Verification and Fault Analysis based on Combination of AADL and Modelica

Tao Huang¹, Zining Cao¹,²,³ and Qing Li¹
¹College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, China
²Science and Technology on Electro-optic Control Laboratory, Luoyang 471023, China
³Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210023, China
Taohuang@nuaa.edu.cn

Abstract. CPS is a multidimensional complex system that can realize the interaction between computing process and physical process. Aiming at the problems of fault occurrence and uncertain behavior, this paper proposes the fault analysis stochastic hybrid automata as a formal model, the attributes of randomness and fault analysis are added through AADL behavior attachment to expand the attributes of hybrid automata, and applies the extended automata to the embedded system for system description and fault analysis. The model is used to model the fire control system, and AADL, Modelica and fault tree are combined to form a new model. The behavior is analyzed, and the conversion algorithm and conversion example are given.

1. Introduction

Cyber-Physical System [1] is a next generation intelligent system based on embedded system, computer network and wireless sensor network, etc. For the in-depth analysis and research of CPS, preliminary modeling is essential. In the current research field, semiformal modeling tools include AADL, Modelica and so on. AADL [2] is a modeling language, which is widely used in the modeling of key application system models in aerospace industry. In practical application, Due to the limitations of AADL, it needs to be extended. The purpose of extending AADL language is to better support the modeling and analysis of CPS system architecture, through the extension of AADL behavior attachment, it supports the discrete behavior of CPS. Modelica [3] is an open, object-oriented and equation based computer language, which can span different fields and conveniently realize the modeling of complex physical systems. It can well describe the continuous behavior of CPS. The system can not meet the development requirements of complex key systems, which makes software safety accidents occur frequently. The reliability [4] of the system has become the focus of embedded system development.

In recent years, scholars have made further research on CPS. Reference [5] proposed a modeling form of dynamic network of stochastic hybrid automata, which allows natural modeling of concepts. By extending finite state automata, reference [6] improved stochastic hybrid automata (SHA) and proposed a method to represent complex systems. Reference [7] proposed an uncertainty analysis method to evaluate the reliability of the system, and applied the fault tree [8] to evaluate the reliability and importance of performance. In this paper, the discrete and continuous behavior of the system is described by using the characteristics of hybrid automata, the information system is described by
AADL, the physical system is described by Modelica, and the CPS system is formally modeled. The two attributes of randomness and fault analysis are extended on the AADL behavior attachment, the fault analysis stochastic hybrid automata is proposed, the transformation rules and algorithms of the model are given by combining AADL and Modelica, and the correctness of the transformation rules is verified by bisimulation equivalence [9][10]. This chapter introduces the relevant background and work. The second chapter will introduce the combination of fault analysis stochastic AADL and Modelica. The third chapter will introduce the rule algorithm of transforming AADL and Modelica into automata. The fourth chapter will introduce case study, converts AADL and Modelica into automata, and combines them into automata, and carries out fault analysis. The fifth chapter makes a summary.

2. Combination of fault analysis stochastic AADL and Modelica

2.1. Fault analysis stochastic behavior properties

CPS is closely related to people's daily life, which may have uncertain behavior, run-time failures and other issues. This paper introduces AADL behavior attachments, Modelica and fault tree analysis [11] to model and describe. AADL behavior attachments are sublanguages of AADL and it can be connected to AADL components such as subroutines, threads.

To clarify how to use fault analysis stochastic behavior attachments, we model the internal behavior of components in the form of state machines using AADL behavior attachments. Fault -analysis stochastic behavior attachment language is mainly composed of variables, states, faultStates and transitions.

FASBA ::= { **
  variables
  states
  faultStates
  transitions
 **} where

- variables defines a local variable within an attachment to an analytic stochastic behavior that describes the nature of the component at a given time.
- states defines the normal state that occurs in the system, including the set of states that exist in the system and the initial state of the system. The syntax of the state is defined as follows:
  
  states ::= {states, initial_states};

- faultStates defines the failure state that occurs in the system.
- transitions defines the transition relationship of states in a system. When a system is in a state, it jumps to the corresponding subsequent state when the corresponding conditions are met. The definition of transition relationship is as follows:
  
  transitions ::= {state-[guard] -> state{action, resetVariables}}, guard is a jump guard that can enter the next state when there is a state that meets the requirements. It can be a Boolean expression of a variable in the current state or an input or output action performed by the current state. The syntax of guard is defined as follows:
  
  guard ::= data_expression | input/output_action

  data_expression ::= data_communication

  input/output_action ::= action

where data_communications cause state transition because the values of variables within a state satisfy predicate conditional constraints. Actions represent actions of input and output, triggering conditions and changing states when actions occur. ResetVariables is an assignment expression for reset variables whose state has been migrated. The syntax of resetVariables is defined as follows:

  resetVariables ::= variable = k, where k is a real constant.

Considering the practical problems of CPS, in this paper, two attributes, randomness and fault analysis, are added to the transition state. AADL behavior attachments can be used to describe the attributes of uncertainty behavior and failure probability in the process of state transition, respectively. The fault
analysis stochastic transition (FAStransitions) process syntax is defined as follows:

\[
\text{FAStransitions} ::= \{\text{state}\{\text{guard}\} \rightarrow \text{stacho}, \text{weight}\{\text{action}, \text{resetVariables}\}\}\}
\]

where stacho is the probability of randomness in the process of state transition, stacho \(\in [0, 1]\) \(\sum \text{stacho}_j = 1\), where \(i\) is a fixed state, that is, the target state, and \(j\) is an arbitrary state. Weight is the weight of fault occurrence probability in state transition, which is determined by the transition.

2.2. Extending the attribute sets corresponding to the Modelica element on AADL

In CPS, there are not only discrete state transitions, but also continuous changes of variables in each state. These continuous changes of variables usually occur in physical system and described by differential equations or flow conditions. The elements between AADL and Modelica [12] do not correspond one to one. To solve this problem, this section uses the AADL built-in data type aadlstring to extend the attribute set in the AADL attribute set Modelica_property add variables, initial values, constants, constant values, equations and other attributes to property, and then add the extended attribute set to the hardware component through the ‘applies to’ clause. The extended AADL attribute sets are as shown in table 1:

| The extend AADL attribute |
|---------------------------|
| property set M_property is |
| initialValues: aadlstring applies to (device); |
| variables: aadlstring applies to (device); |
| constantValues: aadlstring applies to (device); |
| equations: aadlstring applies to (device); |
| end M_property; |

Table 1. The extend AADL attribute.

3. Transformation from combination of AADL and Modelica to automata

3.1. Fault analysis stochastic hybrid automata

This paper extend the two attributes of fault analysis and randomness on the basis of traditional hybrid automata. A new automata is proposed, which is defined as follows:

**Definition 1.** (fault analysis stochastic hybrid automata, FASHA): An fault analysis stochastic hybrid automata is a tuple FASHA = \((S, ES, FT, E, AC, X, A, P, W, s_0, x_0, P_0)\), where:

1. \(S\): a finite set of discrete states \(\{s^1, s^2, \ldots, s^m\}\);
2. \(ES\): the state in which a fault occurs \(\{es_1, es_2, \ldots, es_m\}\);
3. \(FT\): a fault tree. Each fault state is composed of fault variables in the fault tree. Different variable combinations will lead to different faults.
4. \(E\): a finite set of deterministic events or stochastic events \(\{e_1, \ldots, e_r\}\);
5. \(X\): a set of finite real variables varying with time \(\{x_1, \ldots, x_n\}\), variable \(x_i\), the vector form is expressed as \(x = [x_1, \ldots, x_n]^T\);
6. \(A\): a finite set of arcs in the form of \((s, e_j, G_k, R_k, s')\), where \(s\) and \(s'\) are the discrete states of the starting point and target of arc \(k\), respectively \(e_j\) is the event connecting the arc, \(G_k\) is the guard condition of state \(x\) in \(s'\), \(R_k\) is the reset function of \(x\) in \(s'\) state;
7. \(A_c\): \(X \times S \rightarrow (\mathbb{R}^n^+ \rightarrow \mathbb{R})\) is the ‘activity’ function, which is associated with \(X \times S\) element is associated and is a function defined on \(\mathbb{R}^n^+\) and adopts;
8. \(P: P = [p^j_i]\) a matrix of discrete probability distributions, where \(p^j_i\) is the discrete distribution of probability from \(s^j\) state to \(s^i\) state \(p(s^i \mid s^j, e)\); For example, let \(e_0\) be the same event that defines the transition from discrete state \(s^i\) to discrete state \(s^j\), \(s^2, \ldots, s^j\) (therefore, there are \(j\) conversion conflicts, and the bottom layer of FSA is not deterministic), making \(p^i_1\) is the probability of transition from state \(s^1\) to state \(s^3\), \(p^i_2\) is the probability of transition from state
to state \( s^2 \), and the probability of \( p^l_j \) transition from state \( s^l \) to state \( s^l \), where \( p^l_1+p^l_2+\cdots+p^l_j=1 \);
(9) \( W \): weighted label function \( S \times A \times S' \rightarrow \mathbb{R} \), add weights for each fault state;
(10) \( s_0, x_0 \) and \( P_0 \) respectively correspond to the initial discrete state, the initial value of the continuous state vector in the discrete initial state and the initial distribution of the transition probability;

3.2. Transformation algorithm from combination of AADL and Modelica to automata
AADL and Modelica both are semiformal modeling language, the fault analysis stochastic behavior attachment of AADL and the extended attribute set of Modelica can be transformed into the corresponding fault analysis stochastic hybrid automata \( \text{FASHA} = (S, ES, FT, E, A_C, X, A, P, W, s_0, x_0, P_0) \), the conversion rules are shown below:
(1) The ‘variables’ module in the behavior attachment is used to describe the variable set in the information system, which corresponds to the \( X \) variable set in \( \text{FASHA} \);
(2) The ‘states’ module in the behavior attachment represents various normal operating states in the system, corresponding to the \( S \) state set in \( \text{FASHA} \) and the initial_state in ‘states’ corresponds to \( s_0 \) in the automata, and a fault tree is added to the attachment added by the modeling language combined with AADL and Modelica. The faultStates module in the attachment represents various possible fault states of the system and corresponds to the fault state \( ES \) state set in \( \text{FASHA} \);
(3) The ‘transitions’ module in the behavior attachment corresponds to multiple other components in the automata. The ‘transitions’ set corresponds to set \( A \) in the automata, where the ‘guard’ condition in the ‘transitions’ module corresponds to \( G_k \) in \( A \). The ‘state’ contains \( s \) and \( s' \) respectively, and ‘action’ corresponds to set \( E \) in automata, stacho corresponds to the probability matrix in automata \( P, \omega \) corresponding to the fault analysis tag function \( W \), ‘resetVariables’ corresponds to \( R_k \) in \( A \).
(4) The ‘initialValues’ in the attribute set of Modelica extension correspond to \( x_0 \) in the automata, and the equations in a state correspond to the function \( A_c \) in the automata;

Algorithm 1 transformation algorithm of AADL and Modelica combined model (pseudo code)

ALGORITHM 1: AADL & Modelica model convert to FASHA

**Input:** AADL fault analysis stochastic behavior annex: \( \text{FASBA} := (\text{variables}, \text{states}, \text{faultStates}, \text{transitions}) \)

Modelica extended property set: \( \text{Ms} := (\text{initialValues}, \text{constantValues}, \text{equations}) \)

**Output:** fault analysis stochastic hybrid automata \( \text{FASHA} := (S, ES, FT, E, A_C, X, A, P, W, s_0, x_0, P_0) \)

1: **Begin**
2: \( \text{FASHA}.S := \text{FASBA}.states; \)
3: \( \text{FASHA}.s_0 := \text{FASBA}.initial_states; \)
4: \( \text{FASHA}.X := \text{FASBA}.variables; \)
5: For transitions in \( \text{FASBA}.transitions \) do
6: \( \text{State sourceState} := \text{transitions.getSourceState}(); \)
7: \( \text{State destinationStates} := \text{transition.getDestinationState}(); \)
8: \( s := \text{sourceState}; \)
9: \( s' := \text{destinationStates}; \)
10: If \( \text{transitions.gaurd} \) is not NULL then
11:   For destinationState in destinationStates
12:     \( G_k(s) := \text{destinationState.action}; \)
13:     \( R_k(s) := \text{destinationState.resetVariables}; \)
14: Event e := transition. s'.action;
15: Stacho sto(s, e_j, s') := transition. s'.sto;
16: Weight \( \omega(s, e_j, s') := \text{transition. s'.\omega}; \)
17: Boolean_expression \( G_k := \text{transition. s'.\guard}; \)
18: Assignment_expression \( R_k := \text{transition.s'.resetVariables}; \)
19: FASHA.A := A \cup (s, e_j, G_k, R_k, s', \omega);
20: FASHA.E := E \cup e_j;
21: FASHA.Stacho := stacho \cup \{sto(s, e_j, s')\};
22: FASHA.W := W \cup \{\omega(s, e_j, s')\};
23: If transitions.guard is null and F is not null then
24: faultStates := faultStates \cup FT(s)
25: FASHA.x_0 := Ms.initialValues;
26: For state in FASHA
27: Function A_c := Ms.equations(state);

Algorithm 1 is a model combining AADL and Modelica transformed into a blending automaton by a rule transformation. The input is the AADL fault analysis stochastic behavior annex and the extended property set of Modelica, and the output the fault analysis stochastic hybrid automaton. Rows 2-4 of the algorithm correspond to the state, initial state and variables of FASHA to the attachments in FASBA. Rows 5-13 are based on the ‘transitions’ process, changing the sanitary conditions and reset functions from the source state to the target state. Rows 14-18 are actions, stochastic conditions, weights, and reset variables in the transition process. Rows 19-22 add the transformed attributes back to the original set. Rows 23-24 are the judgement of failure state and will produce a set of failure states combined; Rows 25-27 are based on Modelica's extended set of attributes, in the initial state x_0 corresponds to the change of ‘equations’ in the case of ‘initialValues’ when the state changes.

Bisimulation equivalence is that two systems can imitate each other's behavior and reflect the relationship between the behaviors of the two systems in the objective world from a certain side. The above conversion algorithm is based on the proof of mutual simulation equivalence relationship, which ensures the correctness of the model and keeps the behavior unchanged.

4. Case study
Fire control system is a complete set of equipment to control the aiming and firing of artillery, artillery group or missile launcher. It guides the aircraft to the target area, searches, approaches, identifies and tracks the target, measures the motion parameters of the target and the carrier aircraft, carries out fire control calculation, controls the firing mode, quantity and fuse setting of weapons.

4.1. The information component modeled by AADL

![Figure 1. FASHA motivating example automata.](image)
As shown in Figure 1, this section chooses to study the target tracking and detection system in the fire control system. The following examples include parameters such as distance \(d\), azimuth \(\text{pos}\), speed \(v\), raw rate \(\text{raw}\), etc. It also includes a series of states such as initial off state, open state on, calculation state compute, ready state, error state, etc. And guard conditional modes from state off to on modes\_OffToOn, weapon A/B guard condition compute\_A/B. Weapon A/B select guard condition choose\_A/B. Launch confirmation guard condition fire\_Check, run the fault guard condition runError, calculate the fault guard condition computeError, machine fault guard condition deviceError and other guard conditions that cause action transition. Among \(\omega_r\), \(\omega_c\), \(\omega_d\) represents the probability of occurrence of the above three faults. AADL is used to model the target tracking and detection system, as shown in table 2.

Table 2. The information component modeled by AADL.

```plaintext
process controller
  features
    t_distance: in data port;
    t_position: in data port;
    t_velocity: in data port;
    t_p_raw_rate: out data port;
  end controller;
process implementation controller.impl
  annex weight_stochastic_behavior{**
    variables:
      d: Base_Type::Unsigned_32;
      t: Base_Type::Unsigned_32;
      pos: Base_Type::Unsigned_32;
      v: Base_Type::Unsigned_32;
      raw: Base_Type::Unsigned_32;
    states:
      off :initial state;
      on, compute, ready, fire, error: states;
      modes_offToOn, compute_A/B, choose_A/ B, fire_check, runError,
      computeError, deviceError: guards;
    transitions:
      off -[modes_offToOn] -> on;
      on -[compute_A/B] -> compute\{v = required_A/B_velocity – actual_velocity,
      raw = compute_raw(pos, pit, h)\};
      compute -[choose_A/B] -> ready;
      ready -[fire_check] -> fire;
      on -[runError] error : \(\omega_r\);
      compute -[computeError] error : \(\omega_c\);
      fire -[deviceError] error : \(\omega_d\);
  **}
end controller.impl;
```

4.2. The physics component modeled by Modelica

The ranging sensor, azimuth meter, sensing gyroscope, velocity sensor, accelerometer and other equipment in the target detection and tracking system are regarded as the physical system in CPS. The constants \(d\), \(\text{pos}\), \(v\) and \(\text{raw}\) defined in this example represent the distance between the target and the weapon carrier, the position, speed and range change rate of the target respectively. When the system is running, the control system starts calculation and completes a series of aiming, calculation, strike and so on. Firstly, set the parameters, set \(d\), \(\text{pos}\), \(\text{pit}\) and other parameters to 0 or null respectively, set an acceleration constant \(a\), bring the speed into the whole calculation program, and get the desired result through a series of operations. As shown in table 3 below, Modelica models the physical part.
4.3. Transition from Modelica and AADL to FASHA
The Modelica model is converted to the component of AADL that extends the attribute set. The Modelica model is converted to the hardware device in AADL: the device enters the device through the data detected from the external environment in data port, and then is processed and sent to the controller in the form of out data port for data calculation. Variables, initial variable values, constants, constant values, and equations in Modelica are converted to the extended attribute set in AADL. The converted components are shown in Table 4.

The fault tree is introduced into the hybrid automata, as shown in Figure 1: this figure is an example of FASHA. The automata is simply established based on the process of gun launched missile launch system.

The branches in the error state, the fault tree after fault tree analysis and randomness attribute addition, are simply presented. The three fault states are launching-system-error, combat-unit-error and main-unit-error. Where \( p_1, p_2, p_3 \) is the probability of stochastic occurrence of the behavior; \( \omega \) is the weight of failure during state transition, indicating the probability of failure during state transition, e.g: \( \omega_{la} = p_1 \times p_{la} \cdot F_1 \cdot F_2 \cdot F_3 \) is the combination of fault state variables. The accidental ignition fault of the launch system includes three forms of fault problems, namely primer fault, ignition fault and launch fault, \( F_{p,r}, F_{i,g}, F_{la} \) respectively corresponds to the variable combinations of the above three states, and different combinations of variables cause different faults. The accidental ignition fault of the launch system and fuse fault are introduced in detail in Fig. 2(a)(b), eg: \( F_{la} = (X_1 \cdot \neg X_2 + X_1 \cdot \neg X_3) \cdot (X_4 \oplus X_5) \). The transmission fault is composed of different variables, where \( X_i \) indicates that

| Table 3. The physical component modeled by AADL. |
|-----------------------------------------------|
| model launcher                                |
| import Modelica.Blocks.Interfaces.BooleanInput; |
| import Modelica.Blocks.Interfaces.RealInput1;  |
| import Modelica.Blocks.Interfaces.RealInput2;  |
| import Modelica.Blocks.Interfaces.RealInput3;  |
| import Modelica.Blocks.Interfaces.RealOutput1; |
| BooleanInput fire_check = true;               |
| RealInput distance;                           |
| RealInput position;                           |
| RealInput velocity;                           |
| RealOutput raw_rate;                          |
| Real d = 0;                                   |
| Real pos = null;                              |
| Real v = 0;                                   |
| Real raw = 0;                                 |
| Parameter a = 20;                             |
| equation                                      |
| when ready = true then                        |
|   d = 0;                                      |
|   distance = d;                               |
|   \( \dot{v} = a; \)                         |
|   velocity = v;                               |
|   raw = compute_raw(pos, h);                  |
| end when;                                     |
| when ready = false then                       |
|   continue until ready = true;                |
| end when;                                     |
| end launcher;                                 |
the switch is normally open, $X_2$ indicates normal calculation, $X_3$ indicates normal aiming, $X_4$ indicates normal operation of the system, $X_5$ indicates that the equipment operates normally, and the combination of these variables leads to failure. $F_{ff} = (X_6 \oplus X_7) \cdot (\neg X_8 + X_9)$, where $X_6$ indicates that the needle striking safety mechanism is normal, $X_7$ indicates that the slider safety mechanism is normal, $X_8$ indicates that the fuse components are normal, $X_9$ indicates detonation sequence failure.

**Table 4.** Transition from Modelica and AADL to FASHA.

```markdown
with M_property
device launcher
features
t_distance: in data port;
t_position: in data port;
t_velocity: in data port;
t_p_raw_rate: out data port;
properties
M_property:: constantVariables => "a";
M_property:: constantValues => "20";
M_property:: variables => "d, pos, pit, h, v, ran, raw";
M_property:: initialValues => "0, null, 0, 0";
M_property:: equations => "
  when ready = true then
    d = 0;
    distance = d;
    v = a;
    velocity = v;
    raw = compute_raw(pos, h);
  end when;
  when ready = false then
    continue until ready = true;
end when;"
```

**Figure 2.** (a) Fault tree of launch-error (b) Fault tree of front-fuse-error.

### 4.4. Fault analysis

In engineering practice, fault mode [14] and normal working mode analysis is an important work for multi-component complex systems. Fault tree analysis is recognized as a good method to analyze the safety and reliability of complex systems. We quote the calculation formulas of importance and probability in reference [15], and conduct fault tree analysis based on Figure 2. The failure probability table is shown in table 5, which $P_{xi}$ represents the probability of failure.
Table 5. Probability of failure states.

| id   | basic event name                        | number of failures | $P_{x_i}$    |
|------|----------------------------------------|--------------------|--------------|
| X_1  | Switch opens normally                  | 0                  | 0            |
| X_2  | Normal calculation                      | 1                  | $10^{-5}$    |
| X_3  | Aiming normally                         | 2                  | $10^{-4}$    |
| X_4  | System operates normally                | 1                  | $5 \times 10^{-4}$ |
| X_5  | Normal operation of device              | 1                  | $5 \times 10^{-4}$ |
| X_6  | Needle striking safety mechanism        | 1                  | $5 \times 10^{-4}$ |
| X_7  | Slider safety mechanism normally        | 1                  | $5 \times 10^{-4}$ |
| X_8  | Fuse components normally                | 1                  | $5 \times 10^{-4}$ |
| X_9  | Detonation sequence failure             | 3                  | $10^{-3}$    |

5. Conclusion

In this paper, the definition of AADL fault analysis stochastic behavior attachment is proposed, and the attribute of AADL attachment is extended to generate the corresponding Modelica extended attribute set. Due to the limitations of semiformal language, this paper combines fault tree with AADL and Modelica, proposes a formal model, analyzes errors, randomly hybrid automata, gives a model conversion algorithm, and proves the correctness of model conversion through bisimulation. In the future work, we hope to deeply analyze and confirm the algorithm based on the existing formal model.

Acknowledgments

This work was supported in part by the Aviation Science Foundation of China under Grant 20185152035, in part by the National Natural Science Foundation of China under Grant 61572253, and in part by the Fundamental Research Funds for the Central Universities (NJ2020022, NJ2019010).

References

[1] M S Edgar and J G Ricardo, 2021, “Cyber-Physical Systems: a multi-criteria assessment for Internet-of-Things (IoT) systems,” Enterprise Information Systems, vol. 15, no. 2, pp. 332-351.
[2] H Mkaouar, B Zalila, J Hugues and Jmaiel M, 2021, “A formal approach to AADL model-based software engineering,” International Journal on Software Tools for Technology Transfer, vol. 22, no. 2, pp. 219-247.
[3] Fei Deng, Yunqiang Yan, Feng Gao and Linbo Wu, “Modeling and Simulation of CPS based on SysML and Modelica (S),” Lisbon, Portugal, ISBN, The 31st International Conference on Software Engineering and Knowledge Engineering, pp. 15-25, July 2019.
[4] Li Dongmin, Li Jing and Lin Huafeng, 2017, “Reliability analysis method of AADL model of embedded system based on fault tree analysis,” Computer science, vol. 44, no. 06, pp. 182-188.
[5] G. Hequet, N. Brinzei and J. F. Pépin, “Usage profile in physical systems modeled with stochastic hybrid automata,” Žilina, Slovakia, ISBN, International Conference on Information and Digital Technologies (IDT), pp. 220-229, June 2021.
[6] G Castaneda, J F Aubry and N Brinzei, 2011, “Stochastic hybrid automata model for dynamic reliability assessment,” Proceedings of the Institution of Mechanical Engineers Part O Journal of Risk and Reliability, vol. 225, no. 01, pp. 28-41.
[7] D Alexandre, G L Kim, L Axel and B P Danny, 2014, “Statistical Model Checking of Dynamic Networks of Stochastic Hybrid Automata,” Electronic Communication of the European Association of Software Science and Technology, vol. 66, pp. 91-104.
[8] Xia Yu, Zhao Heming, Zhang Hongtao, Yang Yu and Li Shaoqing, 2021, “Fault tree analysis of gun launched missile system,” Science technology and engineering, vol. 21, no. 14, pp. 5789-5795.
[9] Changyan Zhou, R. Kumar and Shengbing Jiang, 2006, “Control of nondeterministic discrete-event systems for bisimulation equivalence,” *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 754-765.

[10] J Petr, 2020, “Deciding semantic finiteness of pushdown processes and first-order grammars w.r.t. bisimulation equivalence,” *Journal of Computer and System Sciences*, vol. 109, pp. 22-44.

[11] G T Michael, E W Page, “Boolean difference techniques in fault tree analysis,” 1976, *International Journal of Parallel Programming*, vol. 5, no. 1, pp. 81-88.

[12] S Anton, S Igor, Z Borut, 2020, “Cyber-physical modelling in Modelica with model-reduction techniques,” *Journal of Systems and Software*, vol. 163, p. 110517.

[13] Kaiyi Qian, Long Yu and Shbin Gao, 2021, “Fault Tree Construction Model Based on Association Analysis for Railway Overhead Contact System,” *International Journal of Computational Intelligence Systems*, vol. 14, no. 1, pp. 96-105.

[14] C. Zhou, Q. Chang, H. Zhao, M. Ji and Z. Shi, 2021, “Fault Tree Analysis With Interval Uncertainty: A Case Study of the Aircraft Flap Mechanism,” *IEEE Transactions on Reliability*, vol. 70, no. 3, pp. 944-956.

[15] M. Barrère and C. Hankin, “Fault Tree Analysis: Identifying Maximum Probability Minimal Cut Sets with MaxSAT,” Valencia, Spain, IEEE, 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks, pp. 53-54, June 2020.