Formal Analysis of Fault Tree using Probabilistic Model Checking: A Solar Array Case Study

Marwan Ammar, Khaza Anuarul Hoque, Otmane Ait Mohamed
Concordia University
Montreal, Canada
Email: {m_amma,k_hoque,ait}@ece.concordia.ca

Abstract—**Fault Tree Analysis (FTA) is a widespread technique used to assess the reliability of safety-critical systems. The traditional way of conducting FTA is either through paper and pencil proof or through computer simulation techniques, which are inefficient and prone to inaccuracy. In this paper, we propose the use of probabilistic model checking to automatically analyze fault trees of safety-critical systems. Our methodology consists in the probabilistic formalization of the gates used in a fault tree to a Discrete-Time Markov Chain (DTMC) and a Markov Decision Process (MDP), and the subsequent probabilistic verification using PRISM tool to quantitatively analyze the system. To illustrate the proposed approach we perform the fault tree analysis of a solar array system, used as power source for the DFH-3 satellite. The results show that harsh thermal environment is the main cause of system failures.**

I. Introduction

Fault Tree Diagram (FTD) is a top-down graphical model of a system, which represents all paths and events that may lead to failure within that system [1]. Events in FTDs are nodes, connected through logic gates in such a way that an error in one of the bottom nodes can propagate to the higher level nodes and reach the top-level event, compromising the functionality of the entire system. Fault Tree Analysis (FTA) is the study of such diagrams in order to discover and assess the effect of undesirable events or faults [1]. FTA allows safety and reliability engineers to better understand how the system can fail, identifying the best possible ways to make it safer, as well as the system's event rates. FTA is commonly used in the aerospace industry for both hardware and software [1] as means for investigating a system's modes, potential faults occurrences with their causes, and to quantify their contribution to system unreliability in the course of product design. Traditionally, FTA is based on simulation techniques [2], [1], [3], with the main techniques being: Monte-Carlo simulation, Quasi-Monte-Carlo method, time-sequential simulation, and discrete event simulation [4]. Assessing the causes and the probability of a punctual failure occurrence in the system using simulation-based techniques is very costly, since each failure condition must be evaluated separately, one at a time, creating a very large state-space and requiring tremendous effort to analyze the whole scope of the system.

An alternative to avoid the aforementioned problem is the use of Probabilistic Model Checking(PMC). PMC is a formal verification method that designates a collection of techniques for the automatic analysis of reactive, finite state concurrent systems. This technique has several advantages over simulation. Notably, probabilistic model checking is an exhaustive, accurate, efficient and completely automated verification technique [5], providing a comprehensive and reliable solution for fault tree analysis. In this work, we propose a PMC-based methodology for FTA modeling, using PRISM language [6]. Added to the inherent advantages of PMC listed above, PRISM's modularity allows for easily expandable, state-efficient models. The modeling methodology is applied to a case study of a solar array mechanical system [7], first using Discrete-Time Markov Chain (DTMC) [8] to model known environment scenarios where the probabilistic distribution of the system's behavior is known, then using Markov Decision Process (MDP) to model the non-deterministic behavior of the system when subjected to unknown environments.

Our goal is to provide a way for developers to evaluate the weaknesses of their systems. Through PMC properties, our approach can be used to ascertain not only correctness, but also quantitative measures such as performance and reliability, without the time and intensive processing required by simulation techniques. This work focuses on evaluating the dynamics of fault propagation in FTDs, and we observe that the lowest levels in the tree are the most positively affected by fault masking, events connected to multiple gates are especially unreliable in non-deterministic environments (MDP), and that events connected to *AND* gates are more affected by non-determinism than events connected to *OR* gates.

The following section presents some important background information about PMC, the PRISM tool and FTA. Section III considers related works. In Section IV we describe our modeling approach. In Section V we demonstrate the application of our proposed methodology, performing the analysis on a solar array FTD and Section VI concludes the paper with some future research directions.

II. Preliminaries

A. Probabilistic Model Checking with PRISM

Probabilistic model checking is a formal verification technique derived from regular model checking and applied on systems that present a random or probabilistic behavior, like real life applications, where the resulting models usually contain a very large number of states. This technique can deal with a wide range of quantitative measures; the results show an exact figure of the property being verified, usually in parts-per-hundred; it can be fully automated, provides an exhaustive analysis of the model, and it is very efficient. PMC works with several model types and temporal logic specification languages. In this paper we use DTMC and MDP as the model types and Probabilistic Computation-Tree Logic (PCTL) as the property specification language. In [9] DTMC is defined as a tuple *D=(S, $\overline{s}$,P,L)* where *S* is a countable set of states, $\overline{s} \in S$ is an initial state, $P : S \times S \rightarrow [0,1]$ is a transition probability matrix such that $\sum_{s' \in s} P(s,s') = 1$ for all $s \in S$, and $L : S \rightarrow 2^{AP}$ is a labeling function mapping each state to a set of atomic propositions taken from a set AP. An MDP is defined in [9] as a tuple $M = (\mathrm{S}, \overline{s}, \alpha_{\mathrm{M}}, \delta_{\mathrm{M}}, \mathrm{L})$ where S is a finite set of states, $\overline{s} \in S$ is an initial state, α_{M} is a finite alphabet, $\delta_{\mathrm{M}} : \mathrm{S} \times \alpha_{\mathrm{M}} \rightarrow Dist\,(S)$ is a (partial) probabilistic transition function and $\mathrm{L} : \mathrm{S} \rightarrow 2^{AP}$ is a labelling function mapping each state to a set of atomic propositions taken from a set AP. As such, the MDP is a stochastic system where all the decisions are made in a non-deterministic manner.

To complete the model checking process, we specify properties using a probabilistic extension of CTL temporal logic called PCTL. PCTL can be used with both DTMC and MDP models, working at discrete time domain. The main difference is that using PCTL over MDP model requires to extend the *P[]*(probability query) operator with the *min* and *max* operators. As such, each path formula is evaluated in a best or worse case scenario [10]. Below are two illustrative examples with their natural language translation:

1) DTMC - $P_{>0.8}$ $[\neg\ a \bigcup b]$ - "The probability of *a* being false *until b* is true is bigger than 0.8.
2) MDP - P_{min} $[\ F\ (a\ >\ 0) \bigcup \neg\ b]$ - "What's the minimum probability that *eventually a* will be bigger than zero *until b* is false.

We perform PMC with PRISM [6], a free, open source probabilistic symbolic model checker. It works with its own high-level modeling language, written as state-based modules. Each module is composed by a set of guarded commands. PRISM supports a wide range of model analysis methods and it features a very efficient implementation, making use of multiple model checking engines (based on BDDs and their extensions). These engines enable PRISM to handle models with up to 10^8 states. The model checking is done by dynamically creating graph-based computations [11] in order to reach a numerical solution based on linear equation systems and optimization problems. PRISM also features advanced algorithms such as symmetry reduction and abstraction refinement. The syntax of the PRISM language as well as some examples will be given in Section IV.

B. Fault Tree Analysis

The fault tree analysis method was developed in 1962 by Bell Telephone Laboratories and it is a widely used method for risk assessment, mainly in the area of avionics, nuclear and chemical industries [1]. FTA follows a deductive approach, which means that it starts from an undesirable general event in order to find what circumstances may lead to that event. In the context of FTA, the general event is known as *top event*, from which the fault tree branches out vertically. The top event is defined as the failing point of a system in operating conditions, whether those conditions are considered normal or abnormal. A single fault tree can be used to analyse one single top event, which can then be fed into another fault tree as a bottom event. Bottom events are the ones at the very bottom of the fault tree, independent from any other events, and, assuming the FTA is performed following a quantitative evaluation, these events receive fault probabilities that will dynamically spread through the rest of the tree during the analysis. The elements of a Fault Tree and their graphical representations are summarized in Table I.

TABLE I: Elements of a Fault Tree

Token	Element	Description
▭	Top or Intermediary Event	System or component failure
⬭	Bottom Event	A basic initiating fault event
▢	Conditioning Event	Specific condition or restriction that can apply to any gate
⌂	External Event	Event that is normally expected to occur
◇	Undeveloped Event	Event that's not further developed due to lack of importance or knowledge
⌓	AND Gate	The output is true if all inputs are true
⌓	OR Gate	The output is true if at least one input is true
⬡ n	Combination Gate	The output is true if *n* inputs are true
⌓	Exclusive OR Gate	The output is true if exactly one of the inputs is true
⌓	Priority AND Gate	The output is true if all the inputs become true in a specific sequence
⬡	Inhibit Gate	The output is true if the single input becomes true in the presence of an enabling condition

III. Related Works

PRISM model checker has several application domains, specially for safety critical systems. In [12] the authors assess the feasibility of using model checking for verification of Unmanned Aircraft Systems (UAS) in civil airspace. The authors begin by modeling simple UAS systems into the SPIN tool and then refining the model by incorporating probabilities and using probabilistic model checking with PRISM. Lastly, they model the UAS using the autonomous agent language Gwendolen and compare and contrast the various approaches. The work in [13] uses PRISM tool to perform the formal modeling and verification of RAM related properties on satelite systems, using Erlang distribution to improve discrete time delays in CTMC by approximating nonexponential holding times with intermediate states based on a phase type distribution.
In recent years, a number of works related to fault tree analysis has emerged. In [14], DFTCalc tool for fault tree analysis is presented. It is capable of modeling fault trees via compact representations and the dependability analysis is performed using stochastic techniques. DFTCalc allows modelling of most FTD constructs but it cannot check the correctness and the completeness of fault trees. Moreover, an important difference is that their work does not include the possibility of fault

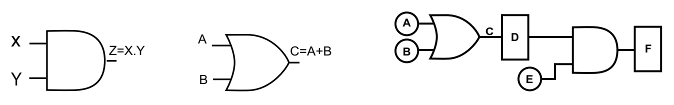
masking, whereas in our work this important phenomenon is taken into account. The work presented in [15] gravitates towards theorem proving, using HOL4 and higher-order logics to analyse safety-critical systems through FTA by formalizing the gates of a fault tree and conducting FTA-based failure analysis. However, a model checking approach has several advantages over theorem proving, such as being systematically exhaustive, fully automated, and more time efficient. In [16], authors propose the use of Binary Decision Diagrams (BDD) to perform quantitative analysis in fault trees. Their method improves the accuracy of the calculated failure rates of bottom events over simulation techniques. It consists of converting the FTD into a format compatible with Shannon's decomposition, allowing the failure rates to be accurately calculated. The binary decision diagram approach is extended to qualitative FTA analysis, in [17], where BDDs are employed to evaluate minimal cutsets of a fault tree without creating probabilistic inaccuracies like in the conventional qualitative analysis techniques.

All these works exemplify the versatility and importance of fault tree diagrams and their relevance for assessment of safety-critical systems related to diverse areas. The research presented in this paper is different than what is found in the related work because on top of formalizing the various gates of a fault tree, allowing for the representation of virtually any system, we introduce a modeling technique that is state-efficient and easily scalable. In addition, our modeling can handle both quantitative and qualitative analysis, with a simple change in the PCTL properties being verified.

IV. Modeling

In this section we show how the fault tree diagrams are modeled in PRISM. Our emphasis is on *AND* and *OR* gates, since those are the types of gates present in the case study, presented in Section V. The fault-masking mechanic adds a probability of fault mitigation inside the gates, which are designed to allow easy system composition with a reduced number of state transitions. The modeling of the other FTA gates follow a similar approach but are not included in this paper due to space constraints. Since the DTMC and the MDP modeling approaches are similar in PRISM, we will focus our explanation on the DTMC modeling, with graphical representations and a detailed explanation of each gate's state transitions in Subsections IV-A and IV-B. It is important to note that the main difference between DTMC and MDP is that in DTMC, in each state, the successor state is determined by a discrete probability distribution, whereas in MDP, in each state, the successor state is determined by a nondeterministic choice between several discrete probability distributions.

MDP model, in each state S, the successor state is decided in two steps: the first step is non-deterministic and the second step is random, according to the probability distribution of the transition matrix. The DTMC model only has one step, which is the random probabilistic step.



(a) AND gate (b) OR gate (c) 2-gate Example

Fig. 1: Fault Tree Gates

A. Modeling of an AND Gate

The *AND* gate is defined as follows:

Definition 1: Given two inputs X and Y and their output Z, connected through an *AND* gate, output Z becomes true if and only if X and Y are true. Figure 1(a) is a representation of an *AND* gate. The *AND* modeling in PRISM follows these assumptions:

1) All the inputs to the AND gate represent events, each of which have a probability of being triggered.
2) Only one input can trigger at a time.
3) If one of the inputs of the *AND* gate is triggered then the other will be given an additional probability of triggering.
4) Before an output is generated, there is a certain probability that the fault will be masked.

The model of the *AND* gate can be defined formally as a finite transition system $(S, \overline{s}, P, L)$, where S is the set of states $S = (S_0, S_1, S_2, S_3)$, $\overline{s}$ is the initial state $\overline{s} = S_0$, P is a transition probability matrix, P_{ij}, such that $P_{\overline{s},S_1} = (p_1)$, $P_{\overline{s},S_2} = (p_2)$, $P_{S_1,S_3} = (p_3)$, $P_{S_1,S_4} = (p_5)$, $P_{S_2,S_3} = (p_4)$, $P_{S_2,S_4} = (p_5)$ and $L : S \rightarrow 2^{AP}$ is a labeling function, mapping states with properties of interest, where L(S_3)=*propagate*. The DTMC model of the *AND* gate is shown in Fig. 2.

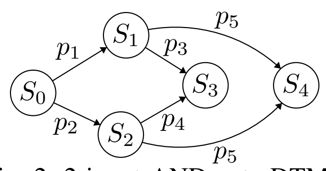


Fig. 2: 2-input AND gate DTMC

Starting from an initial state S_0(X=0, Y=0) the next state can either be S_1(X=1, Y=0) or S_2(X=0, Y=1), with probabilities *p1* and *p2*, respectively. At this point, the system can either move to S_3(X=1, Y=1, Z=1), with probability *p3* or *p4*, signifying fault propagation, or the system can move to S_4(X=0, Y=0, M=1), with probability *p5*, signifying fault masking. The model in Fig. 2 is then encoded into PRISM. A PRISM command is a tuple $cmd = (act, guard, rate, action)$ following the format `[<act>] <guard> → <rate> : <action>;`, where *act* is an action label, *guard* is a predicate over a variable, *rate* is a numerical evaluation referent to the probability of an action and *action* is a set of *n* variable updates that will translate into transitions in the model. Fig. 3 shows the PRISM representation of the *AND* gate. Please note that variable *M* stands for *masking* and variable *and* indicates if the module is idle (and=0), waiting for first input (and=1) or waiting for second input (and=2).

```
module and_gate
[](and=1)&(X=0)&(Y=0)&(M=0)&(Z=0)->p1:(X'=1)&(and'=2)
                                          +p2:(Y'=1)&(and'=2);
[](X=1)&(Y=0)&(M=0)->p5:(M'=1)&(X'=0)+p3:(Y'=1)&(Z'=1);
[](Y=1)&(X=0)&(M=0)->p5:(M'=1)&(Y'=0)+p4:(X'=1)&(Z'=1);
endmodule
```

Fig. 3: PRISM modeling of an AND gate

B. Modeling of an OR Gate

The *OR* gate, seen in Figure 1(b), is defined as:

Definition 2: Given two inputs *A* and *B* and their output *C*, output *C* becomes true if any of the inputs *A* or *B* are true.

The modeling process takes into consideration the following assumptions:

1) All the inputs to the gate represent an event, each of which have a probability of being triggered.
2) Only one input can trigger at a time, after which no other input can trigger.
3) Before an output is generated, there is a certain probability that the fault will be masked.

We formally define the *OR* gate as a finite transition system $(S, \overline{s}, P, L)$, where *S* is the set of states $S = (S_0, S_1, S_2, S_3)$, $\overline{s}$ is the initial state $\overline{s} = S_0$, P is a transition probability matrix, p_{ij}, such that $P_{\overline{s},S_1} = (p_6)$, $P_{\overline{s},S_2} = (p_7)$, $P_{S_1,S_3} = (p_8)$, $P_{S_2,S_3} = (p_9)$, $P_{S_1,S_4} = (p_{10})$, $P_{S_2,S_4} = (p_{10})$ and $L : S \rightarrow 2^{AP}$ is a labeling function, mapping states with properties of interest, where L(S_3)=*propagate*. The *OR* gate DTMC model is shown in Figure 4.

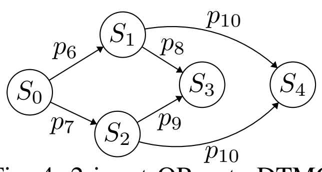


Fig. 4: 2 input OR gate DTMC

From an initial state S_0(A=0, B=0) the next state can either be S_1(A=1, B=0) or S_2(A=0, B=1) with probabilities p_6 and p_7 respectively. At this point, the system may either move to S_3(A=0, B=0 and C=1), with probabilities p_8 or p_9, signifying fault propagation, or to S_4(A=0, B=0 and M=1), with probability p_{10}, signifying fault masking. The *OR* gate model is then encoded into PRISM as shown in Fig. 5. The *or* variable is equivalent to the *and* variable, previously explained.

```
module or_gate
[](or=1)&(A=0)&(B=0)&(M=0)&(C=0)->p1:(A'=1)&(or'=2)
                                        +p2:(B'=1)&(or'=2);
[](A=1)&(C=0)&(M=0)->p5:(M'=1)&(A'=0)+p3:(A'=0)&(C'=1);
[](B=1)&(C=0)&(M=0)->p5:(M'=1)&(B'=0)+p4:(B'=0)&(C'=1);
endmodule
```

Fig. 5: PRISM modeling of an OR gate

C. Sample Modeling of a 2-Gate System

To illustrate the modeling process of a fault tree using our modular approach (pre-modeled gates in PRISM), we provide an example using a simple two gates fault tree with three inputs *A*, *B*, *E*, and one output *F*. *A*, *B* and *E* are bottom events. *D* is an intermediary event and *F* is the top event for this example, as shown in Figure 1(c).The assumptions listed in Sections IV-A and IV-B are also applicable for this example, with two additions:

1) There exists one additional module *twogate*, in the PRISM code, that serves as a control module, where the order of the gates in the system can be specified.
2) The system starts at the *OR* gate, where inputs A and B each have a probability of triggering.
3) Output C becomes input D as it enters the *AND* gate.
4) Input *E* has a chance of being triggered after the output C propagates, following our *AND* gate assumptions.

In this small example, we illustrate how to use the gates, defined in the previous sections, as building blocks to construct more complex fault tree diagrams. Fig. 6 shows the DTMC model of the two-gates example.

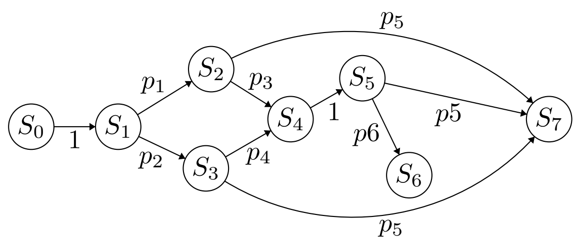


Fig. 6: 2-gate DTMC

From state S_0, the system is initialized and moves to S_1. In S_1 the system can go to S_2 or S_3 with probabilities *p1* or *p2*. From S_2 or S_3, the system can go to S_7, with probability *p5*, signifying masking, or to S_4, signifying propagation of the *OR* gate. The output of the *OR* gate serves as one of the inputs of the *AND* gate, thus the system moves to S_5. In S_5, if input *E* is triggered and the output propagates, the system moves to state S_6, otherwise the system moves to S_7, signifying masking. The two-gates model is encoded in PRISM as shown in Fig. 7.

```
module twogate
[] or=0 -> (or'=1);
[] c=1 -> (c'=0)&(d'=1);
endmodule
module or_gate
[](or=1)&(a=0)&(b=0)&(m=0)&(c=0)->p1:(a'=1)&(or'=2)
                                        +p2:(b'=1)&(or'=2);
[](a=1)&(c=0)&(m=0)->p5:(m'=1)&(a'=0)+p3:(a'=0)&(c'=1);
[](b=1)&(c=0)&(m=0)->p5:(m'=1)&(b'=0)+p4:(b'=0)&(c'=1);
endmodule
module and_gate
[](and=1)&(d=0)&(e=0)&(m=0)&(f=0)->p6:(d'=1)&(and'=2)
                                         +p7:(e'=1)&(and'=2);
[](d=1)&(e=0)&(m=0)->p5:(m'=1)&(d'=0)+p8:(e'=1)&(f'=1);
[](e=1)&(d=0)&(m=0)->p5:(m'=1)&(e'=0)+p9:(d'=1)&(f'=1);
endmodule
```

Fig. 7: PRISM modeling of the 2-gate example

It is important to note that all variables are declared globally, with starting value of zero. The variable declarations are suppressed in Fig. 7 for space constrains. The module *twogate* controls the flow of the system. In the first line of the module, the *or* variable sets the starting point of the system, activating the *OR* gate. The second line of the module takes the output of the *OR* gate, C, and channels it to the input of the *AND* gate, D. The modules above can be used to build a fault tree diagram, in any desired configuration, by PRISM's module renaming feature.

V. Case Study

To show the applicability of our approach, we perform a quantitative analysis on the solar array case study, taken from [7]. We will present the obtained analytical results and a possible solution to improve reliability. Solar arrays are one of the most important components of any satellite mission, as they generate power for all other components of the satellite. The arrays are usually in a folded position during the launch phase of the satellite, becoming unfolded once the satellite is fully deployed in space. The goal of this component is to have the solar array aimed at the sun at all times in order to maximise power generation for the satellite. Fig. 8 shows the fault tree diagram of the mechanical components in the solar array. A detailed description of each component can be found in [7].

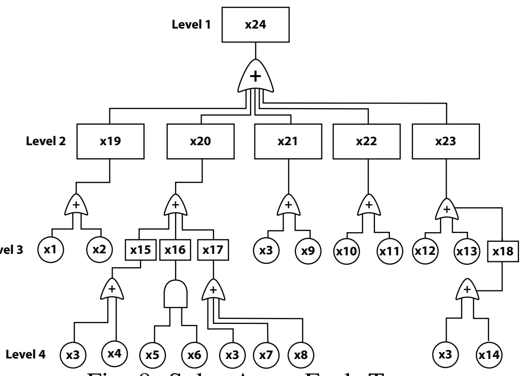


Fig. 8: Solar Array Fault Tree

The top event *X24* represents the failure of the solar array system. Intermediary events *X19*, *X20*, *X21*, *X22* and *X23*, in the second level of the tree, represent the possible causes of failure in the solar array. At the third and fourth levels, *X1*, *X2*, *X3*, *X4*, *X5*, *X6*, *X7*, *X8*, *X9*, *X10*, *X11*, *X12*, *X13*, and *X14* are the bottom events, that can cause a failure in events *X19*, *X20*, *X21*, *X22* and *X23*. The solar array fault tree was modeled first, as a DTMC and then, as an MDP. We reiterate that the choice of those types of Markov chain was motivated by the need to verify the system in a known and predictable environment as well as in an unknown environment. Every bottom event in the fault tree has a probability of being triggered and such a probability is based on the truth degree values specified in Table II [7]. For the sake of simplicity, we assume that only one bottom event can trigger at a time, with the exception of *X4* and *X5*, since those events are connected to an *AND* gate and thus both must be triggered to model the behavior of fault propagation. All other events in the tree are connected through *OR* gates. The objective of the experiment is to assess and compare the likelihood of faults originated at the different bottom events of the tree to reach the top event *X24* causing a system failure. It is important to note that bottom event *X3* is connected to multiple gates in the system. This makes *X3* the most important bottom event in the tree since it has the highest probability of triggering a fault in another node. The model is encoded in PRISM following the approach specified in Section IV. The output of each logical gate has intrinsic probability of fault masking which, in the real world, means a non-destructive failure or a transient fault, which the system is able to detect and fix by itself, continuing its normal operation. The probabilities of fault masking are the same for all gates of the model. We performed our analysis using two different values for masking probabilities, 5% then 10%. However, since the model is parametric, any other value can be easily evaluated. The probabilities of each bottom event to reach a system failure are evaluated by verifying PCTL properties in PRISM, for both the DTMC and MDP models. The property for a node X_a, connected to an *OR* gate, is defined in PCTL as follows:

Property 1: $P_{max} =?\ [(F\ X_a = 1) \&\ (F\ X24 = 1)]$ - "What is the maximum probability that eventually X_a will trigger and eventually the fault will propagate to *X24*, causing a system failure".

The property for two nodes X_b and X_c, connected to an *AND* gate, is written as follows:

Property 2: $P_{max} =?\ [(F\ X_b = 1) \&\ (F\ X_c = 1) \& (F\ X24 = 1)]$ - "What is the maximum probability that eventually X_b will trigger and eventually X_c will trigger and eventually the fault will propagate to *X24*, causing a system failure".

We use DTMC in the first experiment to assess the probability that a failure originated in a bottom event will reach the top event and the results are presented in Table III.

TABLE II: Fault Probability of Bottom Events

X1	X2	X3	X4	X5	X6	X7
4%	6%	10%	4%	6%	4%	8%
X8	**X9**	**X10**	**X11**	**X12**	**X13**	**X14**
6%	8%	3%	5%	8%	8%	8%

TABLE III: Top Event Failure Probability(DTMC)

Event	5% Mask	10% Mask	Event	5% Mask	10% Mask
X1	0.0361%	0.0324%	X8	0.0514%	0.0437%
X2	0.0541%	0.0486%	X9	0.0722%	0.0648%
X3	0.0868%	0.0749%	X10	0.0270%	0.0243%
X4	0.0342%	0.0291%	X11	0.0451%	0.0405%
X5	0.0021%	0.0018%	X12	0.0722%	0.0648%
X6	0.0021%	0.0018%	X13	0.0722%	0.0648%
X7	0.0685%	0.0583%	X14	0.0685	0.0583%

The second set of experiments is conducted using MDP, to add non-determinism into the model. The scenarios are evaluated in the same manner as the DTMC experiment and Table IV summarizes the result.

TABLE IV: Top Event Failure Probability(MDP)

Event	5% Mask	10% Mask	Event	5% Mask	10% Mask
X1	0.0361%	0.0324%	X8	0.0514%	0.0437%
X2	0.0541%	0.0486%	X9	0.0722%	0.0648%
X3	0.0902%	0.0810%	X10	0.0270%	0.0243%
X4	0.0342%	0.0291%	X11	0.0451%	0.0405%
X5	0.0041%	0.0034%	X12	0.0722%	0.0648%
X6	0.0041%	0.0034%	X13	0.0722%	0.0648%
X7	0.0685%	0.0583%	X14	0.0685	0.0583%

After comparing the results, it becomes clear that the increased probability of masking is considerably more effective for errors occurring in the lower layers of the fault tree. It is notable that while most results are the same for DTMC and MDP, they do differ in *X3*, *X5* and *X6*. The observed reason for these differences lies in the non-determinism present at the core of an MDP. *OR* gates are simple because one error is enough to trigger an output and a chance of propagation to the next node thus, according to out modeling assumptions, the path from bottom to top event becomes more linear and has less room for randomness. For an *AND* gate the scenario is different because the output generation and propagation depends on faults that occur on two bottom events concurrently. As such, there are multiple ways the error scenario can play out (input 1 before input 2, input 2 before input 1, input 1 but not input 2 and so forth), thus opening more possibilities for non-determinism.

It is also of importance that event *X3* is connected to multiple *OR* gates at the same time, alowing for non-determinism. Another very important observation is that the drop in system failure rate between 5% and 10% masking in the *AND* gates of the MDP model was 17.07%, which is the biggest gain in reliability observed in this experiment. Although the overall reliability of *AND* gates is greater in the DTMC model, the reliability gain of those gates is higher in the presence of non-determinism. Our analysis shows that the main causes of system failures in the solar array are components *X3*, *X9*, *X12*, and *X13*. Remarkably, the bottom events that generate all the above failures are located in the second layer of the fault tree and are connected through *OR* gates. As an additional experiment, we take the number one cause of system failure listed above (bottom event *X3* propagating through *OR* gate to *X21*) and we run a test in a hypothetical scenario where we add system redundancy and replace the *OR* gate with an *AND* gate, as seen in Fig. 9. Since *X21* and *X24* are connected through an *OR* gate, we ignore all other connections to that gate.

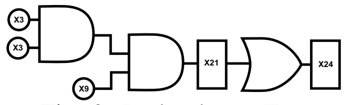


Fig. 9: Redundancy Test

In such hypotetical scenario, the drop in system failure would be of 99.2%, using a DTMC model and a masking rate of 10%. At the end of these experiments, it is clear that the best way to improve the reliability of a fault tree, and consequently the system that the tree is based on, is to place the most critical nodes under *AND* gates, preferably with system redundancy, and to place such critical nodes further down in the tree.

VI. Conclusion

The accuracy of the failure assessment is of utmost importance in safety-critical environments, where a system error may lead to catastrophic outcomes. In this paper, we have proposed a methodology for accurately performing fault tree analysis using probabilistic model checking. The technique consists of modeling the logic gates in a fault tree diagram into DTMCs and MDPs that are encoded into the probabilistic model checker PRISM. The methodology is, then, used to conduct a probabilistic analysis on a solar array fault tree diagram, focusing on the likeness of a single fault to propagate and cause a top event failure. A comparison is shown between the results of both Markov chain models, with an analysis conducted on those results, pointing out the FTD's critical points. Building upon the methodology presented in this paper, a few other elements can be added to make for more complex FTA scenario, such as the introduction of time and concurrency of events.

References

[1] W. Vesely, F. Goldberg, N. Roberts, and D. Haasl, "Fault tree handbook (nureg-0492)," in *http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf*, 1981.

[2] W. Lee, D. Grosh, F. Tillman, and L. C.H., "Fault tree analysis, methods, and applications - a review," pp. 194 – 203, 1985.

[3] K. Raoa, V. Gopikaa, V. Raoa, H. Kushwahaa, A. Vermab, and A. Srividyab, "Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment," in *Reliability Engineering and System Safety, Volume 94, Issue 4*, 2009, p. 872883.

[4] J. Faulin Fajardo, A. Juan Perez, S. Martorell Alsina, and J. Ramirez-Marquez, "Simulation methods for reliability and availability of complex systems," 2010.

[5] C. Baier and J. Katoen, "Principles of model checking," 2008.

[6] "Prism website," in *http://www.prismmodelchecker.org*.

[7] J. Wu, S. Yan, and L. Xie, "Reliability analysis method of a solar array by using fault tree analysis and fuzzy reasoning petri net," in *Acta Astronautica Volume 69, Issues 1112*, December 2011, p. 960968.

[8] J. Norris, "Markov chains," in *Cambridge University Press*, 1997.

[9] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, *Formal Methods for Eternal Networked Software Systems*, 2011.

[10] D. Ardagna, C. Ghezzi, and R. Mirandola, "Rethinking the use of models in software architecture," in *4th International Conference on the Quality of Software-Architectures*, 2008, pp. 1–27.

[11] T. Mähne, A. Vachoux, and V. E., "Proposal for a bond graph based model of computation in systemc-ams," pp. 25–31, 2007.

[12] M. Webster, M. Fisher, N. Cameron, and M. Jump, "Towards certification of autonomous unmanned aircraft using formal model checking and simulation," in *30th International Conference, SAFECOMP, Naples, Italy*, 2011.

[13] K. A. Hoque, O. A. Mohamed, and Y. Savaria, "Towards an accurate reliability, availability and maintainability analysis approach for satellite systems based on probabilistic model checking," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1635–1640.

[14] F. Arnold, A. Belinfante, F. Van der Berg, D. Guck, and M. Stoelinga, "Dftcalc: A tool for efficient fault tree analysis," in *32nd International Conference, SAFECOMP 2013, Toulouse, France, September 24-27, 2013. Proceedings*, September 2013, pp. 293–301.

[15] W. Ahmed and O. Hasan, "Towards formal fault tree analysis using theorem proving," in *International Conference, CICM 2015*, 2015, pp. 13–17.

[16] R. Sinnamon and J. Andrews, "Improved accuracy in quantitative fault tree analysis," in *Proceedings of the 12th Advances in Reliability Technology Symposium, Manchester, UK*, 1996.

[17] ——, "Improved efficiency in qualitative fault tree analysis," in *Proceedings of the 12th Advances in Reliability Technology Symposium, Manchester, UK*, 1996.