Complex Engineered Systems Design
Verification Based on Assume-Guarantee Reasoning

Hoda Mehrpouyan
TSYS School of Computer Science
Columbus State University
Columbus, GA
31909, USA

Dimitra Giannakopoulou, Guillaume Brat
NASA Ames Research Center
Moffett Field, CA
94035, USA

Irem Y. Tumer, Chris Hoyle
Complex Engineered Systems Design Lab
School of Mechanical, Industrial,
and Manufacturing Engineering
Oregon State University
Corvallis, OR, USA

Abstract – System verification is one of the most critical tasks into the process of engineered system design. This process is time consuming and prone with errors when a limited set of scenarios is evaluated to guarantee the correct functionality of the system. Therefore, novel design approaches and tools based on a rigorous framework for analysis, verification, and testing are very much needed. This paper provides such a framework where system properties are verified and modeled with respect to the assumptions on the environment where components and (sub)systems’ performances are guaranteed under these assumptions. To validate the proposed approach, this paper provides a case study to demonstrate how the proposed methodology reduces design complexity and presents a formal argument to assess the quality of the design.

1 Introduction

With increasing complexity in the design of complex engineered systems such as aerospace, maritime, nuclear, and major civil infrastructure systems, the cost and time required for design and development are growing at an unsustainable rate. For instance, Boeing and Airbus experienced significant delays and cost overruns in delivering their latest 787 Dreamliner jet and A380 superjumbo projects [1]. Design flaws were cited as basic issues in the A380 design and production, which resulted in a variety of glitches such as engine blow-up, failure of the backup brakes, and discovery of cracks on the wings of the planes. Boeing 787 Dreamliner faced similar issues, as described in the national transportation safety board reports of the fire incident in the auxiliary power unit (APU) on a Japan Airlines 787 flight from Boston on Jan. 2013. The report concluded that a design flaw might be the root cause of the fire. In another case, a Japan Airline Dreamliner flight from Boston faced a considerable delay after a fuel leak resulting from a faulty valve. The design flaws and equipment malfunction cost $5 Billion for Boeing on top of the $1 Billion compensation claim from airlines, such as United Airlines and Air India.

The design complexity of these types of safety-critical systems present various challenges for their safety assessment process. In order to improve the design and development process, manufacturing companies are increasingly relying on simulations to understand the unexpected behavior of the design to improve both robustness and performance of system [2]. Based on his research, Foster [3] concluded that verification process (i.e., establishing that the design, if implemented, would provide the desired functionality) consumes over 60% of the design time. In addition to longer verification time which results in project cost increase, a survey conducted by Collett International Research Inc. [4] revealed that while the traditional verification approaches, i.e. Preliminary Hazard Analysis (PHA), Failure Mode and Effect Analysis (FMEA), are still applicable, they are not sufficient; that is, they do not take into the consideration the whole complex network of relationships, between events, components, and the environments where the components operate in.

Therefore, the aim of this paper is to provide a framework for the effective use of formal methods in the early verification of safety requirements in safety critical systems.
The proposed framework allows for automatic generation of fault trees, and exhaustive safety property verification with the help of model checking algorithms. The approach is based on assume-guarantee compositional reasoning which verifies global safety properties of the system by verifying local properties of each component [5–8]. Therefore, the verification of large systems is made possible through the verification of each system components separately, while using assumptions about each components environment. While compositional reasoning based on assumptions and guarantees is popular in other domains (see, e.g., [9–11]), we are not aware of any assumption-guarantee style reasoning to verify the safety properties of complex engineered systems.

The remainder of this paper is structured as follows: Section 2 presents the background and related research on failure analysis techniques in the early stages of system design, while discussing their strengths and weaknesses. In addition, the definition of assume-guarantee reasoning and its commonly used terminologies and operators are addressed in Section 2. In Section 3, an overview of the step-by-step implementation of the assume-guarantee reasoning algorithm on the components of the design architectures is explained. Section 4 outlines the application of the proposed methodology in the analysis and verification of the safety properties of the satellite electrical power system design. The paper ends with conclusions and future work.

2 Background

A variety of modeling approaches and tools are used in industry or in academia, e.g., AADL [12], Modelica [13], Ptolemy [14], MATLAB/Simulink [15], SysML [16]. These tools are used to model the functionality and architecture of the system design, then simulation is carried out to verify the design. However, most of the simulation experiments are designed to evaluate a limited set of scenarios in order to deal with the system complexity. The effects of this informal and incomplete verification is the possibility that a non-tested scenario could result in unexpected behavior and catastrophic system failure. To address the incomplete verification of designs via simulation, formal methods have been proposed to increase the confidence level. Formal verification enables the evaluation of safety properties at different levels of abstractions, i.e., component, sub-system, system, guaranteeing the systems’ behavior in every possible scenario. It is important to note that even formal verification is as good as the abstract models and properties that one verifies on such models.

One of the objectives of the verification process is to make sure that the design complies with safety requirements. In order to satisfy most regulatory guidelines and safety standards, designers must develop a safety case to prove the safety justification of a design. These cases should represent all potential hazards and appropriate steps be taken to rectify the situation. These types of safety documents usually include safety specifications, results of failure and risk analysis, verification approach, and results of all the verification activities. Figure 1 depicts the general view of the verification process.

2.1 Design Tools For Safety and Reliability Analysis

This section provides a review of the scope and limitation of currently available system safety and reliability tools with respect to verification of complex system designs. One of these methods is the risk assessment matrix [17] which categorizes risks in the form of probability and severity (loss of function per unit time). In this approach a matrix based on the importance of severity versus the probability of failure occurrence is created. However, one limitation of this method is its lack of ability to identify failure and its propagation path. Another initial reliability study is possible through the use of Preliminary Hazard Analysis (PHA) [18] which identifies and provides a database of failures and failure propagation paths. PHA uses risk assessment matrix to assess the risk of each identified failure. The limitation of PHA is its lack of ability to evaluate risks of aggregated hazards or simultaneous design failure modes. In addition, the risk analysis of a complex system design will result in large and costly implementation. Failure Mode and Effect Analysis (FMEA) [19] is a bottom up approach that investigates failure modes of components and their effects on the rest of the system. FMEA provides an exhaustive analysis to identify the single point of failures and their effects on the rest of the system. The result of the analysis is used to increase reliability, incorporate mitigation into the design, and optimize the design. The result of FMEA analysis can be added to the PHA analysis of the design, since every failure mode of each component is evaluated and additional information about the hazards resulting from the failures is obtained. However, FMEA is very costly in terms of resources, particularly when implemented at the component level within complex systems. Also, occurrences of simultaneous failures and multiple faults is not evaluated [20, 21]. The completeness and correctness of the analysis is very much dependent on the expert knowledge.

The next group of reliability analysis methods is based on the symbolic logic of the conceptual models of failure scenarios within a design. The goal is to assess the probability of failure occurrence in the system design. One of these methods is the Reliability Block Diagram (RBD) [22], which divides the system into elements based on the functional model of the system design, where each system element is assigned a reliability factor. Then a block diagram of
the elements in a parallel, series, or the combination of parallel and series is constructed. Each block represents a function or an event in the system and each element’s failure mode is assumed independent from the rest of the system. The reliability factor may or may not be available for all the system design elements and should be assigned by an expert which make it subjective and hard to validate. Another symbolic logic model is based on the Fault Tree Analysis (FTA) [23] which studies the failure propagation path from the point of start to the vulnerable components and assigns a severity factor to each failure model. One of the benefits of using FTA is its ability to analyze the probability of simultaneous occurrence of failure within a complex systems. On the other hand, the correct probabilistic evaluation requires significant amount of resources.

These traditional safety and reliability tools require detail information about components, their failure modes and probability of failures in order to complete their analysis. On the other hand, conceptual design is a process of developing behaviors and functions to create a design solution that meets the design requirements. At this early stage of design, specific components have not yet been fully specified, therefore collecting data on failure probabilities of the components and how they propagate through the system model is very challenging. The aim of this research is to provide an early design framework that is able to automatically analyze the system architecture and verify its capability of meeting the requirements using abstract design information.

In addition, the traditional methodologies are mostly based on the evolution of the system through time, given an initial event and a set of forcing functions. These physics-based models are based on the laws of physics from the outset. However, mathematical modeling of complex systems may not be feasible since changes in operating conditions and structural dynamics can affect the mathematical model, and it makes it difficult or even impossible to develop mathematical models for all real-life conditions. Therefore, this research takes advantage of automata learning and model checking approaches. Model checking is one of the approaches to formal verification of finite state hardware and software systems [24, 25]. In this approach, a design will be modeled as a state transition system with a finite number of states and a set of transitions. The design model is in essence a finite-state machine, and the fact that it is finite makes it possible to execute an exhaustive state-space exploration to prove that the design satisfies its requirements. Since there is an exponential relationship between the number of states in the model and number of components that make up the system, the compositional reasoning approach is used to handle the large state-space problem. Compositional verification is a promising approach for alleviating the state explosion in model checking. In practice, many systems are composed of various processes running in parallel and interacting in complex ways. The safety specifications for such systems can often be decomposed into safety properties that describe the behavior of small parts of the system. Utilizing the divide and conquer technique, if we can deduce that the system satisfies each local property, and if we know that the combination of the local properties implies the overall specification, then we can conclude that the complete system satisfies the specification as well. The proposed framework supports incremental system design and verification which provides the abstraction required to reduce the inherent complexity and to ensure design satisfies safety requirements, while addressing the challenges and limitations of traditional design methods.

2.2 Verification Based On Formal Methods

In [26], Henzinger et al. cover the advantages that formal verification offers over the above approaches. In formal verification, system designers construct a precise mathematical model of the system under design, so that extensive analysis is carried out to generate proof of correctness. One of the well-established methods for automatic formal verification of the system is model checking, where a mathematical model of a system is constructed and verified with regards to specified properties. In model checking, the desired properties are defined in terms of temporal logic [27]. The defined logical formulae are then used to prove that a system design meets safety requirements and specifications. A model checker to establish assume-guarantee properties of components is called assume-guarantee reasoning (AGR) [5–8]. The assume-Guarantee approach is the evolution of compositional reasoning that interleaves abstraction and learning algorithm to perform automated compositional verification of complex systems. In the assume-guarantee reasoning (AGR) method, the system properties are verified and modeled with respect to the assumptions on the environment where component and (sub)system performances are guaranteed under these assumptions. The assumption generation methodology uses compositional and hierarchical reasoning approaches via a compositional reachability analysis (CRA) [28] technique. CRA incrementally composes and abstracts the component models into subsystem and, ultimately, a high-level system models. Based on the assume-guarantee reasoning (AGR) paradigm, assume-guarantee can be defined as a pair of assumptions and guarantees which formally describe:

1. The context in which the system design is assumed to be used.
2. The requirements which the system design demands to guarantee correct operation. (It is important to note that “guaranteeing the correct operation in an assumed environment” is only possible with a specified probability. In this context "guarantee" does not mean that system will always survive the assumed environment without any failures, it means that system will survive at the probability level in which one specifies, i.e., .9999).

Assume-guarantee reasoning has been widely used in the computer science literature as a means for software verification. As a mathematical foundation for representing the engineering requirements, this approach can be used for verification of complex engineered system design. Additionally, the work focuses on safety property specification and design verification at multiple abstraction layers.

As discussed in the previous section, abstraction and composition are the two most used principles in any sys-
system verification methodology for handling the complexity and analysis of engineered systems. When verifying complex systems, different approaches (e.g., model-based methods) use hierarchical abstraction layers such as functional, structural, and behavioral models to represent the system under study. A functional model is a representation of all the necessary functions that the system must contain in order to meet the design requirements. Kurtoglu et al. [29,30] present a framework for developing a functional model for the hardware components, while Wang et al. [31] suggest object-oriented programming for modularization and functional modeling of the software components. Functional models provide the required information about the flow of EMS and data between components throughout the system design. In functional modeling [32, 33], the EMS flows and functions are modeled using nouns and verbs respectively, e.g., store electricity, actuate electricity, etc. The functional model of the design is developed based on the hierarchical structure of functions and flows [34, 35]. Next, the structural model as a suitable design solution is developed. The structural model describes different system components and the EMS flow relationship between them. Using different design solutions within the complex system design process, various design concepts for a system are developed. While all design concepts share the same functional description, they are implemented differently. They are different in structure and behavior. Finally, the behavioral model of a component contains the nominal and failure states of the component, including transitions leading to these states. The behavioral model results from the relationship between input/output flows and the underlying first principles. Once the behavioral models of the component are developed, they are incorporated into the Labeled Transition Systems (LTSs) model by mapping them with their respective LTSs transitions.

3 Methodology

The contribution of this paper is developing an automated design verification framework to prove the correctness of the complex engineered system design with regards to its functional and safety properties. The proposed framework provides information on the property violation of the composed components during conceptual design, while identifying the failure propagation behavior. The automatic generation of failure propagation paths enables the system designers to better address the safety issues in the design.

3.1 System Modeling

In the proposed approach, finite-state model of a system is analyzed to ensure satisfaction of safety properties that assure a desired system behavior. Finite-state model is a representation of the system behavior that is generated in the form of Finite State Process (FSP) [36]. FSP is an algebraic notation that is used to describe the component’s behavior (Table 1). System designers create the FSP models which are designed to be machine readable, and thus provides an ideal language to specify abstract model of the component’s behavior or function. The developed FSP is then used through a modeling tool such as the Labelled Transition System Analyzer (LTSA) [37] to provide compilation of FSPs into a Labelled Transition System (LTS) [37, 38]. The LTS model is expressed graphically by its alphabet, transition relation, and states including single initial state (Figure 2). The LTS of the system is constructed from the LTS of its subsystems, and is verified against safety properties of the design requirements. In this research, the model checking algorithm is integrated as part of the LTSA tool which performs exhaustive execution of all system’s behavior to determine if the safety property is violated or not. The LTSA takes advantage of the method called compositional reachability analysis (CRA) and creates a reachability graph for the system which contains information about the safety values of each state. A safety property then is checked by analyzing the reachability graph, searching for paths on which the safety property is violated. Labelled Transition Systems (LTS) $T$ is defined as:

- A set $S$ of states
- A set $L$ of actions
- A set $\rightarrow$ of transitions from one state to another.
- An initial state $s_0 \in S$

$$T = (S, L, \rightarrow, s_0).$$

3.2 Parallel Composition of LTSs

In this section the parallel composition operator [39] and the composition mechanism that assists with compositional modeling of the component models are explained. The parallel composition operator enables both associative and commutative composition; therefore the order of LTSs models that are composed together is insignificant. The parallel composition operator, denoted by “$\parallel$”, is a binary operator that accepts two LTSs as an input argument. Based on the definition of this operator, composed LTSs interact by synchronizing on common actions (i.e., exchange of EMS) shared in their FSP models with interleaving of the remaining actions. Designing interacting components with LTSs is therefore sensitive to the selection of action names. In addition, parallel composition is based on the model instantiation which is defined by constructing a copy of a LTS model where each transition label is prefixed by the name of the instance.

3.3 Electrical Power System Design

Validation of the proposed verification framework is through application to an Electrical Power System (EPS) which is designed to provide power to selected loads. In an aerospace vehicle these loads usually include subsystems such as the propulsion, avionics, and thermal management systems. The basic functionality that EPS is required to provide is common to many aerospace applications such as power storage, power distribution, and operation of loads [40]. Figure 3 displays the existing design of the EPS, containing a power source connected through a series of relays to an inverter and several loads consisting of a large fan, a Direct Current (DC) resistor and a Alternating Current (AC) resistor. In order to create an integrated health manage-
ment environment a sensor suite is designed to enable monitoring of currents, voltages, and temperatures throughout the circuit. A series of four AC or DC voltage sensors and three current transmitters measure the voltage and current at different points throughout the circuit. This is the Modelica [41] representation of the EPS design.

In the EPS test-bed, the power source component that is denoted by a circle may have the operational modes on and off. The on-mode has the functional action of generating power, which can also result in over-current spikes. As illustrated in Figure 3, the generated spike affects the AC resistor, fan, and DC resistor that are denoted by circles on the right-hand side of the figure. These components are vulnerable to the spike generated by the power source. Thus, safety properties are needed to protect these vulnerable components and ensure the proper operation of the whole system. The safety properties define the types of failure that a component is vulnerable to and must be checked to ensure the failure state is not reached. There are three different paths 1. \{A, B, C\}, 2. \{A, B, D\}, and 3. \{A, E\} in which the generated spike from the power source can reach the three vulnerable components; these are considered design flaws [42].

3.4 Reusable Models and Binding Interfaces

In the proposed system modeling approach, each component defines a scope for the transition in its behavioral model (e.g., an instance of the resistor is defined by the use of the command res:resistor). Instantiation permits the reuse of LTSs during system modeling through multiple instantiations, e.g., Alternating Current (AC) and Direct Current (DC) resistors behave in a similar fashion under the influence when the power received from the environment exceeds the resistor’s ability to dissipate the heat and therefore the resistor can be used for both components.) Instantiation creates unique labeling of transitions in the LTS models (e.g., each transition in the AC resistor’s behavioral model is labeled with a prefix of "ACres." and each transition in the DC resistor’s model is labeled with a prefix of "DCres.").

In order to create the compositional model of the circuit
breaker and AC resistor, an instance of the circuit breaker is defined by the use of the command \texttt{cb:CircuitBreaker} that is composed with the previously defined instance \texttt{ACres}. However, the two LTSs do not have any alphabet in common so no synchronization is possible. For this reason, the binding between the two models is created by the use of the command \texttt{ACres.inflow/cb.outflow}. The binding leads to a model where the circuit breaker’s output current is recognized as the AC resistor’s input current. As a result, the new label \texttt{ACres.inflow}, is substituted with the old label, \texttt{cb.outflow}. With the binding command in place, the synchronization takes place and the LTSs components have a common action to communicate.

In our context, properties are modeled as safety LTSs. A safety LTS is a LTS that contains no failure states. When checking a property \( P \), an error LTS denoted \( P_{err} \) is created, which identifies possible violations with the failure state.

### 3.5 Verification Process

The contribution of this paper is developing an automated design verification framework to prove the correctness of the complex engineered system design with regards to its functional and safety properties. The proposed framework provides information on the property violation of the composed components during conceptual design, while identifying the failure propagation behavior. The automatic generation of failure propagation paths enables the system designers to better address the safety issues in the design. Figure 4 depicts the relationship between system models and the verification framework that either provides the proof of correctness or failure propagation information.

The main steps to apply the assume-guarantee reasoning (AGR) framework are summarized as follows:

1. A functional model is generated as a representation of all the necessary functions that the system must contain in order to meet the design requirements is used to create the structural model of the system.

2. The structural model is used as a resource to gather information about the different system components and the Energy, Material, and Signal (EMS) flow relationship between them to create a database of their failure states and safety properties and the behavioral models.

3. Once the behavioral models of the component are developed, they are incorporated into the Finite State Processes (FSP) by mapping the behavior models with their respective FSPs’ transitions.

4. The Labeled Transition System (LTS) of the system is automatically constructed from the FSP models that are developed by design engineers.

5. The LTSs are incrementally analyzed and abstracted...
through the use of a reachability graph to determine whether the safety property is violated.

6. If the failure state of the component is reached, then the failure propagation path is provided.

The AGR paradigm requires exact identification of the component properties, which in this case are defined based on the failed states, of the components and (sub)systems. In order to identify the failed states the effects of incoming EMS flows on the operation of the components is analyzed and two generic states of nominal and failed are defined. The component’s state is recognized as nominal when a component is operating with the performance and functionality intended by the system designer. On the other hand, failure state is defined as a component functioning in a way that was not intended by the designer.

3.6 Automated Model Checker Based On Assume-Guarantee Reasoning

In the assume-guarantee paradigm, the formula is a triplet \( \langle A \rangle M \parallel P \), where \( M \) is a component, \( P \) is a property, and \( A \) is an assumption about \( M \)’s environment. The formula is true if whenever \( M \) is part of a system satisfying \( A \), then the system must also guarantee \( P \).

Let \( M \) be a finite-state component (Table 1) with \( \Sigma \) being the set of its interaction points with the environment, and let \( P \) be a safety property. Then there is a natural notion of the weakest assumption \( A_w \), such that \( \langle A_w \rangle M \parallel P \) holds, where \( A_w = \Sigma \). \( A_w \) characterizes all possible environments \( E \) under which the property holds.

It has been shown that, for any finite-state component \( M \), the weakest assumption \( A_w \) exists and can be constructed algorithmically [5]. An ideal \( A_w \) should precisely represent the component in all its intended usages. It should be safe, meaning that it should exclude all problematic interactions, and permissive, in that it should include all the good interactions.

A weakest assumption is generated so that it is both safe and permissive. Safety in this context means to restrict the behaviors of the component to those that satisfy \( P \). Permissiveness, on the other hand is concerned with restricting behaviors only if necessary. Permissiveness is desirable, because \( A_w \) is then appropriate for deciding whether an environment \( E \) is suitable for \( M \) (if \( E \) does not satisfy \( A_w \), then \( E \parallel M \) does not satisfy \( P \)). The simplest assume guarantee rule for checking a safety property \( P \) on a system with two components \( M_1 \) and \( M_2 \) is defined as

**Rule ASYM:**

\[
\begin{align*}
1: & \langle A \rangle M_1 \parallel P \\
2: & \langle \text{true} \rangle M_2 \parallel A \\
\Rightarrow & \langle \text{true} \rangle M_1 \parallel M_2 \parallel P
\end{align*}
\]

In this rule, \( A \) denotes an assumption about the environment of \( M_1 \). Note that the rule is not symmetric in its use of the two components and does not support circularity. Despite its simplicity, experience has shown it to be quite useful in the context of checking safety properties.

The objective is to automatically generate the weakest assumptions for components and their compositions, so that the assume-guarantee rule is derived in an incremental manner. In this research, the algorithm in [43] is used to generate approximate assumptions and guarantees at the component, subsystem, and system level. Figure 5 depicts the algorithm which is based on model checking and machine learning techniques to construct an initial assumption derived from component’s behavioral models. Model checking is used to make sure the generated assumption is safe and permissive based on the assume-guarantee rule. The first step of the algorithm is to check for the first part of the Rule ASYM: \( \langle A \rangle M_1 \parallel P \). If it is violated, it means that the assumption is too weak so it does not prevent \( M_1 \) from reaching its failure state. Based on the generated failure propagation path, the algorithm creates a new assumption which is stronger than the previous one. The iteration continues until the first rule of \( \langle A \rangle M_1 \parallel P \) is addressed. The next step is to check the second rule \( \langle \text{true} \rangle M_2 \parallel A \). If the rule holds then it is concluded that \( \langle \text{true} \rangle M_1 \parallel M_2 \parallel P \), otherwise the failure propagation path is generated to provide the reason why component \( M_2 \) is not able to guarantee \( A \). Then the counterexample is analyzed to realize if component \( M_1 \) reaches its failure state or not. If \( M_1 \) does not reach its failure states then the assumption is too strong and should be weakened and tested again. Otherwise, it is concluded that the composition of \( M_1 \parallel M_2 \) violates property \( P \).

3.6.1 Soundness and Completeness

Soundness of an assume-guarantee rule means that whenever its premises hold, its conclusion holds as well. Without soundness, we cannot rely on the correctness of conclusions reached by applications of the rule, which makes the rule useless for verification. On the other hand, completeness states that whenever the conclusion of the rule is correct, the rule is applicable, i.e., there exist suitable assumptions such that the premises of the rule hold. While completeness is not needed to ensure correctness of proofs obtained by the rule, it is important as a measure for the usability of the rule [5].
3.7 The LTSA Tool

The Labeled Transition System Analyzer (LTSA) [37, 44] is an automated tool that supports Compositional Reachability Analysis (CRA) [28] of a software system based on its architecture. In general, the conceptual design of a complex engineered system has a hierarchical structure and is modular [45]. CRA incrementally computes and abstracts the behavior of composite components based on the behavior of their immediate children in the hierarchy. The input language “FSP” of the tool is a process-algebra style notation with Labeled Transition Systems (LTS) semantics.

A property is also expressed as an LTS, but with extended semantics, and is treated as an ordinary component during composition. Properties are combined with the components to which they refer. They do not interfere with system behavior unless they are violated. In the presence of violations, the properties introduced may reduce the state space of the (sub)systems analyzed. As in our approach, the LTSA framework treats components as open systems that may only satisfy some requirements in specific contexts. By composing components with their properties it postpones analysis until the system is closed, meaning that all contextual behavior that is applicable has been provided.

4 Case Study

Continuing with the EPS design concept, the development of LTSs implies direct mapping between the functional model (Figure 6) and the structural architecture of the system, in which specific component or (sub)system is selected to implement the functional requirements in the actual system design.

In order to construct the LTS model of the EPS design, all internal state transitions of the components are presented in the Finite State Processes (FSP) language. The constant variables(factors that do not change during the course of this experiment) and ranges are defined as follows:

\[
\begin{align*}
&\text{const Low} = 0 \\
&\text{const Medium} = 1 \\
&\text{const Spike} = 2 \\
&\text{const Open} = 1 \\
&\text{const Close} = 0 \\
&\text{range CUR} = \text{Low..Spike}
\end{align*}
\]

The notations 0, 1, and 2 are used to denote low, medium, and spike currents in the EPS components, respectively.

4.1 Primitive Components and Properties

As depicted in Table 1 all internal state transitions of the primitive components are presented in the FSP language. Each component’s nominal behavioral model is incorporated into the FSP code, therefore the resulting model contains no failure state. For example, while the AC resistor is in the operational mode (Table 1), two transitions are possible, which are specified using the "OR" logical operator "|". These two transitions are defined as: 1- the current input into the AC resistor which is in the range of low or medium and the resulting output current also in the low or medium range 2- the current in-flow to the AC resistor is spiking which results in the state of "burn". If the resistor is in the state of burn, no matter what input current level to the AC resistor, there is no output current past the AC resistor.

<table>
<thead>
<tr>
<th>Component</th>
<th>Mode</th>
<th>FSP Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery</td>
<td>nominal</td>
<td>(inflow[v:CUR] → outflow[v] → Battery)</td>
</tr>
<tr>
<td>Relay</td>
<td>nominal</td>
<td>(inflow[v:CUR] → if (Open) then (outflow[v] → Relay else Relay))</td>
</tr>
<tr>
<td>Inverter</td>
<td>nominal</td>
<td>(inflow[v:CUR] → outflow[v] → Inverter)</td>
</tr>
<tr>
<td>DC Resistor</td>
<td>failure</td>
<td>same as AC Resistor</td>
</tr>
<tr>
<td>Fan</td>
<td>failure</td>
<td>same as AC Resistor</td>
</tr>
</tbody>
</table>

The failure modes of components are represented by $P_{err}$. The error LTSs are constructed to represent all the faulty transitions that lead to failure states. In order to model the failure mode of the three vulnerable components discussed in design architecture of Figure 3, a generic property named $P_{err}$ is defined for all three components as below:

\[
\text{property } P_{err} = \text{STOP } + \text{burn}.
\]

The LTSA tool represents failure state by $J$ as it is depicted in Figure 2 in the compositional model of the AC resistor with its property that is reached by the illegal transition of acres.burn.

4.2 Compositional Model

In order to create the compositional model of the EPS system, the order of compositions is decided based on the functional model of the design. Table 2 represents the compositional model of the EPS system for two types of components. Those components that operate in nominal mode...
such as "battery" and "currentsensor240" in module18 are composed by the creation of binding between them. The binding is modeled by the EMS flow between the two components which is represented by the use of the command cs240.inflow[bat2.outflow]. The binding leads to the compositional model where the battery’s output flow is recognized as currentsensor240’s input flow. The second types of components are those with failure states, which are required to be composed with their defined properties before they can be considered for composition with other components, e.g., module1 through module3 in Table 2.

Table 2: Composition of the EPS components

| Module1 = (acRes(Vulnerable \ P_{err})). |
| Module2 = (fan(Vulnerable \ P_{err})). |
| Module3 = (vm256:VoltMeter / cs267.inflow/vm256.outflow). |
| Module4 = (im:Inverter / vm256.inflow/im2.outflow). |
| Module5 = (rel272:Relay / acRes.inflow/rel272.outflow). |
| Module6 = (rel275:Relay / fan.inflow/rel275.outflow). |
| Module9 = (vm256:VoltMeter / cs267.inflow/vm256.outflow). |
| Module10 = (vm281:VoltMeter / rel284.inflow/vm281.outflow). |
| Module14 = (rel244:Relay / vm242.inflow/rel244.outflow). |
| Module15 = (vm240:VoltMeter / rel244.inflow/vm240.outflow). |
| Module16 = (cs240:CurrentSensor / vm240.inflow/cs240.outflow). |
| Module17 = (bat2:Battery / cs240.inflow/bat2.outflow). |

Fig. 7: Parallel Composition in LTS Format

4.3 Compositional Verification

In order to verify the properties of the EPS system, the LTSA "compositional" algorithm is used. This algorithm implements assume-guarantee reasoning in a learning framework to prove that the properties are satisfied or violated.

The advantage of using model checking and automata learning algorithm is its ability to perform CRA in an exhaustive manner to search for violations of design properties. In addition, the LTSA algorithm uses a specific form of learning algorithm based on minimization and abstraction, which dramatically reduces the number of state spaces required for analysis. For example, if the two modules of the EPS LTSs, e.g., module18 and module17 are analyzed in a monolithic manner (\ Test = (Module18 \ Module17) \ the state space of this composition results in 16 states with 27 transitions as illustrated in Figure 7. Eventually, the full monolithic composition of the EPS design results in approximately 232 × 10⁹ states, however, with the proposed method in this paper, the compositional analysis is completed in 2 seconds.

The result of EPS compositional verification concluded by the AGR was that the "system and environment are incompatible". The reason for this conclusion is that the designers of the EPS system assumed normal operating condition for the system at all times. In normal condition, all three susceptible components receive nominal voltage and current, while any variation in load and distribution has an effect on the system. Therefore, the analyzed design is not considered fault tolerant.

In addition to verifying the desired properties of the system design, the proposed methodology automatically computes the required assume-guarantee pair for each component in the design to prove the global properties of the design under consideration. There are cases where no assume-guarantee pair is generated by the verification algorithm because there is no environment in which the design can be implemented safely. Figure 8 represents the assume-guarantee pairs generated by the AGR for each system element in the design, which implies that each component guarantees to output current flow of low or medium (0 or 1) if they receive current input flow of low or medium. In the case of detecting safety violation in the system design, the verification framework returns a counterexample, which provides information of the failure propagation path. Only one counterexample is necessary to prove that the design violates its prop-
The "No Burn current to the AC resistor is always assumed-guarantee reasoning rule of triple type: and fan from burning. The following equation represents the circuit breakers in the design architecture prevents the resistors protecting an electrical circuit from damage caused by overload and over-current spike. The operation of the circuit breaker is used to protect the AC resistor, fan, and DC resistor from able components. The circles highlight the circuit breakers in Figure 9 to prevent the spike reaching the three vulner-
is required to add circuit breakers to the design as modeled 4.4 Design Based On The Result Of Verification

In order to correct the design flaws mentioned above, it is required to add circuit breakers to the design as modeled in Figure 9 to prevent the spike reaching the three vulnerable components. The circles highlight the circuit breakers used to protect the AC resistor, fan, and DC resistor from an over-current spike. The operation of the circuit breaker is similar to that of an electrical switch, which is designed to protect an electrical circuit from damage caused by overload or short circuit (Table 3). Therefore, the integration of circuit breakers in the design architecture prevents the resistors and fan from burning. The following equation represents the assume-guarantee reasoning rule of triple type:

1. \(<\{0..1\}>\text{ACResistor}<\{\text{No Burn}\}>
2. \(<\text{true}>\text{CircuitBreaker}<\{0..1\}>\)

(1) \(<\{0..1\}>\text{AC resistor}<\{\text{No Burn}\}>\) is proven correct if circuit breaker satisfies the assumption that in-flow current to the AC resistor is always \{0..1\}, resulting in guaranteeing property No Burn.

The compositional model of the modified design is represented in Table 4. The full monolithic composition of the modified EPS design results in $19 \times 10^{12}$ states, however, with the proposed method in this paper, the compositional analysis is completed in 2.5 seconds. The result of verification is successful, implying that the "system and environment are compatible" and all the design safety requirements are met.

It is important to note that the generated assume-guarantee pairs, as depicted in Table 5, restrict the current-inflow of low and medium, [0, 1], for a smaller number of components, compared to the EPS design without the circuit breakers. The reason for this is that the circuitbreaker$_{280}$ detects a fault condition and interrupt current flow from reaching the two vulnerable components (AC resistor and fan) in the top branch, while the circuitbreaker$_{280}$ protects the lower branch. Therefore, any components before these two circuit breakers can accept the in-flow current of low, medium, and spike. This is a good indication of the weakest assumptions generated by the proposed framework, guiding the designers in their understanding of the design requirements. Based on the generated assumptions the two circuit breakers (236 and 262) are not required and therefore can be eliminated from the design.

The second alternative design architecture of the EPS system with removed circuit breakers has been verified as a safe design. Tables 6 illustrates the assume-guarantee pair generated for each design component.

As illustrated, the generated assume-guarantee pair is critical in choosing alternative design solutions and migrating between different design architectures with relatively small effort. In addition, the generated assume-guarantee pair enables the system designers to trace the requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>Mode</th>
<th>LTS Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit Breaker</td>
<td>nominal</td>
<td>(inflowv&lt;CUR&gt; to if (v &lt; Spike) then (outflowv&lt;1&gt; CircuitBreaker) else CircuitBreaker) + [outflowv][Spike])</td>
</tr>
</tbody>
</table>
throughout the design architecture. The behavior of the system design is described in an assume-guarantee style specification: a component guarantees certain set of behaviors, given that its environment follows certain assumptions.

In addition, a performance case study is conducted by comparing the performance results of the proposed verification approach with a monolithic approach (Figure 10) which represents linear growth for the proposed verification process, while an exponential growth is predicted for the monolithic verification process.

5 Conclusion and Future Work

This work provides a framework for the effective use of formal methods in the early verification of safety requirements in safety critical systems. This is especially important in proving the correctness of the system design, where it is critical to guarantee that the known interactions between system components do not violate any safety properties. With regard to the design of complex systems, high level system requirements are decomposed into component and (sub)system requirements which logically map to the architectural decomposition of the system. Therefore, proof of
correctness through pre-verification of system components and compositional reasoning is made possible. The aim of compositional reasoning is to improve scalability of the design verification problem by decomposing the original verification task into sub-problems. The simplification is based on the assume-guarantee reasoning that results in approximating the requirements which a component and (sub)system places on its operational environment to satisfy safety properties. The case study of the EPS design demonstrated the capability of the proposed verification methodology to perform virtual integration of system elements and proving system-level requirements from the constraints allocated on the components. As a result, a class of design flaws has been uncovered because of an integration failure that occurs when system components satisfy their requirements in isolation but not at the system-level.

The proposed approach models the behavior of composite components using LTS models of the primitive components and their safety properties, which are based on the structural model provided by the Modelica model of the system design. In addition, a fully automated compositional verification technique is used to determine the correctness of the design with regards to its requirement and generate pairs of assume-guarantee using a learning algorithm. Experimental results showed the effectiveness of the compositional reasoning approach in reducing the complexity of the verification process by using modularity and abstraction. In addition, we showed how a deductive verification tool such as LTSA combined with LTS models can be used for verification of finite-state hardware system designs. The compositional verification helps in breaking a large complex system design into smaller parts whose verification can be checked in order to prove that the safety property of the components and the (sub)system holds. The assume-guarantee approach which is based on a learning algorithm [5], produces and refines assumptions depending on failure propagation paths and queries, the verification process is assured [5] to terminate. In addition, the algorithm returns counterexamples which include failure propagation information in the early stages of conceptual design. Another advantage of the proposed approach for verification of engineered systems is its independence from human intervention and expert user in devising the appropriate assume-guarantee pair. The experimental results provided strong evidence in favor of this line of research.

It must be noted that state explosion is an inherent limitation of model checking, therefore no single technique is expected to be efficient for all kinds of systems. This is also reflected by the fact that most of the existing model-checking tools support several approaches to model checking. The future work will concentrate to build experience in order to determine what types of analyses are appropriate for what kinds of systems. Specifically, we will focus on the design verification and analysis of complex engineered systems with software controlling the hardware. The design of such systems requires collaboration between experts from different design domains to formally define their interacting behavior and verify conflicting requirements or objectives.

In addition, future work expands the approach discussed in this paper to examine the learning algorithm and its generated assumptions to determine the most reliable design architecture of the redundant systems. It is our goal to investigate different aspects of fault tolerant system design requirements while taking into account automatic injection of multiple failures and reasoning about different types of recovery strategies. As a result, the existing verification technique is required to be modified to include systems that exhibit probabilistic behavior. The approach will be based on the multi-objective probabilistic model checking. Properties of these models are formally defined as probabilistic safety properties.

References


