Int J Softw Tools Technol Transfer (2007) 9:5–24
DOI 10.1007/s10009-006-0001-2

REGULAR CONTRIBUTIONS

The FSAP/NuSMV-SA safety analysis platform

Marco Bozzano · Adolfo Villafiorita

Published online: 29 June 2006

Abstract Safety-critical systems are becoming more complex, both in the type of functionality they provide and in the way they are demanded to interact with the environment. Such a growing complexity requires an adequate increase in the capability of safety engineers to assess system safety, including analyzing the behavior of a system in degraded situations. Formal verification techniques, like symbolic model checking, have the potential of dealing with such a complexity and are now being used more often. However, existing techniques have little tool support and therefore their use for safety analysis remains limited. In this paper, we present FSAP/NuSMV-SA, a platform which aims to improve the development cycle of complex systems by providing a uniform environment that can be used both at design time and for safety assessment. The platform makes the modeling and safety assessment of complex systems easier by providing a facility for automatically augmenting a system model with failure modes, whose definitions are retrieved from a predefined library. In this way, it is possible to assess the system safety both in nominal conditions and in user-specified degraded situations, i.e., in the presence of faults. Furthermore, the platform provides a pattern-based definition of temporal logic formulas, which simplifies the definition of safety requirements. The platform consists of a graphical user interface (FSAP) and an engine (NuSMV-SA) which is based on the NuSMV model checker. The model checking engine provides a support for system simulation and standard model checking capabilities, like property verification and the generation of counterexamples. Furthermore, algorithms have been implemented to automate the generation of artifacts that are typical of reliability analysis, e.g., fault trees. The platform can derive fault trees automatically (for both monotonic and non-monotonic systems) from the definition of the system model and of the possible faults. The interface of the platform has been designed to improve usability for people who are not expert in formal verification. The platform has been evaluated in collaboration with an industrial partner and tested on some industrial case studies.

1 Introduction

Safety-critical systems are typically required to operate not only in nominal conditions — i.e., when all of the (sub)components of the system work as expected — but also in degraded situations, i.e., when some of the system are not working properly. Guaranteeing this property typically requires systems to be engineered using development processes in which safety is considered from the early stages of development. In aeronautics for instance, safety requirements stating the (degraded) conditions under which systems must remain operational are defined along with the other system requirements. During system development, the development activities are conducted in parallel with a set of safety analysis activities that have the specific goal of identifying all possible hazards. The identification of hazards, together with their relevant causes, is necessary to assess whether the system behaves as required in all operational conditions. These activities are crucial for system certification to ensure that the development process is able to guarantee the specific safety level assigned to the system. In order

This work has been developed within the European-sponsored projects ESACS, contract no. G4RD-CT-2000-00361, and ISAAC, contract no. AST3-CT-2003-501848.

M. Bozzano (✉) · A. Villafiorita
ITC-IRST, Via Sommarive 18, 38050 Trento, Italy
e-mail: bozzano@irst.itc.it

A. Villafiorita
e-mail: adolfo@irst.itc.it

to certify the system — a necessary step for its deployment and use — the safety requirements must be demonstrated to hold. Safety analysis activities produce artifacts, such as fault trees and failure mode and effect tables, that represent the combinations of failures causing the violation of safety requirements, the effect of failures on the system, and the computation of the probability relevant to the violation of the safety requirements.

Safety-critical systems are becoming more complex, both in the type of functionality they provide and in the way they are demanded to interact with their environment. Such growing complexity requires an adequate increase in the capability of safety engineers to assess system safety. Current informal methodologies, like manual fault tree analysis (FTA) and failure mode and effect analysis (FMEA) [67], rely on the ability of the safety engineer to understand and to foresee the system behavior. As a consequence, these tasks are becoming more time consuming and complex to perform. Emerging techniques like formal methods [68], and in particular model checking [25], are increasingly being used for the verification of real-world safety-critical industrial applications (see, e.g., [18,21,24,42]). These methods allow a more thorough verification of the system's correctness with respect to the requirements, by using automated and (hopefully) exhaustive verification procedures. The use of model checking techniques for reliability and safety analysis, however, is still in its infancy. Very often, existing techniques have little tool support. Moreover, even when these methods are applied, the information linking the design and the safety assessment phases is often carried out informally. The link between design and safety analysis may be seen as an "over the wall process" [38].

In this paper, we present the FSAP/NuSMV-SA platform[1], which is being developed at ITC-IRST. The platform consists of two main components: FSAP (Formal Safety Analysis Platform), which provides a graphical front-end to the user, and NuSMV-SA, based on the NuSMV [22,23] model checker[2], which provides the safety assessment capabilities.

FSAP/NuSMV-SA provides a uniform environment that can be used for the design and safety assessment of complex systems. The platform provides a facility for automatically augmenting a system model with failure modes, whose definition is retrieved from a predefined library. In this way, it is possible to assess the system safety both in nominal conditions and in user-specified degraded situations, i.e., in presence of faults. Furthermore, the platform provides a pattern-based definition of temporal logic formulas, which simplifies the definition of safety requirements.

The NuSMV-SA engine provides support for user-guided or random simulation, as well as standard model checking capabilities like property verification and counterexample trace generation. Furthermore, NuSMV-SA implements algorithms that automate the generation of artifacts that are typical of reliability analysis, like fault trees. The fault tree construction can be performed automatically both for monotonic systems and for non-monotonic ones (the difference being that in the case of non-monotonic systems, events requiring that system components do not fail can be part of the results of the analysis). Additionally, NuSMV-SA can perform the so-called failure ordering analysis [15]. The platform also provides a simple repository, which links the analyses that have been performed to the model, thus helping the development and safety analysis process in case of system changes.

The basic functions of the platform can be combined in different ways. The possibility of using the same models for design and safety analysis and the use of standard notations to present safety analysis results (e.g., fault trees) provide a high degree of flexibility in integrating the platform in different development and safety analysis processes. It is possible, for instance, to support an incremental approach, based on iterative releases of a given system model at different levels of detail. Increasing the level of detail means refining the model, or adding further failure modes and/or safety requirements (see Sect. 5 for an example).

FSAP/NuSMV-SA provides a flexible environment that can be used both by design engineers for the formal verification of a system and by safety engineers to automate certain phases of safety assessment. The major benefits are a tighter integration between the design and the safety analysis teams, and (partial) automation of the activities related to both verification and safety assessment. This, in turn, provides the opportunity to perform safety analyses faster and at the early stages of the development process, thereby providing useful feedback to drive the design/refinement process.

The FSAP/NuSMV-SA platform has been developed within the ESACS[3] project [16] (Enhanced Safety Assessment for Complex Systems), a European-Union-sponsored project in the area of safety analysis, involving several research institutions and leading companies in the fields of avionics and aerospace. Within the project, the industrial and research partners have devised a methodology for guiding the integration of formal techniques in the safety analysis process. This methodology is supported by the state-of-the-art and commercial tools for system modeling and safety analysis. The tools can be combined in (a limited set of) different configurations that are tailored to the needs of the industrial partners participating in the project. The different

[1] `http://sra.itc.it/tools/FSAP.`

[2] `http://nusmv.itc.it.`

[3] `http://www.esacs.org.`

configurations are collectively referred to by the name "ESACS platform". Both the methodology and the ESACS platform have been tested on a set of industrial case studies. The FSAP/NuSMV-SA platform has been evaluated in collaboration with Alenia Aeronautica (the leading Italian company in the avionics field), and Società Italiana Avionica (SIA).

The rest of the paper is structured as follows. In Sect. 2 we give an overview of the ESACS methodology. In Sect. 3 we present the architecture of the system and in Sect. 4 we provide a more detailed description of the functions and use of FSAP/NuSMV-SA. In Sect. 5 we discuss our experience on the use of the platform for safety assessment of systems of industrial relevance. In Sect. 6 we discuss some related work. Finally, in Sect. 7 we outline our future work and in Sect. 8 we draw some conclusions.

2 The ESACS methodology

In order to understand the typical scenario of use of FSAP/NuSMV-SA we present, in this section, the ESACS methodology. The main characteristic of the ESACS methodology, which is strongly tool-based, is the capability of integrating the system design and the system safety assessment processes by providing an environment in which formal notations are the common and shared language between design and safety analysis. The methodology, sketched in Fig. 1, is based on the steps described in the following subsections.

2.1 Model capturing

The starting point of the ESACS methodology is a formal model, i.e., a model written in some formal language. The formal model can be either written by the design engineer or by the safety engineer. This alternative gives rise to two different scenarios. In the first scenario, the formal model is called system model (SM), it is written by the design engineer, and it includes only the nominal behavior of the system. This model is used by the design engineer to verify the functional requirements, and it is then passed to the safety engineer for safety assessment. In order to validate the system with respect to the safety requirements, the safety engineers will enrich the behaviour of the SM by injecting failure modes on the SM, as described in more details below[4].

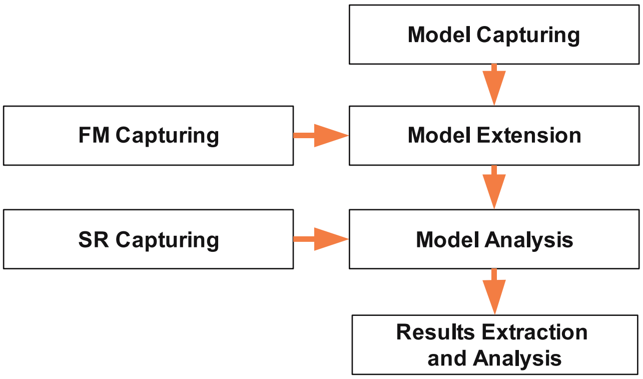


Fig. 1 ESACS methodology steps

In the second scenario, the formal model is built directly by the safety engineer and it is called formal safety model (FoSaM). This model represents a formal view of the system highlighting its safety characteristics. To write a FoSAM, the safety engineer can browse a library of system components (including both nominal and faulty behaviours) and a library of architectural safety patterns. This second scenario occurs during the early phases of the system life cycle, when there are still no design models available, but only some system specification. In this second scenario, the main goal is to assess the system architecture.

2.2 Failure mode capturing and model extension

The second step of the methodology includes the failure modes (FMs) capturing and the model extension phases. When the system model (SM) is written by the design engineer, it must be extended by injecting the failure modes, i.e., a specification of how the various components of the system can fail. This step yields a model that we call extended system model (ESM), in which all the components of the SM can fail according to the specified failure modes. The failure mode types to be injected into a SM can be stored and retrieved from a library of generic failure modes, the Generic Failure Modes Library (GFML) and then automatically injected into the formal system model through an extension facility.

2.3 Safety requirements capturing

As long as a SM/ESM or a FoSaM is available, it is possible to verify its behavior with respect to the desired functional (nominal behavior) and safety requirements (degraded behavior). During the safety requirements capturing phase, design and safety engineers define functional and safety requirements that will be used at a later stage to assess the behavior of the system. In particular, the design engineer and/or the safety engineer will verify the system either by writing the system requirements using some formal notation (e.g., temporal logic [36]) or by loading the basic safety requirements from a Generic Safety Requirement Library (GSRL).

[4] Note that by fault injection we mean the extension of the system model with a specification of the possible failure modes. We use this terminology, which is standard in the ESACS project, even though it may not be fully appropriate.

2.4 Model analysis

This is the phase in which the behavior of a system is assessed against the functional and safety requirements. The model analysis phase is performed by running formal verification tools (e.g., the NuSMV-SA model checker) on the given system properties.

Model analysis includes two main verification tasks. In the case of a system property, the model checking engine can test validity of the property, and generate a counterexample in case the system property is not verified. For instance, if we consider a property that is required to hold for every possible path of the system, the model checking engine will generate a counterexample showing one particular path along which the property has failed.

In case of a safety requirement, the model checking engine generates all possible minimal combinations of components failures, called Minimal Cut Sets (MCS), that violate the safety requirements. MCS can be arranged in the fault tree representation [67]. Fault trees provide a convenient representation of the combination of events resulting in the violation of a given top level event, and are usually represented in a graphical way, as a parallel or sequential combination of AND/OR logical gates.

2.5 Result extraction and analysis

During this phase, the results produced by the model analysis phase are processed and presented in human-readable format. In particular, the result extraction phase is responsible for displaying all the outputs automatically generated by the model checking engine (e.g., simulation traces and minimal cut sets) and to present results of safety analyses in formats that are compatible with traditional fault tree analysis tools used by safety engineers.

After discussing the ESACS methodology, in Sect. 3 we will describe the FSAP/NuSMV-SA architecture, whereas in Sect. 4 we will explain how the tool can be used in the context of this methodology.

3 The FSAP/NuSMV-SA architecture

FSAP/NuSMV-SA consists of two main components:

- FSAP (Formal Safety Analysis Platform) provides a graphical user interface and a repository that can be used by safety engineers and design engineers to share information related to the system under development and the analyses performed;
- NuSMV-SA, based on the NuSMV model checker, provides the core algorithms for formal verification.

FSAP/NuSMV-SA is implemented in C++ as a cross-platform tool. As a result, it currently runs on Windows and Linux platforms. FSAP, the graphical user interface, is based on the FLTK[5] cross-platform toolkit. All the data produced by the platform are stored in XML format, and the corresponding parser is based on the cross-platform Expat[6] library.

NuSMV-SA, the engine, is an extension of the model checking tool NuSMV [22], a symbolic model-checker developed at ITC-IRST. It originated from a re-engineering and re-implementation of SMV [52]. NuSMV is a well structured, open, flexible, and well-documented platform for model checking, and it has been designed to be robust and close to industrial standards.

NuSMV offers a textual input language to describe finite-state machines. One can specify a system as a synchronous Mealy machine, or as an asynchronous network of nondeterministic processes. The language provides for modular hierarchical descriptions, and for the definition of reusable components. The data types in the language are booleans, integer subsets, scalars, and fixed arrays (the integration of the word and real data type is planned for future releases — see also Sect. 7). System specifications are typically written in NuSMV as temporal logic formulas, and efficient symbolic algorithms (based on data structures like BDDs [17] or satisfiability-based techniques [9]) are used to traverse the model and check whether the specification holds or not.

Figure 2 shows the components of FSAP/NuSMV-SA (the solid lines represent the data flow, whereas the dotted ones represent the control flow, i.e., the dialoges which can be activated). The different blocks composing the platform are described in more detail below.

The SAT (*Safety Analysis Task*) manager is the central module of the platform. It is used to store all the information relevant to verification and safety assessment. It contains references to the system model, failure modes, location of the extended system model, safety requirements, and analyses. From the SAT manager, it is possible to call all the other components of the platform.

Model capturing: System models are written using the textual NuSMV input language. FSAP/NuSMV-SA provides users with the possibility of using their preferred text editor for editing the system model.

Failure mode editor and fault injector: These are the modules for defining failure modes and generating an extended system model, respectively.

Safety requirements editor: This is the module for entering safety requirements. Safety requirements are expressed in temporal logic [36] (either LTL or CTL) and can be defined

[5] `http://www.fltk.org.`

[6] `http://expat.sourceforge.net.`

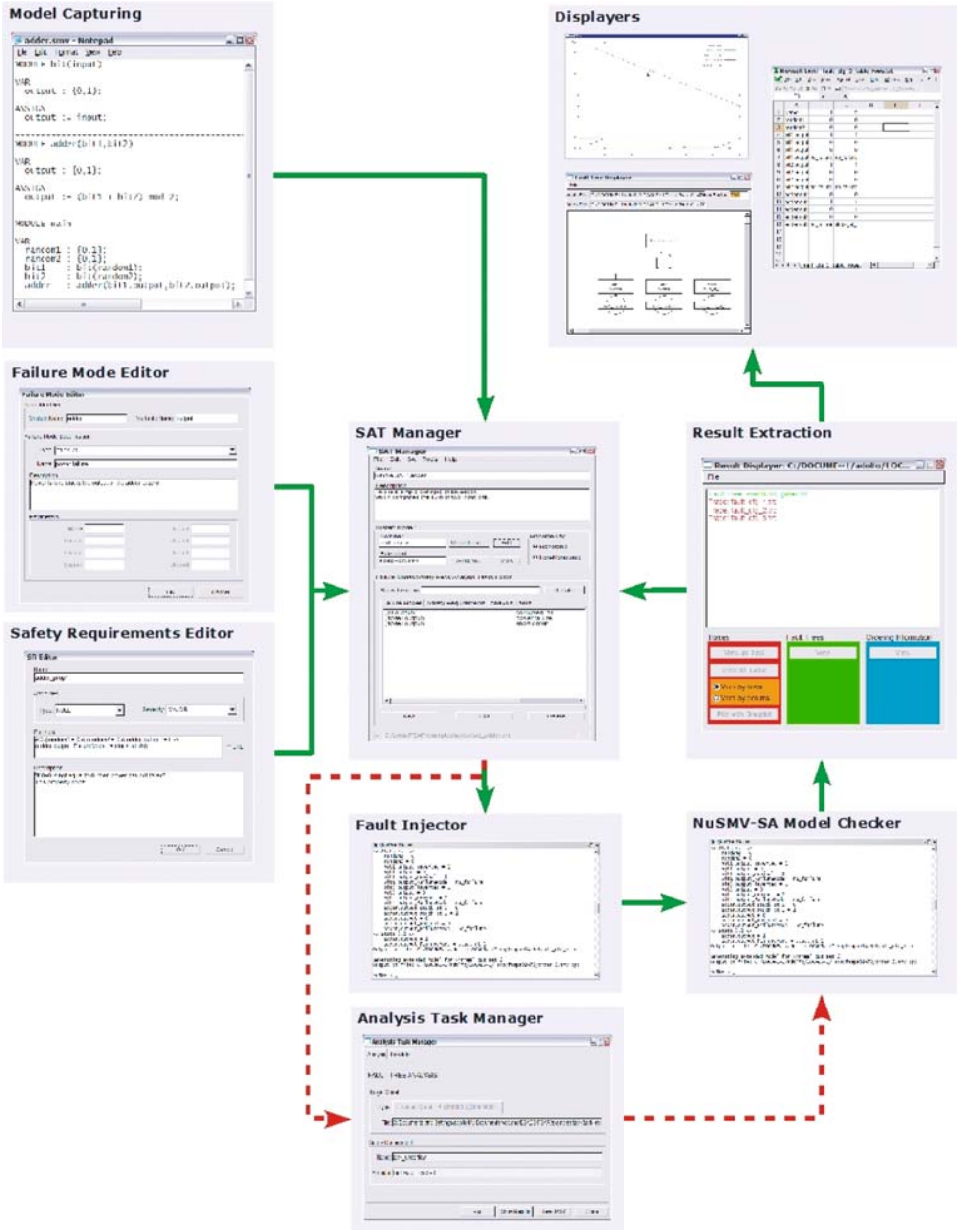


Fig. 2 The FSAP/NuSMV-SA components

by the user by choosing and instantiating patterns taken from a library of requirements.

Analysis task manager: This is the module to define analysis tasks. Analysis tasks are a convenient way to store the specification of the analyses. They are saved in the SAT manager and can be retrieved across different sessions.

NuSMV-SA: The safety analysis and verification capabilities of NuSMV-SA are based on model checking techniques. Model checking [25] is a well-established method for the formal verification of temporal properties of finite-state concurrent systems [22,39,43,46].

Result extraction and displayers: All the results produced by the platform can be viewed using the result extraction interface and the displayers. In particular, it is possible to view counterexamples in textual, structured (XML), graphical, or tabular fashion. Fault trees generated by the platform can be viewed using commercial tools (e.g., FaultTree+[7]) or using a displayer, we especially developed within the project, and can be exported into XML format.

[7] http://www.isograph-software.com.

4 Safety analysis with FSAP/NuSMV-SA

In this section, we give an overview of the FSAP/NuSMV-SA environment, and, in particular, we discuss how the various steps of the ESACS methodology, described in Sect. 2, are supported and implemented by the different architectural components of FSAP/NuSMV-SA, described in the previous section. Being an extension of NuSMV, FSAP/NuSMV-SA provides all the functionalities of NuSMV. Below, however, we will focus on the safety assessment capabilities which are specific to FSAP/NuSMV-SA. In order to describe a typical usage scenario of the platform, we will use a simple example, namely a two-bit adder. The example is deliberately simple for illustration purposes. A discussion on more realistic case studies can be found in Sect. 5.

4.1 Model capturing

The system model definition provides an executable specification (at a given level of abstraction) of the model of the system under development. In particular, in the following example, we will focus on the scenario in which the system model is built by the design engineer (see Sect. 2.1 for a discussion). Currently, the system model can be entered using a text editor (see Sect. 7 for a discussion on future improvements).

Let us consider the simple example, written in the syntax of NuSMV [22], in Fig. 3. It is composed of three modules. The `bit` module models a component which takes an input variable representing the value of a bit, and simply copies it to an output variable. The `adder` module takes in input the outputs of two bit module instances, and computes their sum (modulo two). Finally, the `main` module defines the overall system by instantiating an adder and two different bit components whose inputs are random Boolean variables.

4.2 Failure mode capturing and model extension

In order to study the behavior of the adder circuit in the presence of degraded situations, failure mode definitions can be added to the previous specification. In FSAP/NuSMV-SA, failure modes are defined using a graphical user interface, in which the safety engineer specifies which nodes of the system model can fail, in what ways, and according to what parameters. Figure 4 shows an example of the interface currently provided by FSAP/NuSMV-SA for defining failure modes. Failure modes are retrieved from a library, called GFML. The library contains the specification of the behaviours induced by the failures and the specification of the parameters (whose values must be set by the user) that characterize the failures. The standard GFML provides specification of failures like stuck-at, random output, glitch, and inverted. The library can also be extended to include user-defined failure modes.

Failure modes can be associated with variables defined inside modules of the NuSMV input model. In the adder case, for instance, failure modes may include the adder output being stuck at a given value (zero or one), and an input bit corruption (inverted failure mode). Editing of a failure mode is made easier by means of the data dictionary (see Fig. 5) which allows the user to select the variable and the corresponding NuSMV module to which the failure must be attached.

Once the failure modes have been defined, they can be automatically injected by FSAP/NuSMV-SA into the system model. The result is the so-called extended system model, i.e, a model in which some of the nodes can fail according to the specification of the failure modes. As an example, consider the *inverted* failure mode for the output of the `bit` module in Fig. 3. Injection of this failure mode produces the extension of the system model with a new piece of NuSMV code (instantiated from the GFML) that is automatically inserted into the extended system model. The new piece of code (see Fig. 6) replaces the old definition of the `output` variable of the `bit` module by taking into account a possible corruption of the input bit. Specifically, the new piece of code defines a variable `output_FailureMode`, which models the failure mode (either there is no failure, or the bit is corrupted). Depending on the value of this variable, the code defines the new output of the circuit as being the nominal one (variable `output_nominal`, which is the same as the old output) or the corrupted one (variable `output_inverted`, which is simply the negation of the nominal output). The failure is assumed to be permanent, that is, once the bit is corrupted, it remains corrupted forever.

4.3 Safety requirement capturing

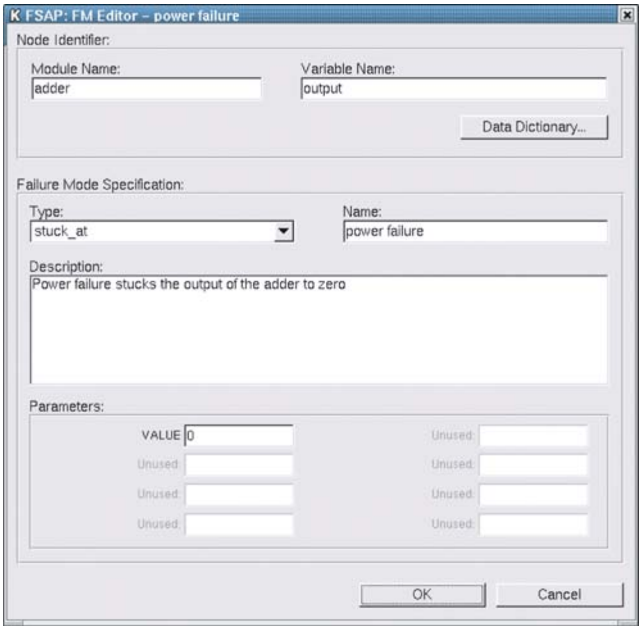
System model definition, failure mode definition and model extension are just a part of the verification and safety assessment process. Formal verification is carried out by defining properties in the form of temporal specifications. The platform supports Computation Tree Logic (CTL), Linear Temporal Logic (LTL) and real-time Computation Tree Logic (RTCTL) [36,37]. As an example, the following CTL properties may be specified for the adder example:

```
AG (random1 = 0 & random2 = 0 → adder.output = 0)

AG (random1 = 0 & random2 = 0 & adder.output != 0)
 → (bit1.output_FailureMode = inverted |
    bit2.output_FailureMode = inverted)
```

```
MODULE bit(input)                  MODULE main
VAR                                VAR
  output : {0,1};                    random1 : {0,1};
ASSIGN                               random2 : {0,1};
  output := input;                   bit1    : bit(random1);
                                     bit2    : bit(random2);
MODULE adder(bit1,bit2)              adder   : adder(bit1.output,bit2.output);
VAR
  output : {0,1};
ASSIGN
  output := (bit1 + bit2) mod 2;
```

Fig. 3 A NuSMV model for a two-bit adder



FSAP: FM Editor – power failure

Node Identifier:

Module Name: adder

Variable Name: output

Failure Mode Specification:

Type: stuck_at

Name: power failure

Description: Power failure stucks the output of the adder to zero

Parameters:

VALUE 0

Fig. 4 Input of failure modes in FSAP

The first one states that the output of the adder must be zero whenever both input bits are zero (this is clearly not the case in degraded situations), whereas the second one states that whenever the sum of two zero input bits yields one, it is the case that at least one of the two input bits is corrupted. Requirements defined in this way can subsequently be verified via the underlying model checking verification engine provided by NuSMV. Properties in FSAP/NuSMV-SA are defined via a graphical user interface, in which users can enter information such as type and severity of the safety requirement. Figure 7 shows how safety requirements are specified by the user.

The input of safety requirements is simplified by the safety pattern dialogue, which allows users to enter safety requirements by choosing and instantiating formulas from a set of predefined patterns. To start with, we introduced a set of basic patterns (see Fig. 8). This set of patterns includes basic safety and liveness temporal properties which are frequently used in verification. An extended and more structured set of patterns will be integrated in future releases of the platform (see Sect. 7 for a discussion).

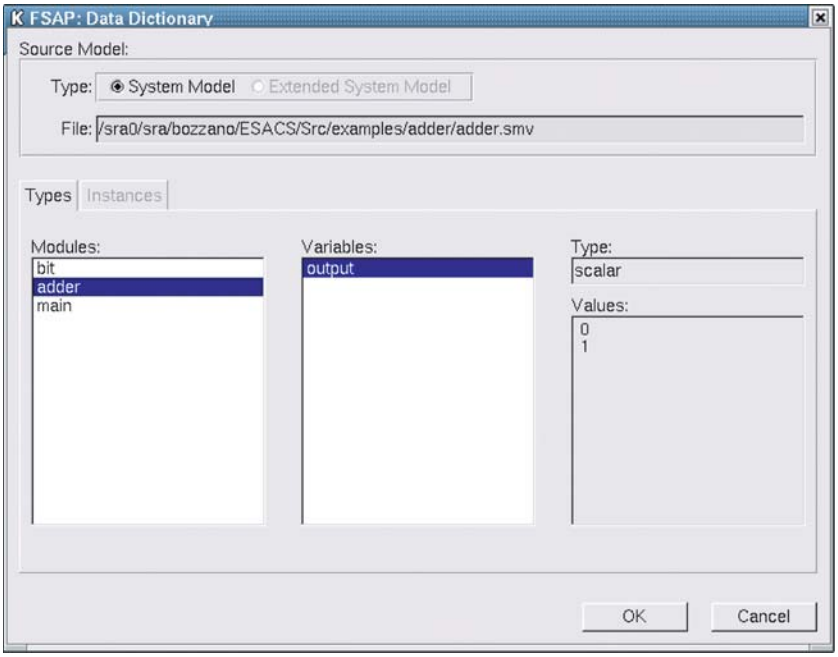
Pattern instantiation is simplified by a data dictionary (see Fig. 5) and by a "keypad" that simplifies the input of data. The safety pattern dialogue implements the concept of the GSRL library (see Sect. 2.3) of the ESACS methodology. For instance, the safety requirements described above are instances of the pattern called "Safety" in the GSRL. According to our experience, the GSRL is an important (if not essential) feature to help safety engineers, who may not be experts in temporal logic, acquire confidence using LTL/CTL notations to write safety requirements.

4.4 Model analysis

During this phase, formal verification and safety assessment of the model are performed. The model under development is tested against the safety requirements, and the results of the analysis are conveniently displayed. Using the facilities provided by the NuSMV engine, it is possible to perform constrained or random simulation, and several kinds of formal verification analyses. Every analysis task performed on the system is stored by the SAT manager, and can subsequently be accessed by the user at any time in order to view the corresponding results of the analysis. The user can also decide to re-run an analysis task from scratch, if the results are not up-to-date. The different kinds of analysis tasks supported by the platform are discussed in more detail below.

4.4.1 Simulation

The FSAP/NuSMV-SA platform allows one to perform random simulation of both the system model and the extended system model, for a given number of steps. The simulation may be guided by providing an additional set of constraints on the system execution. The result of the simulation is a trace that may be displayed either in textual format, imported in a commercial spreadsheet, or displayed in graphical form.



Source Model:
Type: ● System Model ○ Extended System Model
File: /sra0/sra/bozzano/ESACS/Src/examples/adder/adder.smv

Modules: bit, adder, main
Variables: output
Type: scalar
Values: 0 1

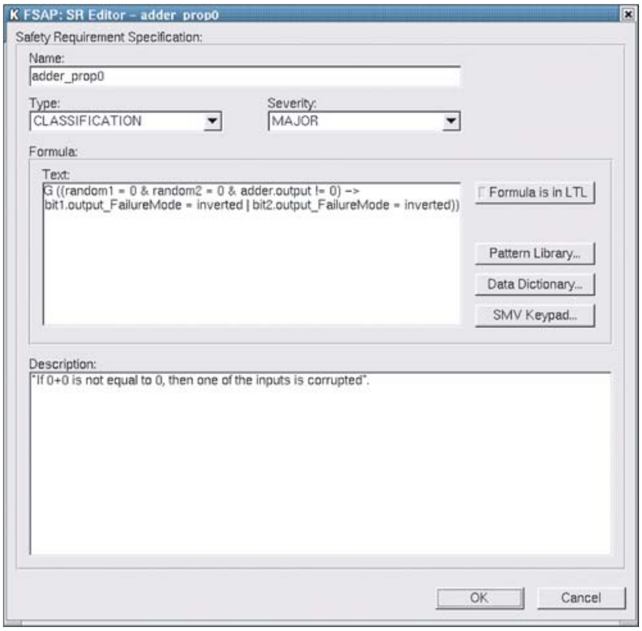
Fig. 5 Data dictionary

Fig. 6 Injecting a fault in the `bit` module

```
VAR    output_nominal     : {0,1};
       output_FailureMode : {no_failure, inverted};
ASSIGN output_nominal := input;
DEFINE output_inverted := ! output_nominal;
DEFINE output := case
         output_FailureMode = no_failure : output_nominal;
         output_FailureMode = inverted   : output_inverted;
       esac;
ASSIGN next(output_FailureMode) := case
         output_FailureMode = no_failure : {no_failure, inverted};
         output_FailureMode = inverted   : inverted;
       esac;
```

4.4.2 Property verification

Exhaustive property verification capabilities of the platform are based on the underlying model checking engine. Both CTL and LTL temporal logic formulas can be checked against the (extended) system model. In addition, each property verification task may be equipped with a set of invariants which may be added to the model in order to verify a given property under particular hypotheses. The result of a property verification task is either a system message stating that the property does hold, or a counterexample trace, which may be displayed in the same way as simulation traces. At the moment, the engine uses BDD-based model checking, but the integration of satisfiability-based model checking is ongoing, as discussed in Sect. 7.



Safety Requirement Specification:
Name: adder_prop0
Type: CLASSIFICATION
Severity: MAJOR
Formula:
Text:
G ((random1 = 0 & random2 = 0 & adder.output != 0) ->
bit1.output_FailureMode = inverted | bit2.output_FailureMode = inverted))
Description:
"If 0+0 is not equal to 0, then one of the inputs is corrupted".

Fig. 7 Input of safety requirements in FSAP

4.4.3 Fault tree construction

FTA [47,56,67] is a safety assessment strategy that complements exhaustive property verification. It is a deductive, top-down method to analyze system design and robustness. It usually involves specifying a top level event (TLE hereafter) for the analysis (e.g., a failure state), and identifying all possible sets of basic events (e.g., basic faults) that may cause that TLE to occur. FTA allows one to identify possible system reliability or safety problems and find out root causes of equipment failures. Fault trees provide a convenient symbolic representation of the combination of events resulting in the occurrence of the top event. They are usually represented in a graphical way, as a parallel or sequential combination of AND/OR gates.

The FSAP/NuSMV-SA platform can be used for the automatic generation of fault trees starting from a given model and TLE. Specifically, it is possibile to extract automatically all collections of basic events (called minimal cut sets) which can trigger the TLE. The extraction of minimal cut sets is based on procedures for computing prime implicants of Boolean functions [27,28,57,58] that internally use BDD data structures.

More in detail, the procedure for generating a fault tree works as follows. First, a forward reachability analysis of the system model is performed. This amounts to performing a fixpoint computation, starting from the set of initial states of the system. Each iteration produces a new set of states which is obtained by accumulating the states reachable using one transition step, to the current set of states. In symbols, the expression

$$\mathrm{Reach} \overset{\mathrm{def}}{=} R^*(I)$$

is computed, where I is a symbolic representation of the set of initial states, and R^* is the reflexive and transitive closure of the transition relation R. Once the reachability set is computed, it is conjoined with a symbolic representation of the states satisfying the TLE. The outcome of this computation is the set of reachable states of the system in which the TLE occurs. Finally, all the system variables except the failure variables are existentially quantified away. In symbols, the expression

$$\exists \bar{v}\ (\mathrm{Reach} \wedge \mathrm{Tle})$$

is computed, where Tle is a symbolic representation of the set of states satisfying the TLE, and $\bar{v}$ is the set of non-failure variables of the model. The resulting expression is minimized using the procedures for minimizations of Boolean functions previously mentioned. After a final step of simplification, we obtain the set of minimal cut sets. Some statistics on the total number of minimal cut sets and their order (i.e., the number of basic events composing a cut set) are also computed by FSAP/NuSMV-SA. The generated cut sets are minimal,in the sense that only failure events that are strictly necessary for the top level event to occur are retained.

Each cut set produced by FSAP/NuSMV-SA represents a situation in which the top level event has been violated owing to the occurrence of some failures. Under the hypothesis that the system model does not violate the top level event, such failures are the cause of the violation. The cut sets are a static representation of the causes of the violation. However, the starting model to which the fault tree computation algorithms are applied, is a dynamic model. As a consequence, the violation of the TLE can be due to complex interactions caused by the various failing and non-failing components of the system. The model checking algorithms (i.e., the ones for reachability analysis) take care of the dynamic part of the model, whereas the algorithms for extraction of the minimal cut sets are based on combinatorial routines.

There are two ways in which NuSMV-SA can present dynamic information that is not visible within standard fault trees. First, NuSMV-SA associates, to each cut set, a counterexample trace that shows, step by step, how the top level event is violated by the failures represented in the cut set. Second, it is possible to perform the so-called failure ordering analysis (discussed in Sect. 4.4.4), in order to investigate the possible ordering (if any) between basic events inside a minimal cut set. We refer the reader to Sect. 7 for a discussion of the ongoing work on this topic.

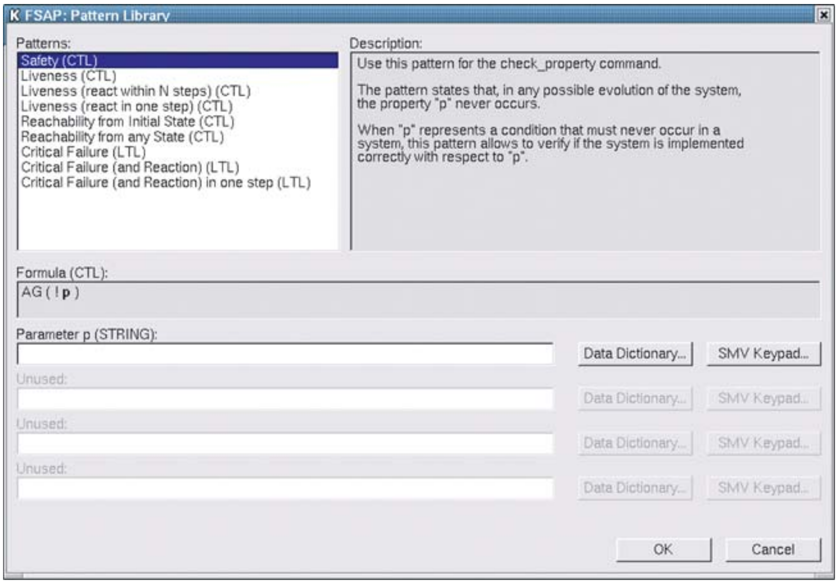
Figure 9 shows an example of fault tree computed for the adder model. It has been generated for the top level event

```
random1 = 0 & random2 = 0 & adder.output != 0
```

specifying a failure state in which both input bits are zero and the output of the adder is different from zero. The fault tree comprises three cut sets (the first one of them is a single failure, whereas the remaining two include three basic events). The fault tree states that the top level event may occur, if and only if either the output of the adder is stuck at one, or one of the input bits (and only one) is corrupted (with the adder working properly). We note that minimality of the generated cut sets implies that the case in which both input bits and the adder are failed is not considered (though causing the top level event as well).

Finally, we note that the fault tree in Fig. 9 shows an example of non-monotonic (also known as non-coherent) fault tree analysis, i.e., basic events requiring system components *not* to fail can be part of the results of the analysis. The traditional monotonic (coherent) analysis (i.e., where only failure events are considered) is also supported by FSAP/NuSMV-SA (in this case, the resulting fault tree would be the same as the one in Fig. 9, except that all events labeled as non-failure are removed). The choice between the different kinds of analyses is left to the user.

Fig. 8 Safety pattern dialogue



Patterns:

- Safety (CTL)
- Liveness (CTL)
- Liveness (react within N steps) (CTL)
- Liveness (react in one step) (CTL)
- Reachability from Initial State (CTL)
- Reachability from any State (CTL)
- Critical Failure (LTL)
- Critical Failure (and Reaction) (LTL)
- Critical Failure (and Reaction) in one step (LTL)

Description:

Use this pattern for the check_property command.

The pattern states that, in any possible evolution of the system, the property "p" never occurs.

When "p" represents a condition that must never occur in a system, this pattern allows to verify if the system is implemented correctly with respect to "p".

Formula (CTL): AG (! **p**)

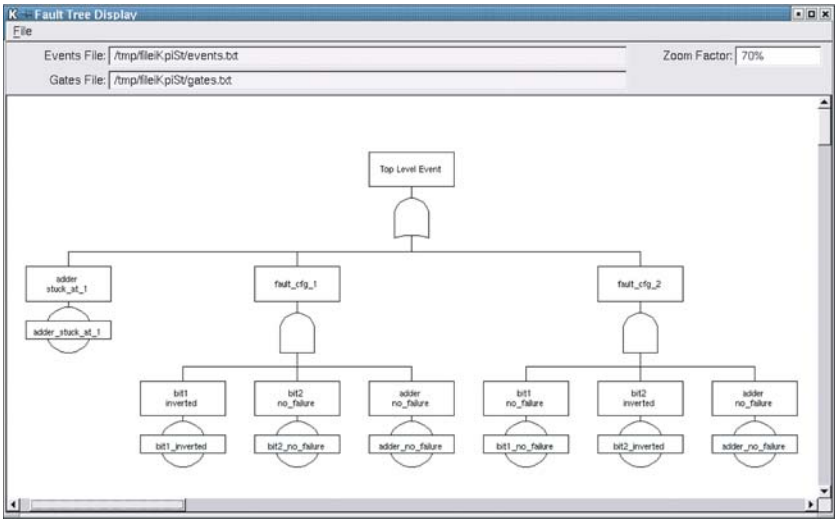
Parameter p (STRING):

Unused:

Unused:

Unused:

Fig. 9 A fault tree generated for the adder model



Events File: /tmp/fileiKpiSt/events.txt

Gates File: /tmp/fileiKpiSt/gates.txt

Zoom Factor: 70%

4.4.4 Failure ordering analysis

In this section, we briefly discuss an additional capability of the FSAP/NuSMV-SA platform, namely the so-called failure ordering analysis. The algorithm for ordering analysis is based on the same procedures for minimizations of Boolean functions [27,28,57,58] used for fault tree computation. A detailed description of the algorithm for ordering analysis is beyond the scope of this paper. We refer the reader to [15] for a description of the algorithm, its implementation and possible applications, whereas in the following we informally recall the main concepts.

In traditional FTA, cut sets are simply flat collections (i.e, conjunctions) of events which can trigger a given TLE. However, there might be timing constraints enforcing a particular event to happen before or after another one, in order for the TLE to be triggered (i.e., the TLE would not show if the order of the two events were swapped). Ordering constraints can be due to a causality relation or a functional dependency between events, e.g., or caused by more complex interactions involving the dynamics of a system. Whatever the reason, failure ordering analysis can provide useful information which can be used by the design and safety engineers to fully understand the ultimate causes of a given system malfunction, so that adequate countermeasures can be taken.

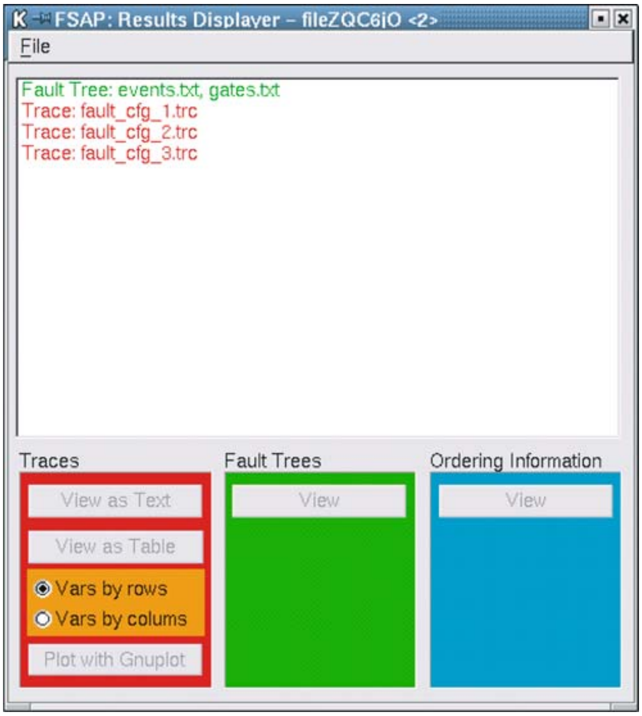
The ordering analysis phase can be tightly integrated with fault tree analysis, as described below. Given a system model, the verification process consists of the following phases. First of all, a top level event to analyze is chosen (clearly, the analysis can be repeated for different top level events). Then, fault tree analysis is run in order to compute the minimal cut sets relative to the top level event. For each cut set, the ordering analysis module of the platform generates a so-called ordering information model and performs ordering analysis on it.

The outcome of the ordering analysis is currently displayed as a precedence graph, which shows the order among events (if any) which must be fulfiled in order for the top level event to occur. In the future, we plan to integrate the ordering analysis into the fault tree generation routines and use a uniform notation based on dynamic fault trees (see Sect. 7 for more details).

4.5 Result extraction and analysis

During the result extraction analysis phase the results of the analyses are displayed in a human-readable form and using standard notations used by safety engineers. They are produced using formats that are compatible with traditional commercial tools (e.g., fault tree analysis tools used by safety engineers).

Every trace (either obtained as a result of a simulation task or as a counterexample trace bound to the verification of a system property or to a minimal cut set) can be displayed in textual, structured (XML), graphical (using the gnuplot utility), or in tabular fashion (the trace can be imported into commercial spreadsheets). Fault trees generated by the platform can be viewed using commercial tools (FaultTree+[7]) or using a displayer we especially developed within the project (see an example in Fig. 9) and can be exported into XML format. The result displayer (see Fig. 10) allows the user to select the results of an analysis task and display them in the desired form.



Fault Tree: events.txt, gates.txt
Trace: fault_cfg_1.trc
Trace: fault_cfg_2.trc
Trace: fault_cfg_3.trc

● Vars by rows
○ Vars by colums

Fig. 10 The result displayer

5 Industrial application of FSAP/NuSMV-SA

Within the ESACS project, FSAP/NuSMV-SA has been tested on two industrial real-world case studies, chosen by the industrial partners, to test the applicability of the techniques described above in an industrial environment. Both case studies share the following characteristics:

- They are of industrial interest.
- They are heterogeneous systems comprising various types of components like electromechanical components (e.g., control valves, relays), mechanical components (e.g., shafts, gearboxes, freewheels), electronic transducers (e.g., speed sensors, pressure sensors), and electronic controllers.
- They were considered of the right level of complexity (medium/high) for the project purposes.

The first case study is derived from work by ONERA[8] on the Hydraulic Boolean System of the Airbus A320 [8]. The model of the system, originally written in the Altarica language [4], has been automatically translated into the NuSMV language and verified, for the purposes of evaluating the FSAP/NuSMV-SA capabilities, with the FSAP/NuSMV-SA platform. The results obtained were in accordance with the ones obtained by ONERA.

The second case study is the Secondary Power System, described in [14]. It consists of a subsystem that drives the hydraulic and electrical utilities of an aircraft. It is a safety-critical system, in that it must prevent any power loss of the utilities, even in presence of failures. To this aim, the default lines driving the utilities are coupled with auxiliary lines. Two computers are responsible for carrying out a recovery procedure consisting of driving the utilities using the auxiliary lines, in case of failure of some component in the default lines.

In the rest of this section, we will focus on the Secondary Power System case study, whereas we refer the reader to [8] for more details on the other case study, studied by ONERA. Modeling and testing of the case study have been conducted as a joint collaboration between ITC-IRST, Alenia Aeronautica (the owner of the case study), and Società Italiana Avionica (SIA). Alenia Aeronautica is the leading

[8] `http://www.cert.fr.`

Italian company in the avionics field, and is among the major industrial complexes in the world. Additional details on the case study can be found in [14], a joint paper written with people from Alenia and SIA. However, we note that, given the nature of the case study and the non-disclosure agreement with the industrial partner, it is not possible to provide any further details regarding how the system works and its actual implementation in NuSMV.

As a general comment, we would like to remark that the collaboration with Alenia/SIA was also intended as a way to investigate and assess the use of techniques based on formal methods in the industry. Although the introduction of these techniques in the actual development cycle was not a realistic goal for the ESACS project, we encouraged the use of the platform in a way as similar as possible to the industrial practice. To this aim, we invited some people from Alenia/SIA to "play" the role of the design or safety engineers, and we had them use the platform in a way as close as possible to the actual work practice. As discussed in Sect. 7, ITC-IRST is currently taking part in a further EU-funded project, called ISAAC[9] (basically a continuation of the ESACS project, involving a superset of the partners of ESACS), hence the aspects related to the impact on the actual work practice will be further pushed and investigated in ISAAC.

The rest of this section is structured as follows. In Sect. 5.1, we illustrate some issues related to modeling and to the interaction with the industrial partners. In Sect. 5.2 we discuss some issues related to verification of the model and present some experimental results. Finally, in Sect. 5.3 we sketch the resulting set of "lessons learned".

5.1 Modeling

The modeling activity was an iterative (and interactive) process, involving people from ITC-IRST and people from Alenia/SIA. Alenia provided the written documentation related to the case study. ITC-IRST was responsible for the actual modeling in the NuSMV language, whereas some people from Alenia/SIA, after an initial training period, were invited to use the platform for checking the models. The role of the people from Alenia/SIA was to provide feedback on the results obtained from the models, and also to use the case study to assess the platform and provide a wish list of possible improvements and extensions.

The way we conceived the modeling activity was by producing a set of models, described at different levels of detail, from the less to the more detailed. As a result, the overall set of models form a hierarchy in which each model has a higher level of detail and complexity with respect to the previous ones. The advantage of using a hierarchy of models is twofold. First, it can ease the modeling process: we model the interaction between the different components, and then we model the actual behavior of each component in an increasingly realistic way. Second, it can be seen as a way to cope with the model scalability issues, that is, the inability of the model checker to analyze the more detailed models.

The set of models we developed can be summarized as follows:

- *Model-1:* this is the simplest model and is a sort of block diagram. It is also representative of the first specification that the safety engineer receives from the design engineer in the standard development process. In our case, this model included both the left and right hand side of the system and a simplified model of the computers. The variables used in the model are all Boolean and the components were blocks which may be either working or not working. This model was used to perform a so-called *block reliability* analysis of the case study, in order to analyze the functional interactions between the components;
- *Model-2:* in this model, the behaviour of the components was modeled in a more realistic manner, although the computers were still simplified. The model included only one side of the system (we exploited the symmetry of the system to reason on the other side). The variables representing physical quantities were discretized, i.e., encoded by means of integer variables in a suitable range;
- *Model-3:* this model is the same as "model-2", except that the computers are modeled in a realistic manner;
- *Model-4:* this is the same as "model-3", with a more realistic modeling of the mechanical components. In particular, we modeled mechanical forces that must be propagated in *reverse* (e.g., in case of an engine failure, the fault can affect components that are further up in the functional chain);
- *Model-5:* this model is the same as "model-2", except that both sides of the system are modeled;
- *Model-6:* this model is the same as "model-4", except that both sides of the system are modeled.

5.2 Verification

In this section, we discuss some issues related to the verification of the models outlined in the previous section and we present some experimental results.

The models of Sect. 5.1 have been enriched automatically with the definition of the relevant failure modes. Typically, failure modes were attached to each component of the model. The definition of the failure modes was retrieved from the GFML library available in FSAP. For instance, the list of failure modes included stuck-at failures(for valves)

[9] `http://www.cert.fr/isaac.`

Fig. 11 Experimental results

MODEL	COMP		FT		PROP1		PROP2	
	TIME	MEM	TIME	MEM	TIME	MEM	TIME	MEM
model-1	< 0.01	1	< 0.01	1	< 0.01	1	< 0.01	1
model-2	8	10	107	13	6	10	6	10
model-3	119	20	205	25	177576	28	174052	28
model-4	1356	63	6	63	1716	109	1695	109
model-5	760	52	-	↑	21603	52	21787	52
model-6	637	29	↑	-	↑	-	↑	-

and ramp-down failures to model the speed of mechanical components (the speed can decrease down to a given value because of a fault).

The safety analysis tasks consisted of the generation of two fault trees for different propositional properties, and in the verification of two safety properties, expressed in CTL temporal logic. The purpose of the analyses was to verify whether the recovery procedure was carried out correctly.

In Fig. 11, we report some experimental results. For fairness, we ran each experiment (fault tree generation or property verification) with a different invocation of the model checker (the execution of a verification task may in fact benefit from the successful completion of a previous task). For each of the models of Sect. 5.1, we report the time (in seconds) and memory (in Mb) used for compiling the model (column COMP), generating one of the two fault trees (column FT) and verifying the two safety properties (columns PROP1 and PROP2). The memory used for completing a verification task includes the memory used for compilation. We do not report the results on the second fault tree, given that they marginally differ from the results obtained for the first one.

The experiments have been run on a two-processor machine equipped with Intel Xeon 3.00 GHz, with 4 Gb of memory, running Linux RedHat 7.1 (only one processor was allowed to run for each experiment). The time limit was set to 72 h and the memory limit to 1 Gb. In Fig. 11, a '↑' in the time or memory columns stands for a time-out or memory-out, respectively. We ran NuSMV-SA version 2.3.0, with the following command-line options: `-reorder -dynamic -thresh 500000 -coi`. That is, we enabled dynamic variable re-ordering, cone-of-influence reduction, and threshold conjunctive partitioning.

We ran some further experiments after enabling the computation of reachable states (option `-f` on the NuSMV-SA command-line) for the verification of safety properties (the computation of reachable states is used by default for the generation of fault trees). The results are given in Fig. 12. Although the results are not always consistent, it is evident that enabling the computation of reachable states may in some cases significantly improve the performance (e.g., in the case of "model-3"). As a drawback, the computation of the reachable states may sometimes be unfeasible due to memory exhaustion (e.g, for "model-5"). The outcome of this set of experiments suggests that there is room for improving the performance of the model checker with proper tuning.

MODEL	PROP1		PROP2	
	TIME	MEM	TIME	MEM
model-1	< 0.01	1	< 0.01	1
model-2	149	18	148	18
model-3	944	45	940	45
model-4	1620	109	1598	109
model-5	-	↑	-	↑
model-6	↑	-	↑	-

Fig. 12 Additional experimental results, with computation of reachable states enabled

5.3 Results and experience

In this section, we discuss some results and experiences arising from the modeling activity and the interaction with the people from Alenia/SIA. For each case study, the various models were tested in order to highlight pros and cons of our approach and to devise possible methodology and tool platform improvements. The main criteria of the evaluation were the effectiveness of the methodology to improve the integration of the design and safety activities on the system, and the effectiveness of the tool in the implementation of the different steps defined by the methodology. In the following, we briefly summarize the results of our evaluation.

Representational issues: One interesting aspect of the case studies concerned the modeling of the various (sub)-components. In particular, one of the most challenging modeling issues has been the modeling of hydraulic and mechanical components. For such systems, in fact, when reasoning about degraded situations, the standard input/output modeling with functional blocks may be particularly difficult. For instance, a leakage in a pipe may cause loss of pressure in the whole pipe. As a second example, in certain situations (e.g., in case of an engine failure) mechanical forces have to be propagated in *reverse*, therefore affecting functional blocks that are further up in the functional chain. Particular care had to be taken to address these issues. More generally, we think that the use of hybrid system modeling tools may be very effective for such kind of models (see Sect. 7).

Integration of the design and safety activities: We experienced that the ESACS approach effectively improves and encourages the interaction between design and safety engineers as they, for instance, can speak the same unambiguous language and share the same formal system model. In addition, the safety evaluation of the system architecture could be performed in the very early phases of system design, by simulating and proving properties of the system model.

Failure mode definition and injection: The facility for failure mode injection and system model extension experimented during the different test cycles of the platform worked well, but it is based on a library of generic failure modes specifically created for the ESACS purpose. As a consequence, the library needs to be enriched in the future to include a taxonomy of the failure modes typical of the main kinds of components (electronic, electric, mechanical, pneumatic components and so on), tailored to the specific industrial needs.

System property definition: The ESACS approach enables the definition of different types of verification tasks on system models, like reachability of a given state (e.g., component failure) or the fulfillment of a given condition (e.g., output from one component being a certain percentage under its nominal value). The system properties can be written in LTL or CTL temporal logic. Such formalisms may be difficult to understand, especially by people who are not expert in formal verification. The possibility of defining system properties by instantiating a class of general-purpose safety patterns was included in the platform as a response to the industrial partners needs, and was particularly appreciated.

System property verification: Performing model checking of functional requirements on the system model often leads to the state explosion problem. As a consequence, it was sometimes necessary to scale the model down, for instance by abstracting away certain characteristics of the design model. As previously explained, one way was to cut the model by exploiting inherent symmetries in it. However, this approach was not feasible for the most detailed models.

In order to mitigate the problem of state explosion, the definition of a set of different models was particularly helpful, as it has been possible to define specialised models for certain properties. On the other hand, at the moment we are working on improvements for the model checking tools which will hopefully be able to ameliorate this problem (we discuss this point further in Sect. 7).

Finally, from an industrial point of view, the possibility of using simulation and exhaustive techniques to drive the system into a particular state was particularly useful, for instance, in order to show that a safety-critical state cannot be reachable when failure modes are disabled.

Fault tree generation: Automatic generation of fault trees has been possible with satisfactory results only for the less complex models, whereas with the more complex ones we encountered difficulties due to the state explosion problem. Nonetheless, the safety engineers judged the fault trees generated for the less complex models to be informative enough.

6 Related work

The ESACS methodology: The FSAP/NuSMV-SA platform has been developed within the ESACS[3] project (Enhanced Safety Assessment for Complex Systems), a European-Union-sponsored project involving various research centers and industries from the avionics sector. For a more detailed description of the project goals and the industrial case studies which have been investigated we refer the reader to [14,16]. Within the project, the ESACS methodology has been also implemented in other platforms. We mention [8] for the platform based on Altarica [4]; [1,30] for the platform based on SCADE[10]; [55] for the platform based on STATEMATE[11].

The works recently presented in [44,45] have a close similarity with the present work and, more in general, with the ESACS methodology. In particular, the idea of integrating the traditional development activities with the safety analysis activities, based on a formal model of the system, and the idea of clearly separating the nominal model from the fault model, using an automatic extension facility for merging them, are ideas that have been pioneered by ESACS [16]. The authors call this approach model-based safety analysis and present a proposal for integrating it into the traditional "V" safety assessment process. The approach is exemplified on a case study modeled and analyzed using SCADE[10] and Simulink [29]. We also mention [53,65,66], which discuss the specification and validation of a Flight Guidance System and a Flight Management System. This work shares with us and the ESACS project the application field (i.e., avionics), and the use of NuSMV as a target verification language (the paper also considers the PVS theorem prover as alternative verification tool). An automatic translator from a specification language called RSML to NuSMV is also provided.

Algorithms: The safety analysis capabilities provided by the platform include traditional fault tree generation [47, 56,67] together with formal verification capabilities typical of model checking [22,25,39,43,46,52]. The algorithms for cut set and prime implicant computation described in Sect. 4.4.3 are based on classical procedures for minimization

[10] `http://www.esterel-technologies.com.`

[11] `http://www.ilogix.com.`

of Boolean functions, specifically on the implicit-search procedure described in [27,28,57,58], which is based on Binary Decision Diagrams (BDDs) [17]. This choice was quite natural, given that the NuSMV model checker makes a pervasive use of BDD data structures. The ordering analysis procedure described in Sect. 4.4.4 also makes use of these algorithms (we refer the reader to [15] for a full description of the procedure and of the related literature). Explicit-search and satisfiability-based techniques for computation of prime implicants are described for instance in [50].

Fault tree construction: Our work shares some similarities with the work in [59,64], which are both concerned with automatically proving the consistency of fault trees using model checking techniques. The paper [64] presents a fault tree semantics based on Clocked CTL (CCTL) and uses timed automata for system specification, whereas [59] presents a fault tree semantics based on the Duration Calculus with Liveness (DCL) and uses Phase Automata as an operational model. The focus of both papers is on using model checking to validate manually constructed fault trees (e.g., to detect incompleteness of a fault tree due to an omitted cause for an hazard). On the contrary, our approach is concerned with using model checking for the automatic generation of a fault tree starting from a formal specification of the system model.

Probabilistic safety assessment and dynamic fault trees: A large amount of work has been done in the area of probabilistic safety assessment (PSA) and in particular on dynamic reliability [60]. Dynamic reliability is concerned with extending the classical event or fault tree approaches to PSA by taking into consideration the mutual interactions between the hardware components of a plant and the physical evolution of its process variables [51]. Examples of scenarios taken into consideration are, e.g., human intervention, expert judgment, the role of control/protection systems, the so-called failures on demand (i.e., failure of a component to intervene), and also the ordering of events during accident propagation. Different approaches to dynamic reliability include, e.g., state transitions or Markov models [3,54], the dynamic event tree methodology and the TRETA package of [26], and direct simulation via Monte Carlo analysis [51,61]. Our approach, which is concerned with the automatic generation of fault trees, differs from the works cited above that are mostly concerned with the evaluation of a given fault tree. Furthermore, we use model checking to support automatic verification of arbitrary CTL and LTL properties (in particular, both safety and liveness properties).

Concerning ordering analysis, the work which is probably closer to ours is [26], which describes dynamic event trees as a convenient means to represent the timingand order of intervention of sub-systems and their eventual failures. Our approach can support simultaneous failures, whereas, at the moment, we are working under the hypothesis of persistent failures (i.e., no repair is possible).

Concerning fault tree evaluation, we mention DIFTree (Dynamic Innovative Fault Tree) [49], a methodology for the analysis of dynamic fault trees. It is implemented in the Galileo tool [62]. The methodology is able to identify independent sub-trees, translate them into suitable models, analyze them and integrate the results of the evaluation. Different techniques can be used for the evaluation, e.g., BDD-based techniques for the evaluation of static fault trees, and Markov techniques or Monte Carlo simulation for dynamic fault trees. The DIFTree methodology also includes techniques to model coverage, i.e., the probability that a system can automatically recover from a fault, given that a failure occurs. Techniques for incorporating coverage modeling [33] into a BDD-based fault tree solution have been studied in [32].

Tools: The Galileo[12] tool [62], already mentioned, is a tool for modeling and analysis of fault trees. It allows the user to edit a fault tree in a textual or graphical format, and to evaluate the fault tree using different techniques. In addition, it supports different probability distributions for component failures. Both the support for probability distributions and the notation for dynamic gates used in Galileo [48] are features that we would like to integrate into the FSAP/NuSMV-SA platform (we refer the reader to Sect. 7 for more details).

Finally, we mention the SMART[13] tool [19]. SMART is a software package integrating various modeling formalisms (e.g., stochastic Petri nets) into a single environment. The analysis of the models can be performed using a variety of evaluation techniques, ranging from CTL-based symbolic model checking, for the computation of the state space and for solving temporal logic queries, to numerical methods and simulation, for performance and reliability measures. The implementation of CTL-based model checking techniques relies on multi-valued decision diagrams (MDDs) to store the state space, and makes use of advanced techniques based on Kronecker encoding for the next state function and an efficient saturation algorithm. These techniques have been shown to be very effective, both in terms of time and space, for model checking asynchronous systems (with loosely connected components) [20]. Although the models used in ES-ACS are typically synchronous systems, it could be worth evaluating the performance of the SMART tool on them.

[12] http://www.cs.virginia.edu/∼ftree.

[13] http://www.cs.ucr.edu/∼ciardo/SMART.

7 Future work

At the moment, the platform is undergoing further development, both at the level of the implementation and at the level of provided functionalities. This further development is being carried out as contribution to the ISAAC[9] project (Improvement of Safety Activities on Aeronautical Complex systems), in which ITC-IRST is involved (the set of partners involved in ISAAC is a superset of the ESACS partners). The goals of ISAAC are, on the one hand, the investigation of some new thematic areas, and on the other hand, the consolidation and the further push of the ESACS experience and methodology into the industrial practice [2]. In particular, future extensions and improvements of the FSAP/NuSMV-SA platform will be based on the feedback from the industrial partners. A list of improvements and extensions to be addressed are discussed below.

Failure modes: At the moment, all failures are assumed to be *permanent*, that is, once a component fails, it remains failed forever (once failed, always failed). In the future, we want to overcome this limitation and be able to deal, for instance, with *transient* failures. This extension may have an impact on the failure mode definition, the analysis routines, and also the result presentation aspects.

Furthermore, we are currently working on extending the failure model with common cause failures. By common cause failures we mean either simultaneous or cascading failures which are due to a common cause, and therefore are not independent. Future releases of the platform will enable the possibility to declare failure sets grouping common cause failures and to generate the corresponding fault tree in a suitable format. Common cause analysis is one of the topics of the ISAAC project.

Furthermore, we plan to enrich the GFML. First, we want to include a richer taxonomy of failure modes (see discussion in Sect. 5.3), tailored to the industrial needs. Second, we would also like to support *probabilistic* evaluation of fault trees (see paragraph on quantitative analysis below). One possibility to achieve this goal would be to create a component library associating stochastic failure distributions to components, together with failure modes.

Failure ordering analysis: Concerning failure ordering analysis, we want to improve the current notation of the outcome of the analysis, which is presented in the form of precedence graphs. We would like to use notations for dynamic gates and integrate the results of ordering analysis into the displayed fault tree. In this way, a uniform notation would be used throughout the platform. An example of notation for dynamic gates and their formal semantics is the one used in the Galileo tool (see [48]).

Hierarchical fault trees: At the moment, the fault trees generated by our platform are *flat*, i.e., they simply collect the cut sets (or prime implicants). This might be a concern, especially when the number of computed cut sets is large (for instance, in the Hydraulic Boolean System case study, mentioned in Sect. 5, the prime implicants for one of the safety requirements are more than 200). In order to improve readability, we are investigating techniques for restructuring the fault tree. By restructuring, we mean the possibility to group common parts of the tree by introducing intermediate levels (gates). We call these trees hierarchical fault trees. Different techniques will be considered, for instance user-guided restructuring or automatic restructuring based on logical equivalences.

Safety patterns: Regarding the safety patterns described in Sect. 4.3, we plan to include a more comprehensive and structured set of patterns in future releases of the platform. In particular, we are considering the structure described in [35][14]). The intent is to support a comprehensive set of patterns that occur frequently in the specification of concurrent and reactive systems.

Quantitative analysis: We plan to extend our framework to deal with *probabilistic* assessment. Although not illustrated in this paper, associating fixed probabilistic estimates to basic events and evaluating the resulting fault tree is straightforward (see, e.g., [28]). Evaluation of probabilities also allows one to exclude minimal cut sets below a certain probability threshold, if desired. However, more work needs to be done in order to support more complex probabilistic dynamics (see, e.g., [31]). A possibility would be to include sample probabilistic distributions like the ones used in DIFTree [49] (see also the paragraph on failures modes above). Concerning common cause failures (see again the paragraph on failures modes) techniques for evaluating the fault tree using BDD-based methods have been investigated in [63].

Usability/expressiveness: An important issue which is related to usability of the platform is the enhancement of the input language. We plan to design and implement a graphical input language, which could serve as an interface between the underlying, textual, NuSMV input language, and the final user. This feature is considered of primary importance by the industrial partners.

Second, as mentioned in Sect. 3, the input language supported by NuSMV will be enriched in the near future, by including the *real* data type as a primitive type. This extension will increase the level of expressiveness of the language, and will enable a more faithful modeling of the dynamics of

[14] See also `http://patterns.projects.cis.ksu.edu.`

a complex system. Namely, it would be possible to model the physical quantities of a given system using real functions (e.g., to model the mechanical forces, as mentioned in Sect. 5.3). Furthermore, we might have a more realistic modeling of time, which, at the moment, is modeled by an abstract transition step. One possibility we are considering is to model systems as hybrid automata [40,41]. An alternative possibility is to use the model of time based on calendar automata described in [34].

Given that having real variables makes the resulting models infinite state, a way is needed to deal with this additional complexity. Specifically, we are planning to use satisfiability-based techniques. In our experience, these techniques can also help relieve the state explosion problem. This is explained in more detail in the next paragraph.

Efficiency: As discussed in Sect. 5, the state explosion problem may prevent the completion of the analysis tasks on the most complex models. In order to alleviate this problem, we are currently integrating the possibility of using satisfiability-based techniques [9] inside the platform. The possibility of using a satisfiability engine for property verification is already supported by the NuSMV model checker, whereas the satisfiability-based algorithms for generating fault trees are currently under development. Satisfiability-based techniques are useful for exploring an initial segment of the state space. BDD-based techniques work by saturating sets of states, as opposed to satisfiability-based techniques that are tipically used to find single traces of bounded lenght. Further traces can be found after ruling out the already discovered ones, by properly modifying the temporal formula to be verified. Typically, satisfiability-based techniques are very effective for incomplete verification (bug hunting). However, the use of induction techniques can make their use competitive also for exhaustive verification. In the future, we also plan to combine BDD-based and satisfiability-based techniques for computing fault trees. The idea is to use satisfiability-based techniques to find a subset of the minimal cut sets, use them to simplify the model, and finally use BDD-based techniques to find the remaining cut sets.

An additional way to alleviate the state-explosion problem is to modify the system model. According to our experience (see Sect. 5.3) the state explosion problem was mainly due to the use of discretized integer variables, which we used to implement the dynamics of physical variables. An alternative to discretization is to model physical quantities directly as real functions, and model systems as hybrid automata. This is not only a way to increase the expressiveness of the model (as explained in the previous paragraph), but it is also a way to deal with the state explosion problem. In fact, verification of systems modeled as hybrid automata can be efficiently performed using decision procedures for combinations of Boolean and mathematical reasoning. In particular, we plan to experiment with MATHSAT [5,7,10–13]. MATHSAT is a satisfiability-based decision procedure, developed at ITC-IRST, which is able to combine Boolean reasoning with reasoning on more complex theories like (integer or real) linear arithmetic, equality and uninterpreted functions, and their combinations. The integration of MATHSAT into NuSMV is part of the current tool development plan. Preliminary results of the application of MATHSAT to the verification of hybrid systems (see [6]) suggest that this approach is promising.

8 Conclusions

In this paper, we have presented the FSAP/NuSMV-SA safety analysis platform. The verification engine of the platform is based on the NuSMV model checker [22]. FSAP/NuSMV-SA can be used as a tool to assist the safety analysis process from the early phases of system design to the safety assessment phase. It provides a uniform environment that can be used both by design engineers for the formal verification of a system and by safety engineers to automate certain phases of safety assessment. The major benefits are a tight integration between the design and the safety teams, and the mechanization of (some of) the activities related to verification and safety analysis.

The main functionalities provided by FSAP/NuSMV-SA include model construction facilities (e.g., automatic failure injection based on a library of predefined failure modes), exhaustive property verification capabilities typical of model checking, and automatic fault tree generation. Fault tree generation can be performed both in the case of monotonic systems (computation of minimal cut sets) and in the case of non-monotonic ones (computation of prime implicants). Furthermore, the results provided by fault tree generation can be conveniently integrated by the so-called failure ordering analysis that allows the user to extract ordering constraints which hold between basic events in a given cut set.

At the moment, the platform is undergoing further development as a contribution to the ISAAC project. The industrial evaluation of the platform will be carried on, in collaboration with Alenia Aeronautica and SIA. A particular emphasis will be put on aspects related to the usability for people that are not expert in formal verification, and to the introduction of the tool in the actual work practice.

The FSAP/NuSMV-SA platform is available for evaluation from `http://sra.itc.it/tools/FSAP`. The download is currently password protected; the password can be obtained by contacting the authors.

Acknowledgements The work presented in this paper would have not been possible without the help of Paolo Traverso, Alessandro Cimatti, and Gabriele Zacco. We would also like to thank the anonymous reviewers for their helpful and pertinent comments, and Charles Jochim for a careful reading of a preliminary draft of this paper. Finally, we would like to thank the following people working in the ESACS project and, in particular: Ove Åkerlund (Prover), Pierre Bieber (ONERA), Christian Bougnol (AIRBUS-F), E. Böde (OFFIS), Matthias Bretschneider (AIRBUS-D), Antonella Cavallo (Alenia Aeronautica), Charles Castel (ONERA), Massimo Cifaldi (SIA), Alain Griffault (LaBri, Université de Bordeaux), C. Kehren (ONERA), Benita Lawrence (AIRBUS-UK), Andreas Lüdtke (OFFIS), Silvayn Metge (AIRBUS-F), Chris Papadopoulos (AIRBUS-UK), Renata Passarello (SIA), Thomas Peikenkamp (OFFIS), Per Persson (Saab), Christel Seguin (ONERA), Luigi Trotta (Alenia Aeronautica), and Laura Valacca (SIA).

References

1. Abdulla, P.A., Deneux, J., Stålmarck, G., Ågren, H., Åkerlund, O.: Designing safe, reliable systems using scade. In: Symposium on Leveraging Applications of Formal Methods ISoLA 2004 (2004)
2. Åkerlund, O., Bieber, P., Böede, E., Bozzano, M., Bretschneider, M., Castel, C., Cavallo, A., Cifaldi, M., Gauthier, J., Griffault, A., Lisagor, O., Lüdtke, A., Metge, S., Papadopoulos, C., Peikenkamp, T., Sagaspe, L., Seguin, C., Trivedi, H., Valacca, L.: ISAAC, a framework for integrated safety analysis of functional, geometrical and human aspects. In: Proceedings of the European Congress on Embedded Real Time Software (ERTS 2006) (2006)
3. Aldemir, T.: Computer-assisted Markov Failure Modeling of Process Control Systems. IEEE Trans. Reliab. **R-36**, 133–144 (1987)
4. Arnold, A., Griffault, A., Point, G., Rauzy, A.: The AltaRica formalism for describing concurrent systems. Fundam. Inform. **40**, 109–124 (2000)
5. Audemard, G., Bertoli, P., Cimatti, A., Korniłowicz, A., Sebastiani, R.: A SAT based approach for solving formulas over boolean and linear mathematical propositions. In: Voronkov, A. (ed.) Proceedings Conference on Automated Deduction (CADE-18), vol. 2392 of LNAI, pp. 195–210. Springer, Berlin Heidelberg New York (2002)
6. Audemard, G., Bozzano, M., Cimatti, A., Sebastiani, R.: Verifying Industrial Hybrid Systems with MathSAT. Electron. Notes Theor. Comp. Sci. **119**(2), 17–32 (2005)
7. Audemard, G., Cimatti, A., Korniłowicz, A., Sebastiani, R.: Bounded Model Checking for Timed Systems. In: Peled, D.A., Vardi, M.Y. (eds.) Proceedings Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2002), vol. 2529 of LNCS, pp. 243–259. Springer, Berlin Heidelberg New York (2002)
8. Bieber, P., Castel, C., Seguin, C.: Combination of fault tree analysis and model checking for safety assessment of complex system. In: Grandoni, F., Thévenod-Fosse, P. (eds.) Proceedings of the European Dependable Computing Conference (EDCC-4), vol. 2485 LNCS, pp. 19–31. Springer, Berlin Heidelberg New York (2002)
9. Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.: Symbolic Model Checking without BDDs. In: Cleaveland, R. (ed.) Proceedings Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS 1999), vol. 1579 of LNCS, pp. 193–207. Springer, Berlin Heidelberg New York (1999)
10. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., Ranise, S., van Rossum, P., Sebastiani, R.: Efficient satisfiability modulo theories via delayed theory combination. In: Etessami, K., Rajamani, S.K. (eds.) Proceedings Conference on Computer Aided Verification (CAV 2005), vol. 3576 of LNCS, pp. 335–349. Springer, Berlin Heidelberg New York (2005)
11. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., Ranise, S., van Rossum, P., Sebastiani, R.: Efficient theory combination via boolean search. In: Information and Computation, Special Issue on Combining Logical Systems (2006) (in press)
12. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Schulz, S., Sebastiani, R.: An incremental and Layered Procedure for the satisfiability of linear arithmetic logic. In: Halbwachs, N., Zuck, L.D. (eds.) Proceedings Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005), vol. 3440 of LNCS, pp. 317–333. Springer, Berlin Heidelberg New York (2005)
13. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Schulz, S., Sebastiani, R.: Mathsat: tight integration of SAT and mathematical decision procedures. J. Autom. Reasoning, Special Issue on SAT (2006) (in press)
14. Bozzano, M., Cavallo, A., Cifaldi, M., Valacca, L., Villafiorita, A.: Improving safety assessment of complex systems: an industrial case study. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) Proceedings of the Formal Methods Europe Symposium (FM 2003), vol. 2805 of LNCS, pp. 208–222. Springer, Berlin Heidelbreg New York (2003)
15. Bozzano, M., Villafiorita, A.: Integrating fault tree analysis with event ordering information. In: Proceedings of the European Safety and Reliability Conference (ESREL 2003), pp. 247–254. Balkema, Rotterdam (2003)
16. Bozzano, M., Villafiorita, A., Åkerlund, O., Bieber, P., Bougnol, C., Böde, E., Bretschneider, M., Cavallo, A., Castel, C., Cifaldi, M., Cimatti, A., Griffault, A., Kehren, C., Lawrence, B., Lüdtke, A., Metge, S., Papadopoulos, C., Passarello, R., Peikenkamp, T., Persson, P., Seguin, C., Trotta, L., Valacca, L., Zacco, G.: ESACS: an integrated methodology for design and safety analysis of complex systems. In: Proceedings of the European Safety and Reliability Conference (ESREL 2003), pp. 237–245. Balkema, Rotterdam (2003)
17. Bryant, R.E.: Symbolic boolean manipulation with ordered binary decision diagrams. ACM Comput. Surv. **24**(3), 293–318 (1992)
18. Chiappini, A., Cimatti, A., Porzia, C., Rotondo, G., Sebastiani, R., Traverso, P., Villafiorita, A.: Formal specification and development of a safety-critical train management system. In: Felici, M., Kanoun, K., Pasquini, A. (eds.) Proceedings Conference on Computer Safety, Reliability and Security (SAFECOMP 1999), vol. 1698 of LNCS, pp. 410–419. Springer, Berlin Heidelberg New York (1999)
19. Ciardo, G., Jones, R.L., Miner, A.S., Siminiceanu, R.: SMART: Stochastic model analyzer for reliability and timing. In: Proceedings of the Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems, pp. 29–34 (2001)
20. Ciardo, G., Siminiceanu, R.: Structural symbolic CTL model checking of asynchronous systems. In: Hunt Jr, W.A., Somenzi, F. (eds.) Proceedings Conference on Computer Aided Verification (CAV 2003), vol. 2725 of LNCS, pp. 40–53. Springer, Berlin Heidelberg New York (2003)
21. Cimatti, A.: Industrial applications of model checking. In: Cassez, F., Jard, C., Rozoy, B.. Ryan, M.D. (eds.) Proceedings of the Modeling and Verification of Parallel Processes (MOVEP 2000), vol. 2067 of LNCS, pp. 153–168. Springer, Berlin Heidelberg New York (2001)
22. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV2: An open-Source tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) Proceedings Conference on Computer Aided Verification (CAV 2002), vol. 2404 of LNCS, pp. 359–364. Springer, Berlin Heidelberg New York (2002)
23. Cimatti, A., Clarke, E.M., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model checker. Softw. Tools Technol. Transf. **2**(4), 410–425 (2000)
24. Cimatti, A., Pieraccini, P.L., Sebastiani, R., Traverso, P., Villafiorita, A.: Formal specification and validation of a vital communication protocol. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) Proceedings of the World Congress on Formal Methods, (FM 1999), Vol. II, vol. 1709 of LNCS, pp. 1584–1604. Springer, Berlin Heidelberg New York (1999)

25. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
26. Cojazzi, G., Izquierdo, J.M., Meléndez, E., Perea, M.S.: The Reliability and safety assessment of protection systems by the use of dynamic event trees. The DYLAM-TRETA Package. In: Proceedings of the XVIII Annual Meeting Spanish Nucl. Soc. (1992)
27. Coudert, O., Madre, J.C.: Implicit and incremental computation of primes and essential primes of boolean functions. In: Proceedings of the Design Automation Conference (DAC 1992), pp. 36–39. IEEE Computer Society Press, (1992)
28. Coudert, O., Madre, J.C.: Fault tree analysis: 10^{20} prime implicants and beyond. In: Proceedings of the Annual Reliability and Maintainability Symposium (RAMS 1993), (1993)
29. Dabney, J.B., Harman, T.L.: Mastering Simulink. Prentice Hall, Englewood Cliffs, NJ (2003)
30. Deneux, J., Åkerlund, O.: A common framework for design and safety analyses using formal methods. In: Proceedings Conference on Probabilistic Safety Assessment and Management (PSAM7/ES-REL'04), (2004)
31. Devooght, J., Smidts, C.: Probabilistic dynamics: the mathematical and computing problems ahead. In: Aldemir, T., Siu, N.O., Mosleh, A., Cacciabue, P.C., Göktepe, B.G. (eds.) Reliability and Safety Assessment of Dynamic Process Systems, vol. 120 of NATO ASI Series F, pp. 85–100. Springer, Berlin Heidelberg New York (1994)
32. Doyle, S.A., Dugan, J.B.: Dependability assessment using binary decision diagrams (BDDs). In: Proceedings Symposium on Fault-Tolerant Computing (FTCS 1995), pp. 249–258. IEEE Computer Society Press (1995)
33. Dugan, J.B., Trivedi, K.S.: Coverage modeling for dependability analysis of fault-tolerant systems. IEEE Trans. Comput. **38**(6), 775–787 (1989)
34. Dutertre, B., Sorea, M.: Modeling and verification of a fault-tolerant real-time startup protocol using calendar automata. In: Lakhnech, Y., Yovine, S. (eds.) Proceedings of the Joint Conference on Formal Modeling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS/FTRTFT 2004), vol. 3253 of LNCS, pp. 199–214. Springer, Berlin Heidelberg New York (2004)
35. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings Conference on Software Engineering (ICSE 1999), pp. 411–420. ACM Press (1999)
36. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 995–1072. Elsevier, Amsterdam (1990)
37. Emerson, E.A., Mok, A.K., Sistla, A.P., Srinivasan, J.: Quantitative temporal reasoning. Real-Time Syst. **4**(4), 331–352 (1992)
38. Fenelon, P., McDermid, J.A., Nicholson, M., Pumfrey, D.J.: Towards integrated integrated safety analysis and design. Appl. Comput. Rev. **2**(1), 21–32 (1994)
39. The VIS Group. VIS: a system for verification and synthesis. In: Alur, R., Henzinger, T.A. (eds.) Proceedings Conference on Computer Aided Verification (CAV 1996), vol. 1102 of LNCS, pp. 428–432. Springer, Berlin Heidelberg New York (1996)
40. Henzinger, T.A.: The theory of hybrid automata. In: Proceedings Symposium on Logic in Computer Science (LICS 1996), pp. 278–292. IEEE Computer Society Press (1996)
41. Henzinger, T.A.: HyTech: A model checker for hybrid systems. Softw. Tools Technol. Transf. **1**, 110–122 (1997)
42. Hinchey, M.G., Bowen, J.P.: (eds.) Industrial Strength Formal Methods in Practice. Formal Approaches to Computing and Information Technology. Springer, Berlin Heidelberg New York (1999)
43. Holzmann, G.J.: The model checker SPIN. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997)
44. Joshi, A., Heimdahl, M.P.E.: Model-based safety analysis of simulink models using SCADE design verifier. In: Winther, R., Gran, B.A., Dahll, G. (eds.) Proceedings Conference on Computer Safety, Reliability and Security (SAFECOMP 2005), vol. 3688 of LNCS, pp. 122–135. Springer, Berlin Heidelberg New York (2005)
45. Joshi, A., Miller, S.P., Whalen, M., Heimdahl, M.P.E.: A Proposal for Model-Based Safety Analysis. In: Proceedings of the AIAA/IEEE Digital Avionics Systems Conference (2005)
46. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. Softw. Tools Technol. Transf. **1**(1-2), 134–152 (1997)
47. Liggesmeyer, P., Rothfelder, M.: Improving System Reliability with Automatic Fault Tree Generation. In: Proceedings Symposium on Fault-Tolerant Computing (FTCS 1998), pp. 90–99. IEEE Computer Society Press (1998)
48. Manian, R., Coppit, D.W., Sullivan, K.J., Dugan, J.B.: Bridging the gap between fault tree analysis modeling tools and the systems being modeled. In: Proceedings of the Annual Reliability and Maintainability Symposium (RAMS 1999), pp. 105–111 (1999)
49. Manian, R., Dugan, J.B., Coppit, D., Sullivan, K.J.: Combining various solution techniques for dynamic fault tree analysis of computer systems. In: Proceedings of the High-Assurance Systems Engineering Symposium (HASE 1998), pp. 21–28. IEEE Computer Society Press (1998)
50. Manquinho, V.M., Oliveira, A.L., Marques-Silva, J.P.: models and algorithms for computing minimum-size prime implicants. In: Proceedings of the International Workshop on Boolean Problems (IWBP 1998) (1998)
51. Marseguerra, M., Zio, E., Devooght, J., Labeau, P.E.: A concept paper on dynamic reliability via Monte Carlo simulation. Math. Comput. Simulat. **47**, 371–382 (1998)
52. McMillan, K.L.: Symbolic Model Checking. Kluwer, Netherlands (1993)
53. Miller, S.P., Tribble, A.C., Heimdahl, M.P.E.: Proving the Shalls. In: Proceedings of the Formal Methods Europe (FM 2003), vol. 2805 of LNCS, pp. 75–93. Springer, Berlin Heidelberg New York (2003)
54. Papazoglou, I.A.: Markovian reliability analysis of dynamic systems. In: Aldemir, T., Siu, N.O., Mosleh, A., Cacciabue, P.C., Göktepe, B.G. (eds.) Reliability and Safety Assessment of Dynamic Process Systems, vol. 120 of NATO ASI Series F, pp. 24–43. Springer, Berlin Heidelberg New York (1994)
55. Peikenkamp, T., Böede, E., Brückner, I., Spenke, H., Bretschneider, M., Holberg, H.-J.: Model-based safety analysis of a flap control system. In: Proceedings of the International Symposium INCOSE 2004 (2004)
56. Rae, A.: Automatic fault tree generation - missile defence system case study. Technical Report 00-36, Software Verification Research Centre, University of Queensland (2000)
57. Rauzy, A.: New algorithms for fault trees analysis. Reliab. Eng. Syst. Safe. **40**(3), 203–211 (1993)
58. Rauzy, A., Dutuit, Y.: Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia. Reliab. Eng. Syst. Safe. **58**(2), 127–144 (1997)
59. Schäfer, A.: Combining real-time model-checking and fault tree analysis. In: Proceedings of the Formal Methods Europe (FM 2003), vol. 2805 of LNCS, pp. 522–541. Springer, Berlin Heidelberg New York (2003)
60. Siu, N.O.: Risk assessment for dynamic systems: an overview. Reliab. Eng. Syst. Safe. **43**, 43–74 (1994)
61. Smidts, C., Devooght, J.: Probabilistic reactor dynamics II. A Monte-Carlo study of a fast reactor transient. Nucl. Sci. Eng. **111**(3), 241–256 (1992)
62. Sullivan, K.J., Dugan, J.B., Coppit, D.: The Galileo fault tree analysis tool. In: Proceedings Symposium on Fault-Tolerant Computing (FTCS 1999), pp. 232–235. IEEE Computer Society Press (1999)

63. Tang, Z., Dugan, J.B.: An integrated method for incorporating common cause failures in system analysis. In: Proceedings of the Annual Reliability and Maintainability Symposium (2004)
64. Thums, A., Schellhorn, G.: Model checking FTA. In: Proceedings of the Formal Methods Europe (FM 2003), vol. 2805 of LNCS, pp. 739–757. Springer, Berlin Heidelberg New York (2003)
65. Tribble, A.C., Lempia, D.L., Miller, S.P.: Software safety analysis of a flight guidance system. In: Proceedings AIAA/IEEE Digital Avionics Systems Conference (2002)
66. Tribble, A.C., Miller, S.P.: Software safety analysis of a flight management system vertical navigation function — a status report. In: Proceedings AIAA/IEEE Digital Avionics Systems Conference (2003)
67. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault Tree Handbook. Technical Report NUREG-0492, Systems and Reliability Research Office of Nuclear Regulatory Research U.S. Nuclear Regulatory Commission (1981)
68. Wing, J.M.: A specifier's introduction to formal methods. IEEE Comput. **23**(9), 8–24 (1990)