provide the integrity specification of safety-critical outputs that is required by threat and risk analysis in security engineering.

As future work, we plan to explore the mutual benefits of other combinations of safety and security engineering methods and processes.

References

1. Aizpurua, J. I., Muxika, E.: Model-based design of dependable systems: Limitations and evolution of analysis and verification approaches. International Journal on Advances in Security **6**(1 & 2), 12–31 (2013)
2. Amarnath, R. et al.: Dependability challenges in the model-driven engineering of automotive systems. In: Proc. of ISSRE Workshops. (2016).
3. Arzt, S. et al.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In: Proc. of PLDI. pp. 259–269. (2014).
4. Bauereiß, T. et al.: A distributed social media platform with formally verified confidentiality guarantees. In: Proc. of Symposium on Security and Privacy. pp. 729–748 (2017).
5. Bozzano, M., Cimatti, A., Mattarei, C., Tonetta, S.: Formal safety assessment via contract-based design. In: Automated Technology for Verification and Analysis. pp. 81–97 (2014).
6. Fovino, I.N., Masera, M., Cian, A.D.: Integrating cyber attacks within fault trees. Reliability Engineering & System Safety **94**(9), 1394 – 1402 (2009).
7. Greiner, S., Birnstill, P., Krempel, E., Beckert, B., Beyerer, J.: Privacy preserving surveillance and the tracking-paradox. In: Proc. of the Future Security - Security Research Conference. pp. 296–302. (2013)
8. Greiner, S., Grahl, D.: Non-interference with what-declassification in component-based systems. In: Proc. of CSF. pp. 253–267 (2016).
9. Greiner, S.: A Framework for Non-Interference in Component-Based Systems. Ph.D. thesis, Karlsruher Institut für Technologie (KIT) (2018).
10. Greiner, S., Mohr, M., Beckert, B.: Modular verification of information flow security in component-based systems. In: Software Engineering and Formal Methods. pp. 300–315 (2017)
11. Hammer, C., Snelting, G.: Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. International Journal of Information Security **8**(6), 399–422 (2009).
12. Hedin, D., Sabelfeld, A.: A perspective on information-flow control. In: Software Safety and Security, NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 33, pp. 319–347 (2012).

13. Höfig, K., Joanni, A., Zeller, M., Montrone, F., Rothfelder, M., Amarnath, R., Munk, P., Nordmann, A.: Model-based reliability and safety: Reducing the complexity of safety analyses using component fault trees. In: Proc. of RAMS (2018).
14. International Electrotechnical Commission (IEC): IEC 61025: Fault tree analysis (FTA) (2006)
15. International Standard Organization (ISO): ISO 26262-4: Road vehicles – functional safety – Part 6: Product development at the system level (2018)
16. Society of Automotive Engineers (SAE): SAE J3061: Cybersecurity Guidebook for Cyber-Physical Vehicle Systems (2016)
17. Kaiser, B., Liggesmeyer, P., Mäckel, O.: A new component concept for fault trees. In: Proc. of SCS. pp. 37–46. (2003),
18. Kaiser, B. et al.: Advances in component fault trees. In: Proc. of ESREL. (2018)
19. Lisagor, O., Kelly, T., Niu, R.: Model-based safety assessment: Review of the discipline and its challenges. In: The Proc. of ICRMS. pp. 625–632. (2011).
20. Mahmud, N., Walker, M., Papadopoulos, Y.: Compositional synthesis of temporal fault trees from state machines. Performance Evaluation Review **39**(4), 79–88. (2012).
21. Möhrle, F., Zeller, M., Höfig, K., Rothfelder, M.: Automating compositional safety analysis using a failure type taxonomy for component fault trees. In: Risk, Reliability and Safety: Innovating Theory and Practice. (2016)
22. Munk, P. et al.: Invited: Semi-automatic safety analysis and optimization. In: Proc. of DAC. (2018)
23. Nagaraju, V., Fiondella, L., Wandji, T.: A survey of fault and attack tree modeling and analysis for cyber risk management. In: Proc of THS. (2017).
24. Nordmann, A. and Munk, P.: Lessons learned from model-based safety assessment with SysML and component fault trees. In: Proc. of MODELS. (2018).
25. Piètre-Cambacédès, L., Bouissou, M.: Cross-fertilization between safety and security engineering. Reliability Engineering & System Safety **110**, 110 – 126 (2013).
26. Rafnsson, W., Hedin, D., Sabelfeld, A.: Securing interactive programs. In: Proc. of CSF. pp. 293–307. (2012)
27. Scheben, C., Greiner, S.: Information Flow Analysis, pp. 453–471. Springer (2016).
28. Sharvia, S., Kabir, S., Walker, M., Papadopoulos, Y.: Model-based dependability analysis: State-of-the-art, challenges, and future outlook. In: Software Quality Assurance, pp. 251 – 278 (2016).
29. Steiner, M., Liggesmeyer, P.: Combination of safety and security analysis - finding security problems that threaten the safety of a system. In: Proc. of DECSSAFECOMP. (2013)
30. Thums, A.: Formale Fehlerbaumanalyse. Ph.D. thesis, University of Augsburg, Germany (2004)
31. Thums, A., Schellhorn, G.: Model checking FTA. In Proc. of FME, pp. 739–757 (2003).
32. Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: Proc. of RISP. pp. 144–161 (1990).