# Formal Safety Assessment via Contract-Based Design[⋆]

M. Bozzano, A. Cimatti, C. Mattarei, S. Tonetta
{bozzano, cimatti, mattarei, tonettas}@fbk.eu

Fondazione Bruno Kessler, Trento, Italy

**Abstract.** Safety Assessment (SA) is an engineering discipline aiming at the analysis of systems under faults. According to industrial practice and standards, SA is based on the construction of complex artifacts such as Fault Trees, which describe how certain faults may cause some top-level events. SA is intended to mirror the hierarchical design of the system focusing on the safety aspects.
In this paper, we propose a formal approach where the nominal specification of a hierarchically decomposed system is automatically extended to encompass faults. The approach is based on a contract-based design paradigm, where components at different levels of abstraction are characterized in terms of the properties that they have to guarantee and the assumptions that must be satisfied by their environment. The framework has several distinguishing features. First, the extension is fully automated, and requires no human intervention, based on the idea that intermediate events are failures to fulfill the contracts. Second, it can be applied stepwise, and provides feedback in the early phases of the design process. Finally, it efficiently produces hierarchically organized fault trees.

## 1 Introduction

Complex systems are often the result of two complementary processes. On the one side, *hierarchical design* refines a set of requirements into increasingly detailed levels, decomposing a system into subsystems, down to basic components. On the other side, the process of *safety assessment* (SA) analyzes the impact of faults. This process is intended to pinpoint the consequences of faults (e.g., a valve failing to operate) on high-level functions (e.g., loss of thrust to engines).

In architectural design, the failure of components is typically not modeled explicitly. Failures are typically artificially introduced in the model for safety assessment. However, the design that is later implemented in real software and hardware components contains only the nominal interfaces and behaviors. It may contain redundancy mechanism or failure monitoring, but not the failure themselves. We call such architectural design the *nominal architecture*. Modeling and analysis of faults is the objective of SA. Unfortunately, there is often a gap between the design of the nominal architecture and SA, which are carried out by different teams, possibly on out-of-sync components. This requires substantial effort, and it is often based on unclear semantics.

⋆ The research leading to these results has received funding from the ARTEMIS JU under grant agreement n$^o$ 295373 and from National funding.

In this paper, we conceived a new formal methodology to support a tight integration between the architectural design and the SA process. Our approach builds on two main ingredients. First, we use Contract-Based Design (CBD) - a hierarchical technique that provides formal support to the architectural decomposition of a system into subsystems and subcomponents. Components at different levels of abstraction are characterized by contracts (assumptions/guarantees). CBD can provide feedback in the early stages of the process, by specifying blocks in abstract terms (e.g., in terms of temporal logic [17]), without the need for a behavioral model (e.g., in terms of finite-state machines). Second, we use the idea of fault injection (a.k.a. model extension), which enables the transformation of a nominal model into one encompassing faults. This is done by introducing additional variables controlling the activation of faults, hence controlling whether the system is behaving according to the nominal or the faulty specification. Within this setting, it is possible to automatically generate Fault Trees (FTs) using model checking techniques. This approach focused in the past on behavioral models [21,13], and is flat, i.e., it generates two-level FTs corresponding to the DNF of their minimal cut sets (MCSs) [11]; as such, it is unable to exploit system hierarchy.

The novel contribution of our approach is the extension of CBD for SA (CBSA): given a nominal contract-based system decomposition, we automatically obtain a decomposition with fault injections. The insight is that the failure mode variables are directly extracted from the structure of the nominal description, in that they model the failure of a component to satisfy its contract. The approach is proved to preserve the correctness of refinement: the extension of a correct refinement of nominal contracts yields an extended model where the refinements are still correct. Once the contracts are extended, it is possible to automatically construct FTs that mimic the structure of the architecture, and formally characterize how lower-level or environmental failures may cause failures at higher levels. This approach has several important features. First, it is fully automated, since SA models are directly obtained from the design models, without further human intervention. Second, it can be applied early in the development process and stepwise along the refinement of the design, providing a tight connection between design and SA. Third, it allows for the generation of artifacts that are fundamental in SA, namely FTs that follow the hierarchical decomposition of the system architecture.

The framework has been implemented extending the OCRA tool [15], which supports CBD. We show experimentally that our approach is able to produce hierarchically organized FTs automatically and efficiently. Furthermore, when applied to behavioral descriptions, the partitioning provided by CBD demonstrates a much better scalability than the monolithic approach provided by previous techniques for Model-Based Safety Assessment (MBSA) [13], which generate flat FTs.

This paper is structured as follows. In Sect. 2, we discuss some related work. In Sect. 3, we present the state of the practice in SA. In Sect. 4, we present some background on formal verification and CBD. In Sect. 5, we discuss contract-based fault injection, and in Sect. 6, we discuss how to generate FTs. In Sect. 7, we present the experimental evaluation. In Sect. 8, we conclude and discuss future work.

## 2 Related work

In recent years, there has been a growing industrial interest in MBSA, see e.g., [13]. These methods are based on a single safety model of a system. Formal verification tools based on model checking have been extended to automate the generation of artifacts such as FTs and FMEA tables [13,12,9,11,10], and used for certification of safety critical systems, see e.g., the Cecilia OCAS platform by Dassault Aviation. However, the scope of such methods is limited to the generation of the MCSs, represented as a two-level FT. This limitation has an impact in terms of scalability, and readability of the FTs. Our approach overcomes the previous limitations – both in terms of scalability (compare Section 7) and significance of the generated FTs (we produce hierarchically organized FTs, as per [4]). Moreover, as a difference with traditional MBSA, we follow a fully top-down development approach, which closely resembles the SA process as described, e.g., in [4], providing feedback in much earlier stages of the design.

An alternative approach for the generation of more structured FTs is based on actors-oriented design [23,20], however these techniques do not account for a stepwise refinement of SA, as outlined in [4]. Specifically, even in presence of minor changes, this approach does not provide the possibility to refine, extend or reuse previous FTA.

Our work is similar in spirit to [5], which presents a methodology based on retrenchment (an extension of classical refinement), to generate hierarchical FTs from systems represented as circuits, exploiting the system dataflow. A major difference is that retrenchment does not focus on top-down development, but rather on the relation between nominal and faulty behaviors. It takes as input the system hierarchy and the behavioral models, hence it does not support the FT generation along the stepwise refinement. Moreover, the framework is theoretical and, although an algorithm for generation of FTs is provided, implementation issues for its realization are not discussed.

In [6], contracts are (manually) generated after a safety and design process. The FT is manually constructed starting from some diagrams describing the system behavior. State machines are extended with faulty behavior to analyze the hazards. Differently from our work, FTA and hazards analysis are used to collect information to specify the contracts. We instead start from the contracts to derive automatically the FT.

In this paper, we based the fault-tree generation on the contract-based refinement. There are other more mature refinement techniques such as the B Method [2], but we are not aware of approaches to FT generation based on these refinements.

Finally, in the context of fault diagnosis, the work described in [25] constructs diagnoses by exploiting the hierarchy of a circuit; the health variable associated with a region of the circuit, called cone, resembles the idea of intermediate event in a FT. However, this work does not focus on architectural design and stepwise refinement.

## 3 Safety Assessment: State of the Practice

Safety assessment is an integral part of the system development of complex systems. As an example, [3] describes the typical development process for civil aircraft as being constituted of different activities, including: a conceptual design, whereby intended aircraft functions are identified; the system architecture design, which is responsible for designing the architecture down to the item level, and allocating aircraft functions to the appropriate items; the allocation of system requirements to items; finally, the implementation of individual items (in software and/or in hardware) and the items/system integration activity. In practice, development may involve multiple iterative cycles, whereby the system architecture and allocated requirements are progressively refined.

In this context, safety assessment has the goal to show that the implemented system meets the identified safety requirements and complies with certification requirements. Safety assessment is strictly intertwined with, and carried out across all phases of development. For example, [4] distinguishes a preliminary aircraft- or system-level safety assessment (PASA/PSSA), which aims at validating the proposed architecture in terms of safety and allocating safety requirements to items, and an aircraft- or system-level safety assessment (ASA/SSA), which systematically evaluates the implemented aircraft and systems in order to show that they meet the safety requirements.

Fault Tree Analysis (FTA) [4,26] is a traditional safety assessment method, which can be applied across different phases. It is as a deductive technique, whereby an undesired state (the so called *top level event*) is specified, and the system is analyzed for the possible chains of *basic events* (e.g., system faults) that may cause the top event to occur. A FT makes use of logical gates to depict the logical interrelationships linking such events, and it can be evaluated quantitatively, e.g., to determine the probability of a safety hazard. FTs are developed starting from the top event; causes which are considered to be elementary faults are developed as basic events, and the remaining causes as *intermediate events*. This rule applies recursively to the intermediate events, which must in turn be traced back to their causes, until the tree is fully developed [26].

*Example 1 (WBS).* The Wheel Braking System (WBS) case study was introduced in [4], and later used to describe a formal specification ([19]) and refinement ([17]) of contracts along a system architecture – it is therefore an ideal case study to evaluate our approach. Fig. 1 shows the WBS architecture. The WBS controls the braking of the main gear wheels for taxiing and landing phases of an aircraft. Braking is commanded either via brake pedals or automatically. The brake is operated by two independent sets of hydraulic pistons, supplied by independent power lines: the "green power supply" (GP), used in normal mode, and the "blue power supply" (BP), used in alternate mode. The alternate system (AWBS) is in stand by and is selected automatically when the normal one (NWBS) fails. An emergency brake system (EWBS) is activated when both NWBS and AWBS fail. In normal mode, the brake is controlled by the Braking System Control Unit (BSCU), implemented with two redundant control units. Each sub-unit (SB1 and SB2) receives an independent pedal-position signal. Monitors detect the failure of the sub-units, producing the "Valid" signals, and of the whole BSCU. The Braking System Annunciation (BSA) monitors the output of the WBS, and it raises a signal in case of every braking systems fail to operate. [4] also describes a PSSA of the WBS, using FTA to analyze the "Unannunciated loss of all wheel braking" top event. The resulting FT (Fig. 2) reflects how the top event depends on the unannounciated loss of the three braking systems and develops the tree downwards, identifying the failures contributing to the unannounciated loss of normal braking. An example of intermediate event is "Normal Brake System does not operate", whereas "Switch failed stuck in intermediate position" is a basic event.

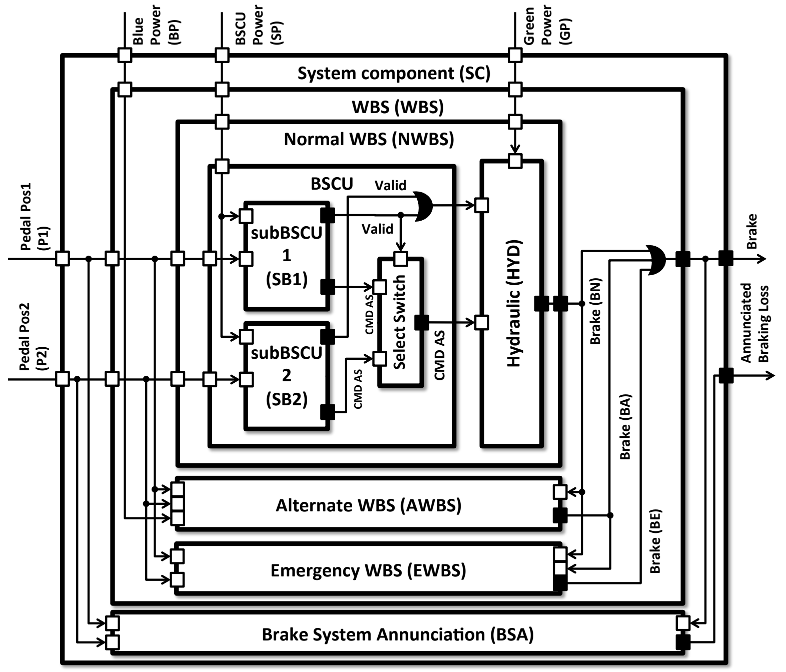

Fig. 1: WBS architecture (the names in parenthesis define the abbreviations)

## 4 Background notions of formal methods

### 4.1 LTL model checking

States and traces are defined over a set $V$ of state variables. A state is an assignment to $V$ of values in a given domain, while a trace is an infinite sequence $\sigma = s_0, s_1, s_2, \ldots$ of states. We denote with $Tr(V)$ the set of all traces over $V$. We define a *language* as a set of traces. We denote with $\sigma[i]$ the $i$-th state of $\sigma$. We use Linear-time Temporal Logic (LTL) [24] to represent sets of traces. We assume that the reader is familiar with LTL. Given an LTL formula $\phi$ and a trace $\sigma$, we write $\sigma \models \phi$ if the trace $\sigma$ satisfies the formula $\phi$. We define the language $\mathcal{L}(\phi)$ as the set of traces $\sigma$ such that $\sigma \models \phi$.

A transition system is a tuple $\langle V, \iota, \tau \rangle$, where $V$ is a set of state variables, $\iota$ is the initial formula over $V$, $\tau$ is the transition formula over $V$ and $V'$ ($V'$ is the set of next versions of the variables in $V$). A path of a transition system $M = \langle V, \iota, \tau \rangle$ is a sequence $s_0, s_1, \ldots$ of assignments to $V$ such that $s_0$ satisfies $\iota$ and for each $k \geq 0$, $\langle s_k, s_{k+1} \rangle$ satisfies $\tau$. We denote with $\mathcal{L}(M)$ the set of paths of $M$. Given a transition system $M$ and an LTL formula $\phi$, the model checking problem is the problem of checking if every trace accepted by $M$ satisfies $\phi$, i.e., if $\mathcal{L}(M) \subseteq \mathcal{L}(\phi)$.

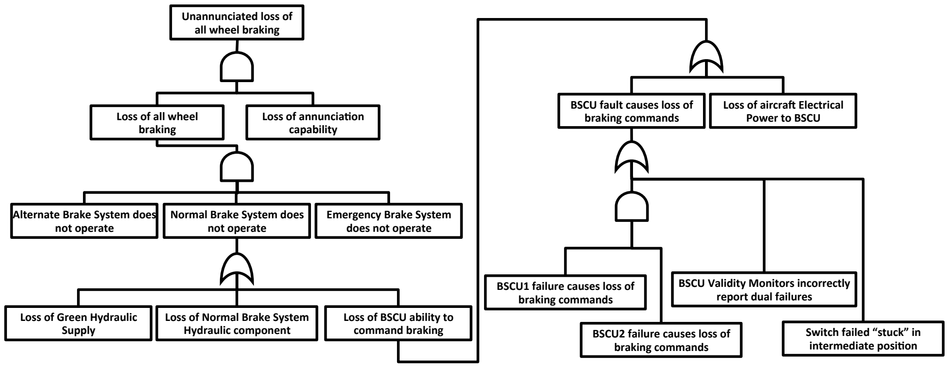
Unannunciated loss of all wheel braking
Loss of all wheel braking
Loss of annunciation capability
Alternate Brake System does not operate
Normal Brake System does not operate
Emergency Brake System does not operate
Loss of Green Hydraulic Supply
Loss of Normal Brake System Hydraulic component
Loss of BSCU ability to command braking
BSCU fault causes loss of braking commands
Loss of aircraft Electrical Power to BSCU
BSCU1 failure causes loss of braking commands
BSCU Validity Monitors incorrectly report dual failures
BSCU2 failure causes loss of braking commands
Switch failed "stuck" in intermediate position

Fig. 2: Fault tree of an unannounciated loss of all wheel braking developed in [3]

## 4.2 Cut-Sets and Fault Tree

As described in Section 3, FTA produces all possible configurations of system faults (called *fault configurations*) that cause the reachability of an unwanted condition (the Top Level Event). More formally, given a set of faults represented as Boolean failure mode variables $\mathcal{F} \subseteq V$, we call *fault configuration* a subset $FC \subseteq \mathcal{F}$. The set $FC$ can be expressed with a formula over $\mathcal{F}$, namely $FC^{\top} = \bigwedge_{f \in FC}(f = \top)$.

A *cut set* represents a fault configuration that may cause the top event. Formally, we generalize the definition in [11] to infinite traces and LTL, as follows. Let $L$ be a language of traces over the variables $V$ and let $TLE$ an LTL formula over $V$. We say that $FC$ is a cut set of $TLE$ in $L$, written $FC \in CS(L, TLE, \mathcal{F})$, iff there exists a trace $\sigma$ in $L$ such that: i) $\sigma \models TLE$; ii) $FC \subseteq \mathcal{F}$ and $\forall f \in FC\ \exists\, i,\ (\sigma[i] \models f = \top)$.

Intuitively, a cut set corresponds to the set of failure mode variables that are active along a trace witnessing the occurrence of $TLE$. *Minimal cut sets* (MCSs), written $MCS(L, TLE, \mathcal{F})$, are those that are minimal in terms of failure mode variables: $MCS(L, TLE, \mathcal{F}) = \{cs \in CS(L, TLE, \mathcal{F}) \mid \forall cs' \in CS(L, TLE, \mathcal{F})\ (cs' \subseteq cs \rightarrow cs' = cs)\}$. Moreover, the set $MCS(L, TLE, \mathcal{F})$ can be expressed with a formula over $\mathcal{F}$ in disjunctive normal form, namely $MCS^{\top}(L, TLE, \mathcal{F}) = \bigvee_{FC \in MCS(L, TLE, \mathcal{F})} FC^{\top}$.

A Fault Tree (FT) [26] can be represented as a set of Boolean formulae over Basic Events (BE) and Intermediate Events (IE). This representation defines a tree where leaves are BE, and nodes are IE. More specifically, the Backus-Naur Form of a Fault Tree $FT$ is as follows: $FT ::= IE \mapsto FT | FT \wedge FT | FT \vee FT | BE$. According to this definition, the first level of the FT represented in Figure 2 can be then expressed as "Unannuciated loss of all wheel braking" (the TLE) $\mapsto$ "Loss of all wheel braking" (an Intermediate Event) $\wedge$ "Loss of annunciation capability" (a Basic Event). The second level extends the IEs of the first one, and in this example it is as: "Loss of all wheel braking" $\mapsto$ "Alternate Brake System does not operate" $\wedge$ "Normal Brake System does not operate" $\wedge$ "Emergency Brake System does not operate". The successor levels recursively define the IEs, while the Basic Events are treated as terminals, as defined by the BNF representation of a FT.

## 4.3 Contract-Based Design

**Components and system architectures** A component interface consists of a set of ports, which are divided into input and output ports[1]. Input ports are those controlled by the environment and fed to the component. The output ports are those controlled by the component and communicated to the environment. Formally, each component $S$ has interface $\langle I_S, O_S \rangle$ of input and output ports. We denote with $V_S$ the set of ports related to the component interface $S$ given by the union of $I_S$ and $O_S$.

In order to formalize decomposition, we need to specify the interconnections between the ports, i.e. how the information is propagated around. Intuitively, the input ports of a component are driven by other ports, possibly combined by means of generalized (e.g., arithmetic) gates. These combinations, in the following referred to as *drivers*, depend on the type of the port. Without loss of generality, we assume that ports are either Boolean- or real-valued. The driver for a Boolean port is a Boolean formula; for a real-valued port it is a real arithmetic expression. Therefore, we define a *decomposition* of a component $S$ as a pair $\rho = \langle Sub, \gamma \rangle$ where $Sub$ is a non-empty set of (sub)components such that $S \notin Sub$, and the connection $\gamma$ is a function that:

- maps each port in $O_S$ into a driver over the ports in $I_S \cup \bigcup_{S' \in Sub} O_{S'}$, and
- for each $U \in Sub$, maps each port in $I_U$ into a driver over the ports in $I_S \cup \bigcup_{S' \in Sub} O_{S'}$.

We extend $\gamma$ to Boolean and temporal formulas so that $\gamma(\phi)$ is the formula obtained by substituting each symbol $s$ in $O_S$ and $I_U$ for all $U \in Sub$ with $\gamma(s)$. Note that, since $\phi$ is a Boolean or temporal formula over the ports of a single component, $\gamma(s)$ does not contain $s$ and therefore $\gamma(\phi)$ is well defined (there is no circularity in the substitution).

A system architecture is a tree of components where for each non-leaf component $S$ a decomposition $\langle Sub_S, \gamma_S \rangle$ is defined such that $Sub_S$ are the children of $S$ in the tree. Let $Sub^*$ be the set of components in the architecture tree. Let $\gamma$ be the union of $\gamma_S$ with $S \in Sub^*$, i.e., $\gamma$ takes an expression over $\bigcup_{S \in Sub^*} V_S$ and substitute $s$ with $\gamma_S$ for every $s \in O_S \cup \bigcup_{S' \in Sub_S} I_{S'}$ (we are assuming that the sets of ports of different components are disjoint). We denote with $\gamma^*$ the iterative application of $\gamma$ until reaching a fixpoint. Thus, $\gamma^*$ takes an expression over $\bigcup_{S \in Sub^*} V_S$ and applies $\gamma$ until the expression contains only input ports of the root and output ports of the leaf components.

Note that, for simplicity, we are considering only synchronous decompositions for which we need only a mapping of symbols. The framework can be extended to the asynchronous case by considering also further constraints to correlate the ports. In the following, we also assume that we have only one instance for each component so that we can identify the instance with its type to simplify the presentation. In practice, we deal with multiple instances by renaming the ports adding the instance name as prefix.

*Example 2.* The WBS architecture, informally introduced in Examples 1, can be formalized with the notion of decomposition defined above. For example, the top-level system component SC has two subcomponents, namely WBS and BSA. Therefore

[1] For simplicity, we ignore here the distinction between data and event ports.

$Sub(SC) = \{WBS, BSA\}$. The mapping $\gamma$ is in most of cases just a renaming. For example, the input port P1 of WBS is driven by the input port P1 of SC. Formally $\gamma(WBS.P1) = SC.P1$ (since we avoided the distinction between component types and instances to simplify the notation, we here use the dot notation to have a unique name for each port). In few cases, the driver is not atomic. For example, the output port Valid of BSCU is driven by the disjunction of the homonyms of SB1 and SB2. Formally, $\gamma(BSCU.Valid) = SB1.Valid \vee SB2.Valid$.

**Trace-Based Components Implementation and Environment** A component $S$ encapsulates a state which is hidden to the environment. It interacts with the environment only through the ports. This interaction is represented by a trace in $Tr(V_S)$.

An input trace is a trace restricted to assignments to the input ports. Similarly, an output trace is a trace restricted to assignments to the output ports. Given an input trace $\sigma^I \in Tr(I_S)$ and an output trace $\sigma^O \in Tr(O_S)$, we denote with $\sigma^I \times \sigma^O$ the trace $\sigma$ such that for all $i$, $\sigma[i](x) = \sigma^I[i](x)$ if $x \in I_S$ and $\sigma[i](x) = \sigma^O[i](x)$ if $x \in O_S$.

For simplicity, we do not distinguish between a language (set of traces) and the behavioral model that generates it. Therefore, both implementations and environments of a component $S$ are seen as subsets of $Tr(V_S)$ (note that we are considering also the output ports for the language of the environment because this can be affected by the component implementation).

A decomposition of $S$ generates a composite implementation given by the composition of the implementation of the subcomponents, as well as a composite environment for each subcomponent given by the environment of $S$ and the implementations of the other subcomponents. In order to define formally these notions, we extend $\gamma$ to states seen as conjunctions of equalities (assignments). Note that, if $s$ is a state, then $\gamma(s)$ represents a set of states. Considering the example of $\gamma$ introduced in Example 2, if $BSCU.Valid = \top$, then $\gamma(BSCU.Valid = \top)$ is equal to $(SB1.Valid \vee SB2.Valid) = \top$. Finally, we extend $\gamma$ to traces seen as sequence of states.

Given a decomposition $\langle Sub, \gamma \rangle$ of $S$ with $Sub = \{S_1, \ldots, S_n\}$ and an implementation $M_j$ for each subcomponent interface $S_j \in Sub$, we define the composite implementation $CI_\gamma(\{M_j\}_{S_j \in Sub})$ of $S$ taking the product of the traces of the subcomponents and projecting on the ports of the component $S$:

$$CI_\gamma(\{M_j\}_{S_j \in Sub}) := \{\sigma^I \times \sigma^O \in Tr(V_S) \mid \exists \sigma_1^O \in Tr(O_{S_1}), \ldots, \sigma_n^O \in Tr(O_{S_n}) \text{ s.t. } \sigma^I \times \sigma_1^O \times \ldots \times \sigma_n^O \in \gamma(M_1) \cap \ldots \cap \gamma(M_n) \cap \gamma(\sigma^O)\}$$

Similarly, given a subcomponent $S_h \in Sub$, an implementation $M_j$ for each subcomponent $S_j \in Sub\backslash$ with $j \neq h$, and an environment $E$ for $S$, we define the composite environment $CE_\gamma(E, \{M_j\}_{S_j \in Sub, j \leq h})$ of $S_h$ taking the product of the traces of $E$ and the other subcomponents and projecting on the ports of $\S_h$:

$$CI_\gamma(\{M_j\}_{S_j \in Sub, j \neq h}) := \{\sigma_h^I \times \sigma_h^O \in Tr(V_{S_h}) \mid \exists \sigma^I \in Tr(I_S), \sigma_1^O \in Tr(O_{S_1}), \ldots \ldots, \sigma_n^O \in Tr(O_{S_n}) \text{ s.t. } \sigma^I \times \sigma_1^O \times \ldots \times \sigma_n^O \in \gamma(M_1) \cap \ldots \cap \gamma(M_n) \cap \gamma(\sigma_h^I)\}$$

**Contracts** A component contract is a pair of properties, called the *assumption*, which must be satisfied by the component environment, and the *guarantee*, which must be satisfied by the component implementation when the assumption holds. We assume as given an assertion language for which every assertion $\mathcal{A}$ has associated a set of variables $V_{\mathcal{A}}$ and a semantics $L(\mathcal{A})$ as a subset of $Tr(V_{\mathcal{A}})$. In practice, we will use LTL to specify such assertions, but the approach can be applied to any linear-time temporal logic.

Given a component $S$, a contract for $S$ is a pair $C = \langle \mathcal{A}, \mathcal{G} \rangle$ of assertions over $V_S$ representing respectively an *assumption* and a *guarantee* for the component. Let $M$ and $E$ be respectively an implementation and an environment of $S$. We say that $M$ is an implementation satisfying $C$ iff $M \cap L(\mathcal{A}) \subseteq L(\mathcal{G})$. We say that $E$ is an environment satisfying $C$ iff $E \subseteq L(\mathcal{A})$. We denote with $\mathcal{M}(C)$ and with $\mathcal{E}(C)$, respectively, the implementations and the environments satisfying the contract $C$.

Two contracts $C$ and $C'$ are *equivalent* (denoted with $C \equiv C'$) iff they have the same implementations and environments, i.e., iff $\mathcal{M}(C) = \mathcal{M}(C')$ and $\mathcal{E}(C) = \mathcal{E}(C')$. A contract $C = \langle \mathcal{A}, \mathcal{G} \rangle$ is in normal form iff the complement $\overline{L(\mathcal{A})}$ is contained in $L(\mathcal{G})$. We denote with $nf(C)$ the assertion $\neg\mathcal{A} \vee \mathcal{G}$. The contract $\langle \mathcal{A}, nf(C) \rangle$ is in normal form and is equivalent to (i.e., has the same implementations and environments of) $C$ [7].

*Example 3 (WBS contract).* We are interested in defining the contract related to the requirement of the WBS that, given the application of the braking pedals, must activate the brakes. This is formalized with the LTL formula $\mathcal{G} = \mathbf{G}((P1 \vee P2) \rightarrow \mathbf{F}(Brake))$. The WBS component requires an environment that provides the same signal on the pedal application and such that power is always supplied to the BSCU and hydraulic pumps. This is formalized in the LTL formula $\mathcal{A} = \mathbf{G}((P1 = P2) \wedge GP \wedge BP \wedge SP)$.

**Contract refinement** Since the decomposition of a component $S$ into subcomponents induces a composite implementation of $S$ and composite environment for the subcomponents, it is necessary to prove that the decomposition is correct with respect to the contracts. In particular, it is necessary to prove that the composite implementation of $S$ satisfies the guarantee of $S$'s contracts and that the composite environment of each subcomponent $U$ satisfies the assumptions of $U$'s contracts. We perform this verification compositionally only reasoning with the contracts of the subcomponent independently from the specific implementation of the subcomponents or the specific environment.

In the following, for simplicity, we assume that each component $S$ has only one contract denoted with $C_S$ and is refined by the contracts of all subcomponents (the approach can be easily extended to the general case [17]). Given a component $S$ and a decomposition $\rho = \langle Sub, \gamma \rangle$, the set of contracts $\mathcal{C} = \bigcup_{S' \in Sub(S)} C_{S'}$ is a refinement of $C_S$, written $\mathcal{C} \leq_\rho C_S$, iff the following conditions hold:

1. given an implementation $M_{S'}$ for each subcomponent $S' \in Sub(S)$ such that $M_{S'}$ satisfies the contract $C_{S'}$, then $CI_\gamma(\{M_{S'}\}_{S \in Sub(S)})$ satisfies $C_S$ (i.e., the correct implementations of the sub-contracts form a correct implementation of $C_S$);
2. for every subcomponent $S''$ of $S$, given an environment $E$ of $S$ satisfying $C_S$ and an implementation $M_{S'}$ for each subcomponent $S' \in Sub(S)$ such that $M_{S'}$ satisfies the contract $C_{S'}$, then $CE_\gamma(E, \{M_{S'}\}_{S' \in Sub(S)})$ satisfies $C_{S''}$ (i.e., the correct implementation of the other subcomponents and a correct environment of $C_S$ form a correct environment of $C_{S''}$).

*Example 4 (WBS contract refinement).* As shown in Fig. 1, the WBS component is decomposed into NWBS, AWBS and EWBS. The contracts of these subcomponents are $C_{nwbs} = \langle \mathbf{G}((P1 = P2) \wedge SP \wedge GP), \mathbf{G}((P1 \vee P2) \rightarrow \mathbf{F}(BN))\rangle$, $C_{awbs} = \langle \mathbf{G}(BP), \mathbf{G}(((P1 \vee P2) \wedge \neg \mathbf{F}(BN)) \rightarrow \mathbf{F}(BA))\rangle$, $C_{ewbs} = \langle \top, \mathbf{G}(((P1 \vee P2) \wedge \neg \mathbf{F}(BN) \wedge \neg \mathbf{F}(BA)) \rightarrow \mathbf{F}(BE))\rangle$. The connection are defined in a straightforward way. It is easy to see that the these contracts correctly refine the contract of the WBS component. We remark that the implementation of the NWBS would be sufficient to ensure the guarantee of the parent component i.e., AWBS and EWBS systems are redundant and play a role only in case of failures.

## 5 Contract-Based Fault Injection

The goal of our approach is to take as input an architecture enriched with a correct contract refinement and automatically generate a hierarchically organized FT. The idea is to introduce, for each component and for each contract, two failure ports: one representing the failure of the component implementation to satisfy the guarantee, the other representing the failure of the component environment to satisfy the assumption. This step is represented by the arrow labeled 1.1 in Fig. 3. The connections among such failures are automatically generated and they are later used to produce the FT, as illustrated by label 1.2 in Fig. 3. The successive refinement of components (i.e., layers 2 and 3 in Fig. 3) allows us to extend the analysis and generate a more detailed FT. These characteristics of the CBSA approach mimic the recommended practices outlined in [4].

### 5.1 Extension of components and contracts

Given a component interface $\langle I_S, O_S\rangle$ of the component $S$, we define the extended interface $\langle I_S^X, O_S^X\rangle$ as the interface in which the inputs has been extended with the new Boolean port $f_S^I$ and the output has been extended with the new Boolean port $f_S^O$. Namely, $\langle I_S^X, O_S^X\rangle$ is defined as $\langle I_S \cup \{f_S^I\}, O_S \cup \{f_S^O\}\rangle$. Intuitively, $f_S^O$ represents the failure of the component implementation to meet its requirements, while $f_S^I$ represents the failure of the component environment to fulfill the component's assumptions.

The "nominal" contract of a component is extended to weaken both assumption and guarantee, in order to take into account the possible failure of the environment and of the implementation. Given the contract $\langle \mathcal{A}_S, \mathcal{G}_S\rangle$ of $S$, we define the extended contract $\langle \mathcal{A}_S^X, \mathcal{G}_S^X\rangle$ as follows $\mathcal{A}_S^X = (\neg f_S^I) \rightarrow \mathcal{A}_S$ and $\mathcal{G}_S^X = (\neg f_S^O) \rightarrow \mathcal{G}_S$.

Note that in this simple contract extension the failure is timeless in the sense that either there are no failures and the nominal contract holds, or nothing is assumed or guaranteed. By convention, the failure ports are evaluated initially and the future values are don't cares. More complex contract extensions will be developed in the future.

### 5.2 Contract-based synthesis of extended system architecture

We now describe how we generate an extended system architecture given a nominal one with a correct contract refinement. In the extended architecture, components' interfaces and contracts are extended as described in the previous section, while we automatically

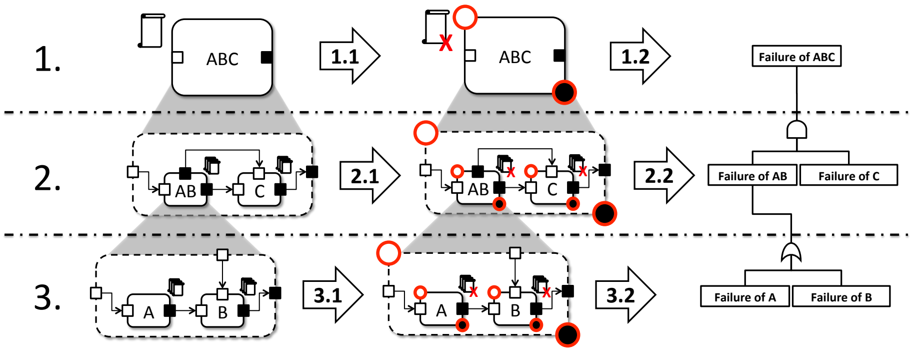

Fig. 3: Contract-based Safety Assessment Process

synthesize the connections among the extended components. The synthesis ensures that the refinement of contracts in the extended architecture is correct by construction.

For each component $S$, we define the extended connection mapping $\gamma^X$ so that $\gamma^X(p) = \gamma(p)$ for all original ports, i.e., for $p \in I_S \cup O_S$, while for the new failure ports $\gamma^X$ is defined as follows:

- $\gamma^X(f^O_S) := MCS^{\top}(\gamma((\bigwedge_{S' \in Sub(S)}(\mathcal{A}^X_{S'} \rightarrow \mathcal{G}^X_{S'})) \wedge \mathcal{A}^X_S), \neg\gamma(\mathcal{G}_S), \{f^I_S\} \cup \{f^O_{S'}\}_{S' \in Sub(S)})$. Intuitively, the driver of the failure of $S$'s guarantee is given by all combinations of the failures of the subcomponents and the environment that are compatible with the violation of the guarantee of $S$.
- for all $U \in Sub(S)$, $\gamma^X(f^I_U) := MCS^{\top}(\gamma(\bigwedge_{S' \in Sub(S) \setminus \{U\}}(\mathcal{A}^X_{S'} \rightarrow \mathcal{G}^X_{S'}) \wedge \mathcal{A}^X_S), \neg\gamma(\mathcal{A}_U), \{f^I_S\} \cup \{f^O_{S'}\}_{S' \in Sub(S) \setminus \{U\}})$. Intuitively, the driver of the failure of $U$'s assumption is given by all combinations of the failures of the other subcomponents and the environment of $S$ that are compatible with the violation of the assumption of $U$.

The resulting extended contract refinement is correct:

**Theorem 1.** *If* $\{C_{S'}\}_{S' \in Sub(S)} \preceq_{\gamma} C_S$, *then* $\{C^X_{S'}\}_{S' \in Sub(S)} \preceq_{\gamma^X} C^X_S$.

*Example 5 (Synthesis of faults dependencies for WBS component).* Given the extended contract $C^X_{wbs}$, the safety analysis will produce the dependencies formulae for each fault port $f^O_{wbs}$, $f^I_{nwbs}$, $f^I_{awbs}$ and $f^I_{ewbs}$. Specifically, the resulting faults dependency for $f^O_{wbs} := (f^O_{awbs} \wedge f^O_{nwbs} \wedge f^O_{ewbs}) \vee (f^I_{wbs} \wedge f^O_{ewbs})$, which means that every assignment of such formula will cause the failure of $f^O_{wbs}$. This result confirms that the braking ability of the WBS is guarantee if at least one of NWBS, AWBS and EWBS is working, but in case of loss of the power sources ($f^I_{wbs}$) the EWBS is necessary in order to guarantee the right behaviour. The following analysis for $f^I_{nwbs}$ and $f^I_{awbs}$ will produce respectively $f^I_{nwbs} := f^I_{wbs}$ and $f^I_{awbs} := f^I_{wbs}$. In fact, the subsystems NWBS and AWBS need for BP, SP, and GP power lines, which functionality is part of the assumption of the WBS. The last step addresses the verification of the proof obligation for $f^I_{ewbs}$ which is unsat, expressing the fact that it has no dependencies to the other fault ports. According to this result, Fig. 1 shows that EWBS is not dependent to any assumptions of the WBS i.e., it does not need any power sources.

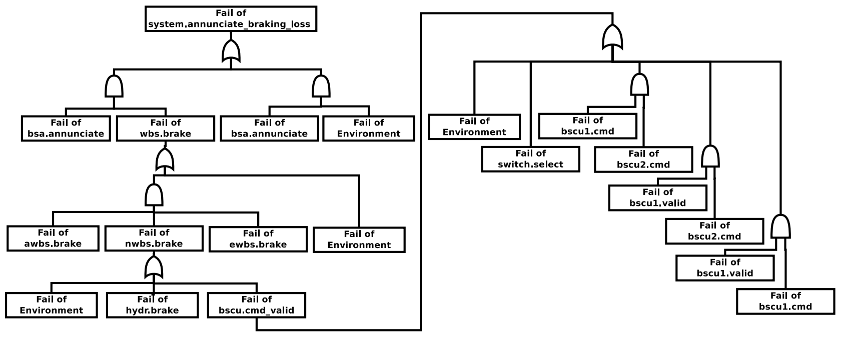

Fig. 4: Fault tree of an unannunciated loss of all wheel braking: automatically generated

## 6 Contract-Based Fault Tree Analysis

### 6.1 Contract-based Fault Tree Generation

Given the extension of the system contract refinement, the FT is automatically generated. The top level event is the failure $f_S^O$ of a non-leaf component $S$. It is labeled with "Fail of $C_S$", where $C_S$ is the contract of $S$. The intermediate events are similarly labeled with the failure of the guarantees of the components that are used in the contract refinement and are not further refined. The failure of the system environment is labeled with "Fail of Environment". The leaves of the tree are basic events, representing the failure of the system's assumption and the failures of the guarantees of contracts that are not further refined. If the architecture is extended further in a step-wise way by decomposing some leaves components, these basic events can become intermediate and be refined further by exploiting the extended contract refinement.

The FT is generated starting from the top level event $f_S^O$ and linking it to the intermediate events present in $\gamma^X(f_S^O)$. Formally, if $f$ is a basic event, then the FT is atomic: $FT(f) := f$; if $f$ is an intermediate event, then $FT(f_S^O) := f_S^O \mapsto \gamma^X(f_S^O)$. Thus, the FT is defined recursively until reaching the basic events. To simplify the tree, we do not label the failure of the assumption of intermediate components. Therefore, if $U$ is not the system component and $f_U^I$ is present in the tree, we replace it with $\gamma^X(f_U^I)$. Note that the same failure may appear in different branches of the FT – this is standard in FTA – hence, in the above top-down procedure we only need to expand one occurrence of the same failure. We also assume that in the relationship among the failures there is no circular dependency. Usually, such dependencies may be broken by introducing time delays [26]. We leave modeling of faults with temporal dynamics and dealing with circular dependencies to future work.

*Example 6 (Automatic generation of WBS FT).* By applying contract refinement to the WBS example, we obtain the FT in Fig. 4. As it can be seen from Table 1, there is nearly a one-to-one mapping with the FT presented in Fig. 2 – the only differences are that: (i) in the contract-based FT the failure of the environment is considered also for the sub-components that depends to it, and this provides a more detailed system failure explanation; (ii) the monitoring function is more detailed in our model.

| Failure of Contract | Description |
|---|---|
| system.annunciate_braking_loss | Unannunciated loss of All Wheel Braking. |
| bsa.annunciate | Loss of Annunciation Capability. |
| wbs.brake | Loss of All Wheel Braking. |
| nwbs.brake | Normal Brake System does not operate. |
| awbs.brake | Alternate Brake System does not operate. |
| ewbs.brake | Emergency Brake System does not operate. |
| hydr.brake | Loss of Normal Brake System Hydraulic Components. |
| bscu.cmd_valid | Loss of BSCU Ability to Command Braking. |
| switch.select | Switch Failure Contributes to Loss of Braking Commands. |
| bscu1.cmd | Loss of BSCU sub system 1. |
| bscu1.valid | Loss of monitoring for BSCU sub system 1. |
| bscu2.cmd | Loss of BSCU sub system 2. |
| bscu2.valid | Loss of monitoring for BSCU sub system 2. |

Table 1: Failure of contracts description

## 6.2 CBSA Cut-Sets semantics

We notice that, in the generated FT, the cut sets local to a single component decomposition are minimal by construction. Here, we consider the cut sets of the whole FT that are obtained by replacing intermediate events with their definition in the FT. We call them flattened cut sets, since they can be represented as a two-level FT. They are defined in terms of the failures of the basic components and of the system environment.

Let $leaves$ be the basic components of the architecture and let $root$ the (root) system component. We denote with $\mathcal{F}$ the set of basic failure ports, i.e., $\mathcal{F} = \{f_l^O\}_{l \in leaves} \cup \{f_{root}^I\}$, and we identify a fault configuration with an assignment to these parameters. A cut set is therefore a fault configuration of a trace violating the top-level guarantee.

Given a failure port $f_S$ (either input or output) of a component $S$ in the architecture, let us define $\gamma^{X*}(f_S)$ as the iterative application of $\gamma^X$ to $f_S$ until reaching a fixpoint, i.e., a Boolean combination of failures in $\mathcal{F}$ only. $\gamma^{X*}(f_S)$ defines the set of flattened cut sets obtained with CBSA. We prove that every cut set (in the standard sense) is also a flattened cut set for CBSA.

**Theorem 2.** *Let* $L^X = \mathcal{L}(\gamma^*(\bigwedge_{l \in leaves}(\mathcal{G}_l^X) \wedge \mathcal{A}_{root}^X))$.
*If* $FC \in CS(L^X, \neg(\mathcal{G}_S), \mathcal{F})$*, then* $FC^\top \models \gamma^{X*}(f_S^O)$.

Here, $L^X$ represents the extension of the system architecture in a MBSA-like fashion, where the guarantees of leaf components and the root assumption are extended locally without explicit constraints among component failures (hence, $\gamma^*$ is used instead of $\gamma^{X*}$). The converse is not true in general. In fact, for the contract refinement to be correct, it is sufficient that the contract of the composite component is weaker than the composition of those of the subcomponents. However, this may create cut sets that are present considering the weaker contract, while are they ruled out by the composition.

## 6.3 Relationship between contracts and generated fault trees

We remark that the FT generated with the proposed approach is clearly sensitive to the contracts and can be used to improve the CBD. For example, in the contract specification of the WBS proposed in [19], each redundant sub-BSCU guarantees that the input pedal application is followed by the braking command or the Validity Monitor set to invalid within a given time bound. Following this approach, the proposed procedure generates a FT in which each sub-BSCU is a single point of failure. In fact, a failure of its contract means that it can keep the Validity Monitor set to true without ever braking. This contrasts with [4]. The FT shown in Fig. 4 is actually obtained with an improved specification, where we separated the functional part of the contract from the monitoring of safety, providing a contract that says that every pedal application is followed by the braking command and another contract demanding that the Validity Monitor is set to invalid if the pedal is applied but the brake is never commanded.

## 7 Implementation and experiments

We implemented our methodology on top of OCRA [15], a tool for architectural design based on CBD. The OCRA language allows the user to specify contracts (written in various temporal logics of different expressiveness, including LTL and HRELTL [16]), and associate them to architectural components. The correctness of refinements is reduced to a set of proof obligations (as per Section 4.3) – temporal satisfiability checks that are carried out by nuXmv [22], the underlying verification platform, which provides reasoning capabilities via BDD-, SAT-, and SMT-based techniques.

We extended OCRA in the following directions. First, we implemented primitives to automatically extend the architectural description by means of symbolic fault injection, extending the ports and the contracts. Second, we implemented the procedure for the synthesis of the interconnections between failure ports among different levels, as per Section 5. Finally, we implemented the procedure to extract FTs from the extended models, as per Section 6. The algorithms are based on pure BDD [18], in addition to a combination of Bounded Model Checking (BMC) [8] and BDD. In particular, the BMC+BDD approach first computes MCSs up to a specific k-depth using BMC, and then a BDD based routine is run to generate the remaining results.

We first evaluated the CBSA approach by modeling (several variants of) the WBS case-study in OCRA[2]. The analysis demonstrated very useful to provide feedback on the structure of the contracts. In fact, as described in Section 6.3, we could improve over the first version of the WBS model described in [19,17]. We then compared our approach with the "flat" MBSA approach implemented in xSAP- a re-implementation of FSAP[12]. xSAP supports FTA for behavioral models (finite state machines written in the SMV language). We refer to the xSAP approach as *monolithic*, since it generates FTs that are "flat"(i.e., presented as DNF of the MCSs). In OCRA, FTs can be generated from behavioral models, by associating each leaf component with an SMV implementation, where the activation of failure modes causes the violation of contracts. For the evaluation, we associated concrete implementations to the leaf WBS components. We first evaluated the tightness of the contract extension. As described in Section 6, CBSA can provide a "pessimistic" interpretation of the system failure, due to the hierarchical partitioning imposed by contract decomposition. Indeed, our results confirm that this is the case for the WBS: if the concrete implementations happen to operate correctly even if the power is not provided, then the monolithic approach provides a tighter set of

[2] The models and the binaries necessary to reproduce the experiments described in this paper can be download at `https://es.fbk.eu/people/mattarei/dist/ATVA14/`.

MCSs. However, if the concrete implementations are such that a loss of power implies a loss of functional behavior, then both techniques result in the same sets of MCSs. We also compared the scalability of the monolithic and the CBSA approach for FTA. We considered a parameterized version of the WBS, by varying the total number of faults ($M$), and the upper bound for the cycles needed to wait until performing an emergency reaction ($N$). The experiments were run on an Intel Xeon E3-1270 at 3.40GHz. We first varied the delay $N$ (with $M = 9$). With $N = 10$, CBSA takes 11m40s (BDD), and 2s (BMC+BDD, with k=20), whereas the monolithic approach takes 14m and 7s, respectively. For $N = 15$, CBSA times do not vary, while the monolithic approach requires more than 50m (BDD), and 15s (BMC+BDD). The stability in performance shown by the CBSA approach is motivated by the fact that the time needed to compute the FT is mainly spent during the contracts evaluation, whereas analyzing the leaves takes always less than 1s. We then fixed $N = 5$ and varied $M$ from 9 to 29. The results are reported in Table 2, where “BD” and “BB” stand for BDD and BMC+BDD (with k=20). CBSA is subject only to a marginal degradation in performance, since the variation is local to the computation of the FTs for the leaves. In contrast, the monolithic method passes from 10m19s to timing out after one hour for $M = 12$ (BDD), and from 3s to 582s (BMC+BDD). This degradation is directly correlated to the increased number of MCSs, that are enumerated by the monolithic approach. As a final remark, notice that the CBSA approach is fully incremental: the only variation required when exploring different implementations is in constructing the FTs resulting from the analysis of each finite state machine with respect to its contracts. This contrasts with the considerable efforts required in the monolithic approach, that needs to be repeated for each different implementation.

| M | 9 | 10 | 11 | 12 | 13 | ... | 29 |
|---|---|---|---|---|---|---|---|
| MCS | 6 | 11 | 22 | 42 | 50 | ... | 3316 |
| CBSA.BD | 701 | 701 | 701 | 702 | 702 | 702 | 703 |
| Mono.BD | 619 | 1106 | 3180 | T.O. | T.O. | T.O. | T.O. |
| CBSA.BB | 1.8 | 1.8 | 1.8 | 1.8 | 1.8 | 1.8 | 1.8 |
| Mono.BB | 3.1 | 3.4 | 4.1 | 4.6 | 4.9 | ... | 582 |

Table 2: Scalability comparison

## 8 Conclusions

In this paper we proposed a new, formal methodology for safety assessment based on CBD and automated fault injection. This approach is able to generate automatically hierarchical FTs mimicking system decomposition, and overcomes two key shortcomings of traditional MBSA [13], namely the lack of structure of the generated FTs, and the poor scalability. Moreover, it provides full support to the informal, manual state of the practice, and it can provide important feedback in the early stages of system design.

As future work, we will investigate methods to pinpoint situations where the hierarchical decomposition leads to over-constraining, and to generate suitable diagnostic information. Second, we will generalize fault injection with the introduction of more fine-grained failure dynamics based on temporal patterns and the use of specific fault models (similar to the contract extension with “exceptional” behavior [14]). We will investigate aspects related to fault propagation [1] and extend the framework to consider richer contract specification languages to enable quantitative evaluation of FTs.

## References

1. S. Abdelwahed, G. Karsai, N. Mahadevan, and S.C. Ofsthun. Practical Implementation of Diagnosis Systems Using Timed Failure Propagation Graph Models. *IEEE T. Instrumentation and Measurement*, 58(2):240–247, 2009.
2. J. R. Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge Univ. Press, 1996.
3. ARP4754A Guidelines for Development of Civil Aircraft and Systems. SAE, December 2010.
4. ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, SAE, December 1996.
5. R. Banach and M. Bozzano. The Mechanical Generation of Fault Trees for Reactive Systems via Retrenchment II: Clocked and Feedback Circuits. *FAC*, 25(4):609–657, 2013.
6. I. Bate, R. Hawkins, and J.A. McDermid. A Contract-based Approach to Designing Safe Systems. In *SCS*, pages 25–36, 2003.
7. A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple Viewpoint Contract-Based Specification and Design. In *FMCO*, pages 200–225, 2007.
8. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.
9. M. Bozzano, A. Cimatti, J.P. Katoen, V.Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*, 54(5):754–775, 2011.
10. M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta. Symbolic Model Checking and Safety Assessment of Altarica models. *ECEASST*, 46, 2011.
11. M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *ATVA*, pages 162–176. Springer, 2007.
12. M. Bozzano and A. Villafiorita. The FSAP/NuSMV-SA Safety Analysis Platform. *STTT*, 9(1):5–24, 2007.
13. M. Bozzano and A. Villafiorita. *Design and Safety Assessment of Critical Systems*. CRC Press (Taylor and Francis), an Auerbach Book, 2010.
14. M. Broy. Towards a Theory of Architectural Contracts: - Schemes and Patterns of Assumption/Promise Based System Specification. In *Software and Systems Safety - Specification and Verification*, pages 33–87. IOS Press, 2011.
15. A. Cimatti, M. Dorigatti, and S. Tonetta. OCRA: A Tool for Checking the Refinement of Temporal Contracts. In *ASE*, pages 702–705. IEEE, 2013.
16. A. Cimatti, M. Roveri, and S. Tonetta. Requirements validation for hybrid systems. In *Computer Aided Verification*, pages 188–203. Springer, 2009.
17. A. Cimatti and S. Tonetta. A property-based proof system for contract-based design. In *SEAA*, pages 21 –28, 2012.
18. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2):244–263, 1986.
19. W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand. Using contract-based component specifications for virtual integration testing and architecture design. In *DATE*, pages 1023–1028, 2011.
20. M.L. McKelvin Jr, G. Eirea, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. A formal approach to fault tree synthesis for the analysis of distributed fault tolerant systems. In *EMSOFT*, pages 237–246. ACM, 2005.
21. The MISSA Project. `http://www.missa-fp7.eu`.
22. nuXmv: a new eXtended model verifier. `https://nuxmv.fbk.eu`.
23. C. Pinello, L.P. Carloni, and A. Sangiovanni-Vincentelli. Fault-tolerant deployment of embedded software for cost-sensitive real-time feedback-control applications. In *DATE*, page 21164. IEEE Computer Society, 2004.

24. A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science (FOCS-77)*, pages 46–57. IEEE Computer Society Press, 1977.
25. S.A. Siddiqi and J. Huang. Hierarchical Diagnosis of Multiple Faults. In *IJCAI*, pages 581–586, 2007.
26. W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick III, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Technical report, NASA, 2002.