No. 125/23, 7–24
ISSN 2657-6988 (online)
ISSN 2657-5841 (printed)
DOI: 10.26408/125.01

Submitted: 23.11.2022
Accepted: 19.01.2023
Published: 31.03.2023

FORMAL VERIFICATION OF STPA WITH MODEL CHECKING

Ryeonggu Kwon[1]*, Gihwon Kwon[2]

[1,2] Kyonggi University, 6, Seowon 4-gil, Gwanak-gu, Seoul, Republic of Korea, Department of Computer Science,

[1] e-mail: rkkwon@kyonggi.ac.kr, ORCID 0000-0002-4942-247X

[2] ORCID 0000-0002-8221-4939

* Corresponding author

Abstract: As technology advances, hardware-centric systems are rapidly moving towards software-centric ones, and their complexity is rapidly increasing. In particular, systems directly related to safety require thorough verification. Model checking exhaustively explores the state space of the abstracted system to check whether properties written in a logical formula are achieved. In this paper, the control algorithm of the controller is verified using model checking to discover risk scenarios during the STPA steps. Two case studies are conducted using the widely used model checkers NuSMV and UPPAAL. We then explain the empirical results and compare two model checkers based on their characteristics. Finally, we discuss the benefits of applying model checking in the process of STPA.

Keywords: formal verification, model checking, STPA.

1. INTRODUCTION

With the advancement of technology, systems are rapidly changing from hardware-centric to software-centric. In addition, as control and interaction between components become very important in a software-oriented system, and the risk of the system is directly related to safety, the importance of a new risk analysis technique has increased. As the existing risk analysis technique has limitations in applying it to software-oriented systems, an efficient risk analysis technique was needed to overcome this situation, therefore STPA (System-Theoretic Process Analysis) was developed [Leveson and Thomas 2018]. However, in order to perform the process of STPA, human intervention such as a safety analyst is required, many parts must be analysed manually, and a lot of effort is required when deriving a risk scenario for STPA. In particular, when deriving a risk scenario, a method of building a context table of a process model is used. However, this also required human intervention, and other systematically organised methods are needed to achieve completeness. As such a systematic method, model checking [Baier, Katoen and Larsen 2014], which is one of the representative techniques of formal verification, may be used. In this paper, model checking is used to verify that the control algorithm of the controller sufficiently achieves the safety constraint in the process of STPA. Once a counterexample is obtained as a result of model checking, it is used to identify risk scenarios in which safety constraints are not achieved and to refine the control algorithm.

For model checking, NuSMV and UPPAAL, which are widely used in academia and industry, were selected. The structure of this paper is as follows. Sections 2 and 3 introduce related studies applying model checking to backgrounds and STPA. Section 4 details research interests and how to apply model checking to STPA. Section 5 shows how to build a model and write safety properties using each model checker for a case study of the Door Interlock System and Auto-Hold System. Section 6 discusses the empirical results of carrying out case studies and explains the characteristics of each model checker and what roles and boundaries it has when used in STPA. In addition, the advantages of the proposed method when applied to STPA are explained. The last section presents a summary of the paper and future research.

2. BACKGROUNDS

2.1. STPA

Recently, as the limitations of existing risk analysis methods have been revealed, a method that can analyse risks from a new perspective is needed, due to the increasing complexity of the software in the whole system. It has become difficult to limit the factors that can cause accidents to just a specific component problem. In addition, if you look at the recent accident patterns, not only the system but also various external factors (people, policies, environment, etc.) cause accidents. Accordingly, in 2012, STPA, a risk analysis method with a new perspective different from the existing risk analysis method, was announced. STPA basically believes that accidents are caused by control problems between systems or components rather than a failure of specific components (Component Failure).

Therefore, when STPA analyses the system, rather than listing and combining all components (or functions), the system is structured and understood around important control relationships that affect safety. Therefore, by using STPA, it is easy to understand and perform risk analyses by abstracting a complex system composed of numerous components.

2.2. Model checking

Model checking is one of the formal techniques that abstracts the system to be verified as states and transitions, expresses the properties to be achieved in a logical formula, and checks whether there are any violating properties. In general, the abstract model is built with a finite state machine, and the properties are written in a logical formula such as CTL (computation tree logic), LTL (linear temporal logic) or TCTL (timed computation tree logic). The model checker accepts both the model and the property as inputs and checks whether the property is satisfied in the model by thoroughly examining the state space. If the property is not satisfied in the model, it provides a counterexample (or the result of the simulation). By analysing the traces of counterexamples, we can find cases where the property is violated, and the property can be achieved by modifying the model.

3. RELATED WORKS

[Tsuji et al. 2020] introduce a method to prioritise hazardous scenarios identified by STAMP/STPA with the help of a statistical model checking technique. It shows a procedure for systematically transforming the model defined by STAMP/STPA to a formal model for a statistical model checking tool. It represents scenarios in a formal model and calculates the probabilities by using statistical model checking. [de Souza et al., 2020] introduce a method that combines STPA and SysML modelling activities to provide simulation and formal verification of systems' models. It makes it possible to develop and verify the system in a more systematic manner, taking advantage of the integration of TTool and UPPAAL. It translates the STPA safety requirements into properties to be verified by UPPAAL from TTool.

Two challenges are presented in this paper. The first is related to SysML modelling and, more specifically, to the elaboration of the state machine diagrams of the components. The second is to map the STPA safe requirements into properties in TTool/UPPAAL. The future research of this paper is to deal with more complex scenarios and degradation situations, and to develop automation tools. [Zhong et al. 2022] propose to build a model in SysML, describe the timing with MARTE, transform the SysML model into a NuSMV model, and output loss scenarios automatically with a model checker.

There are two advantages to this method. The first is that the loss scenario can be generated automatically. The second is better collaboration with SysML-based engineering. It also introduces three disadvantages. The first is that it is not sufficient to express continuous behaviour. The second, it is not suitable for dealing with extremely complex systems. The third is that a safety constraint must be converted into temporal logic for the identification of the loss scenario. As a direction for future research, the following are suggested. The first is to formalise time-dependent

UCAs, and the second is to handle the automatic conversion between SysML and NuSMV models.

[Dakwat and Villani 2018] introduce a method for combining STPA and model checking, in order to provide a formal and unambiguous representation of the system under analysis and the threats identified by STPA. [Abdulkhaleq, Wagner and Leveson 2015] introduce a comprehensive safety engineering approach based on STPA, including software testing and model checking approaches for the purpose of developing safe software. It highlights the advantages of applying STPA to software at the system level to identify potentially unsafe control actions of software and to derive the corresponding safety requirements that prevent software from transitioning into a hazardous state. Its limitations are that the main steps require manual interventions performed by the safety analyst and the difficulty of using formal verification in practice.

4. PROPOSED METHOD

This section describes how to apply formal verification to STPA. The focus of our research is derivation of the hazard scenario, which is the last stage of STPA. The following Figure 1 is an overview of the method proposed in this paper. Before we go into detail, the assumptions of this process are:

- the safety analyst provides the results of the 3rd step of STPA;
- the system engineer identifies the minimum achievable functional requirements.

Under this assumption, the system to be verified is modelled through the model checker. The model obtained through this process is regarded as the control algorithm and process model of the controller in the control structure.

After that, the unsafe control actions identified through step 3 of STPA are written as safety constraints through CTL, LTL, and TCTL. The control algorithm and safety properties are input to the model checker, and after checking the result, if the verification fails, a counterexample is obtained.

The counterexample refines the model so that the desired safety property can be achieved in the model. A model that achieves all safety constraints can be obtained by repeatedly performing the process of re-verification of the refined model.

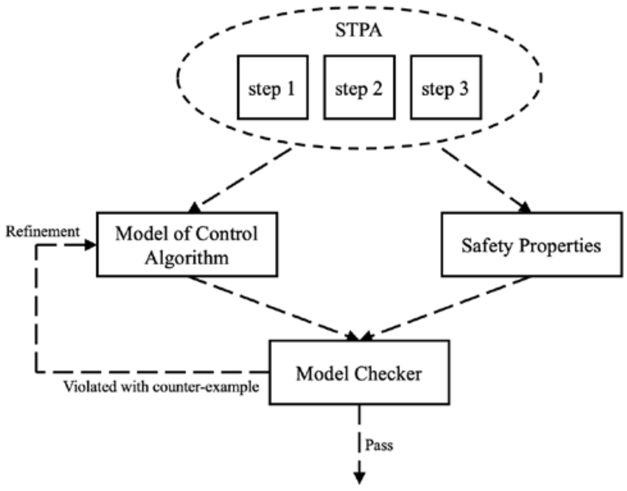


Fig. 1. Overview of proposed method

Source: our own study.

5. CASE STUDIES

In this section, we apply the previously introduced method to two cases. The first case is the Door Interlock System, and the second is the Auto-Hold System of an automotive vehicle. NuSMV and UPPAAL were selected as model checkers for creating the control algorithms, process models, and safety properties of each system. It shows the resources acquired before proceeding with each case, and you can check how the control algorithm and process model are modelled through each model checker, and how the safety properties are written.

5.1. Door Interlock System

A Door Interlock System [Leveson 2011] is a system to block a specific space through which high-voltage current flows. When a person opens a door to enter a space exposed to high-voltage current, the high-voltage current is cut off to prevent people from being exposed to this high-voltage current. The system consists of two subsystems – a power controller and a power source – which can be operated by a human operator. A human operator can request a command to open or close a door, and the power controller must disconnect or connect the power appropriately, as requested. And the power controller can use a sensor to determine if the door is open or completely closed.

The hazardous system behaviours identified in the Door Interlock System are listed in Table 1.

Table 1. Hazardous System Behaviours for Door Interlock System

Control action	Not providing causes hazard	Providing causes hazard	Wrong timing or order causes hazard	Stopped too soon or applied too long
Power off	Power not turned off when door opened		Door opened, controller waits too long to turn off power	
Power on		Power turned on while door opened	Power turned on too early; door not fully closed	

Source: [Leveson 2011].

Tables [2, 3] show the results of modelling the Door Interlock System in each model checker. Only the core part of the model is included, not the whole model, due to the space requirements of the paper.

Table 2. Models for Door Interlock System using NuSMV

Process model for Power Controller	

door_pos : 0..2;
power_st : 0..1;
actions : {nothing, open_door, close_door, turn_off_power, turn_on_power, push};	

door_pos	A variable indicating whether a door is fully closed, ajar, or fully open
power_st	A variable indicating whether power is disconnected or connected
actions	A variable representing actions that can occur in the entire system
Control Algorithm for Power Controller	
(i = 0 & next(actions) = push -> next(i) = 1) & (i = 1 & door_pos = 0 & next(actions) = turn_off_power & next(power_st) = 0 -> next(i) = 2) & (i = 2 & next(actions) = open_door -> next(i) = 3) & (i = 3 & next(actions) = push -> next(i) = 1) & (i = 1 & door_pos = 2 & next(actions) = close_door -> next(i) = 4) & (i = 4 & door_pos = 0 & next(actions) = turn_on_power & next(power_st) = 1 -> next(i) = 0);	
i	A variable to indicate the states of FSM
Model for Human Operator	
(door_pos = 0 | door_pos = 2 -> next(actions) = push);	

cont. Table 2

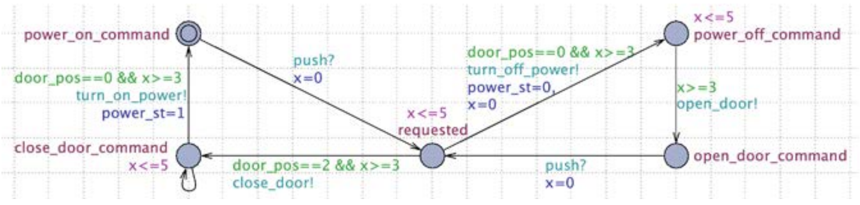
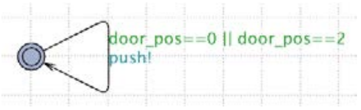
Model for Power Source	
(k = 0 & next(actions) = turn_off_power -> next(k) = 0) & (k = 0 & next(actions) = turn_on_power -> next(k) = 0) & (k = 0 & next(actions) = open_door & next(door_pos) = 1 -> next(k) = 1) & (k = 1 & next(door_pos) = 2 -> next(k) = 0) & (k = 0 & next(actions) = close_door & next(door_pos) = 1 -> next(k) = 2) & (k = 2 & next(door_pos) = 0 -> next(k) = 0);	
k	A variable to indicate the states of FSM

Source: our own study.

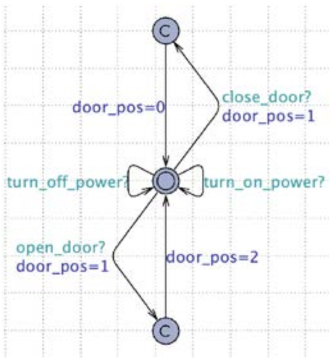
Table 3. Models for Door Interlock System using UPPAAL

Process model for Door Interlock System	

int[0, 2] door_pos = 0;
int[0, 1] power_st = 1;
chan open_door, close_door, turn_off_power, turn_on_power, push;	

door_pos	A variable indicating whether a door is fully closed, ajar, or fully open
power_st	A variable indicating whether power is disconnected or connected
channels	A variable representing actions that can occur in the entire system
Control Algorithm for Power Controller	
 power_on_command door_pos==0 && x>=3 turn_on_power! power_st=1 close_door_command x<=5 push? x=0 x<=5 requested door_pos==0 && x>=3 turn_off_power! power_st=0, x=0 x<=5 power_off_command x>=3 open_door! open_door_command door_pos==2 && x>=3 close_door! push? x=0	
Model for Human Operator	
	

cont. Table 3

Model for Power Source
 door_pos=0 close_door? door_pos=1 turn_off_power? turn_on_power? open_door? door_pos=1 door_pos=2 C C

Source: our own study.

Tables [4, 5] list the safety properties and shows the verification results of each model checker.

Table 4. Safety Properties and Results using NuSMV

SP1	CTLSPEC AG !(power_st = 1 & door_pos >= 1);	true
SP2	N/A	N/A
SP3	CTLSPEC AG !(actions != turn_on_power & door_pos >= 1);	true
SP4	LTLSPEC (actions != turn_on_power U door_pos != 0);	true

Source: own study.

Table 5. Safety Properties and Results using UPPAAL

SP1	A[]!(power_st==1 && door_pos>=1)	true
SP2	A[]!(x<3 && x>5 && door_pos>=1)	true
SP3	A[]!(PowerController.power_on_command && door_pos>=1)	true
SP4	A[]!(PowerController.power_on_command && door_pos!=0)	true

Source: our own study.

5.2. Auto-Hold System

An Auto-Hold System [Placke 2014] maintains a vehicle in a stopped state by providing appropriate pressure, even when the brake pedal is released, when the vehicle is completely stopped by pressing the brake pedal, until the accelerator is pressed. The components involved in this system are the driver, braking system, propulsion system and the auto hold module, which is our main concern. The driver can enable or disable auto hold and press or release the brake pedal. The auto hold module performs four actions (hold, additional pressure, release, apply parking brake) according to the driver's operation. The braking system mainly transmits the current brake pressure and wheel speed to the auto hold module, and the propulsion system sends information to the auto hold module when the driver accelerates or changes gears.

The hazardous system behaviours identified in the Door Interlock System are listed in Table 6.

Table 6. Hazardous System Behaviours for Auto-Hold System

Control action	**Not providing causes hazard**	**Providing causes hazard**	**Wrong timing or order causes hazard**	**Stopped too soon or applied too long**
HOLD	Not providing HOLD is hazardous if AH is active and the vehicle comes to rest with the brake pedal on	Providing HOLD is hazardous if the driver is applying the accelerator		Providing HOLD is hazardous if the driver has inactivated AH
		Providing HOLD is hazardous if AH is DISABLED		Providing HOLD is hazardous if there is sufficient wheel torque
		Providing HOLD is hazardous if AH is ENABLED and the vehicle is not at rest		Providing HOLD is hazardous if the required time at rest has not been met
ADDITIONAL_PRESSURE	Not providing ADDITIONAL_PRESSURE is hazardous if AH is in HOLD-MODE and the vehicle is slipping	Providing ADDITIONAL_PRESSURE is hazardous if AH is not in HOLD-MODE		
		Providing ADDITIONAL_PRESSURE		

cont. Table 6

		is hazardous if it exceeds the brake system specs		
RELEASE	Not providing RELEASE is hazardous if the driver has commanded sufficient wheel torque via the accelerator pedal	Providing RELEASE is hazardous if AH is in HOLD-MODE and the driver has not commanded sufficient wheel torque		Providing RELEASE before the there is sufficient wheel torque is hazardous
	Not providing RELEASE is hazardous if the driver DISABLES AH			
APPLY EPB	It is hazardous not to provide APPLY EPB if the driver has released AH w/o sufficient wheel torque or brake pedal pressure	It is hazardous for AH to provide APPLY EPB if AH is not in HOLDMODE		

Source: [Placke 2014].

Table [7, 8] shows the results of modelling the Auto-Hold System in each model checker. Only the core part of the model is included, not the whole model, due to the space requirements of the paper.

Table 7. Models for Auto-Hold System using NuSMV

Process model for Auto Hold Module	

mode : 0..2;
brake : 0..1;
gear : 0..4;
accel_pedal_level : 0..9;
brake_pressure_level : 0..9;
wheel_speed_level : 0..9;
actions : {nothing, enable_ah, disable_ah, accelerate, brake_pedal_on, brake_pedal_off, shift, ah_enabled, ah_disabled, hold, additional_pressure, release, brake_pedal_feel, brake_pressure, wheel_speed, accel_pedal_perc, PRNDL};	

Mode	A variable indicating that the feature is in Disable, Enable, or Hold mode.
Brake	A variable indicating whether the brake pedal was pressed.
gear	A variable that represents the current car gear.
accel_pedal_level	A variable that indicates how much the accelerator pedal is pressed.
brake_pressure_level	A variable that indicates the level of brake pressure.

cont. Table 7

wheel_speed_level	A variable indicating the level of rotation of the wheel.
actions	A variable representing actions that can occur in the entire system
Control Algorithm for Auto Hold Module	
(j = 0 & actions = enable_ah -> next(mode) = 1 & next(j) = 1) & (j = 1 & actions = disable_ah -> next(mode) = 0 & next(j) = 0) & (j = 2 & actions = PRNDL -> (next(gear) = 0 | next(gear) = 1 | next(gear) = 2 | next(gear) = 3 | next(gear) = 4) & next(j) = 0) & (j = 3 -> next(j) = 0) & (j = 4 & actions = accel_pedal_perc -> (next(accel_pedal_level) = 0 | next(accel_pedal_level) = 1 | next(accel_pedal_level) = 2 | next(accel_pedal_level) = 3 | next(accel_pedal_level) = 4 | next(accel_pedal_level) = 5 | next(accel_pedal_level) = 6 | next(accel_pedal_level) = 7 | next(accel_pedal_level) = 8 | next(accel_pedal_level) = 9) & next(j) = 0) & (j = 5 & actions = brake_pedal_on -> next(brake) = 1 & next(accel_pedal_level) = 0 & next(j) = 7) & (j = 6 -> next(j) = 1) & (j = 7 -> next(actions) = hold & next(mode) = 2 & next(j) = 8) & (j = 8 & actions = brake_pressure -> (next(brake_pressure_level) = 0 | next(brake_pressure_level) = 1 | next(brake_pressure_level) = 2 | next(brake_pressure_level) = 3 | next(brake_pressure_level) = 4 | next(brake_pressure_level) = 5 | next(brake_pressure_level) = 6 | next(brake_pressure_level) = 7 | next(brake_pressure_level) = 8 | next(brake_pressure_level) = 9) & next(j) = 9) & (j = 9 & actions = wheel_speed -> (next(wheel_speed_level) = 0 | next(wheel_speed_level) = 1 | next(wheel_speed_level) = 2 | next(wheel_speed_level) = 3 | next(wheel_speed_level) = 4 | next(wheel_speed_level) = 5 | next(wheel_speed_level) = 6 | next(wheel_speed_level) = 7 | next(wheel_speed_level) = 8 | next(wheel_speed_level) = 9) & next(j) = 10) & (j = 10 & wheel_speed_level >= 1 -> next(actions) = additional_pressure & (next(brake_pressure_level) = 0 | next(brake_pressure_level) = 1 | next(brake_pressure_level) = 2 | next(brake_pressure_level) = 3 | next(brake_pressure_level) = 4 | next(brake_pressure_level) = 5 | next(brake_pressure_level) = 6 | next(brake_pressure_level) = 7 | next(brake_pressure_level) = 8 | next(brake_pressure_level) = 9) & (next(wheel_speed_level) = 0 | next(wheel_speed_level) = 1 | next(wheel_speed_level) = 2 | next(wheel_speed_level) = 3 | next(wheel_speed_level) = 4 | next(wheel_speed_level) = 5 | next(wheel_speed_level) = 6 | next(wheel_speed_level) = 7 | next(wheel_speed_level) = 8 | next(wheel_speed_level) = 9) & next(actions) = additional_pressure & next(j) = 10) & (j = 11 & actions = accelerate -> next(mode) = 1 & next(j) = 12) & (j = 12 & actions = accel_pedal_perc -> (next(accel_pedal_level) = 0 | next(accel_pedal_level) = 1 | next(accel_pedal_level) = 2 | next(accel_pedal_level) = 3 | next(accel_pedal_level) = 4 | next(accel_pedal_level) = 5 | next(accel_pedal_level) = 6 | next(accel_pedal_level) = 7 | next(accel_pedal_level) = 8 | next(accel_pedal_level) = 9) & next(j) = 13) & (j = 13 & actions = accelerate -> next(j) = 12) & (j = 14 & actions = brake_pedal_on -> next(brake) = 1 & next(accel_pedal_level) = 0 & next(j) = 7);	
j	A variable to indicate the states of FSM
Model for Human Driver	
(i = 0 -> next(actions) = brake_pedal_on & next(i) = 1) & (i = 1 & actions = brake_pedal_feel -> next(i) = 0) & (i = 0 -> next(actions) = enable_ah & next(i) = 2) &	

cont. Table 7

(i = 2 -> next(i) = 0) & (i = 0 -> next(actions) = brake_pedal_off & next(i) = 3) & (i = 3 -> next(i) = 0) & (i = 0 -> next(actions) = disable_ah & next(i) = 4) & (i = 4 -> next(i) = 0) & (i = 0 -> next(actions) = accelerate & next(i) = 5) & (i = 5 -> next(i) = 0) & (i = 0 -> next(actions) = shift & next(i) = 6) & (i = 6 & actions = PRNDL -> next(i) = 0);	
i	A variable to indicate the states of FSM
Model for Braking System	
(k = 0 & actions = brake_pedal_on -> next(k) = 1) & (k = 1 -> next(actions) = brake_pedal_feel & next(k) = 0) & (k = 0 & actions = brake_pedal_off -> next(k) = 2) & (k = 2 -> next(k) = 0) & (k = 0 & actions = hold -> next(k) = 3) & (k = 3 -> next(actions) = brake_pressure & next(k) = 4) & (k = 4 -> next(actions) = wheel_speed & next(k) = 0) & (k = 0 & actions = additional_pressure -> next(k) = 5) & (k = 5 -> next(k) = 0);	
k	A variable to indicate the states of FSM
Model for Propulsion System	
(l = 0 & actions = accelerate -> next(l) = 1) & (l = 1 -> next(actions) = accel_pedal_perc & next(l) = 0) & (l = 0 & actions = shift -> next(l) = 2) & (l = 2 -> next(actions) = PRNDL & next(l) = 0);	
l	A variable to indicate the states of FSM

Source: our own study.

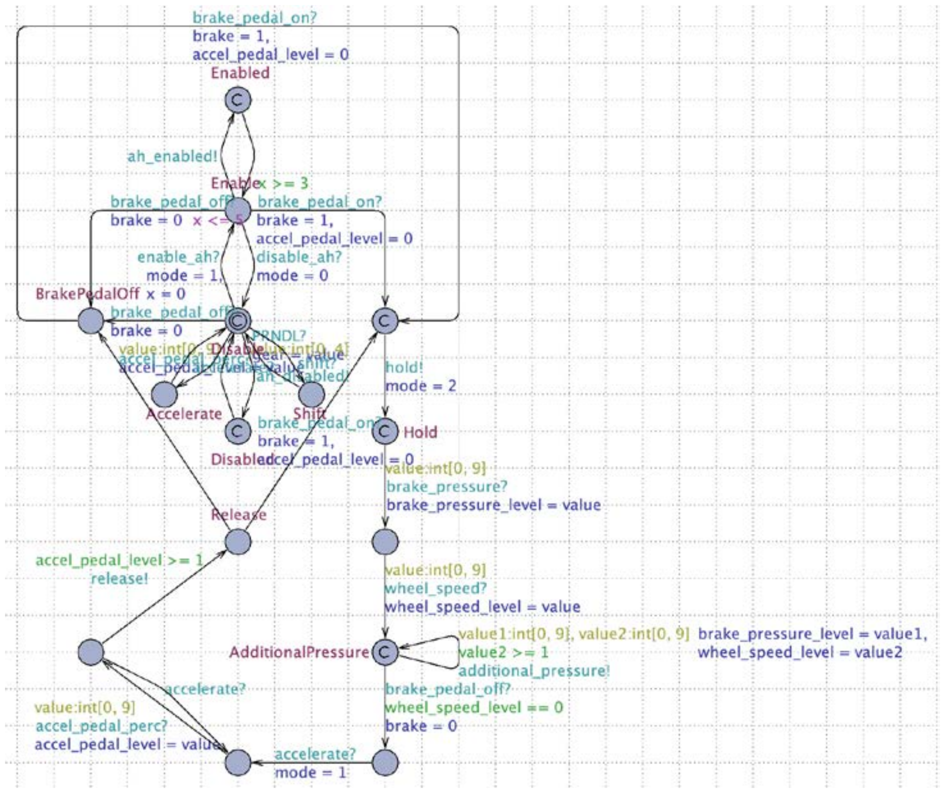
Table 8. Models for Auto-Hold System using UPPAAL

Process model for Auto Hold Module

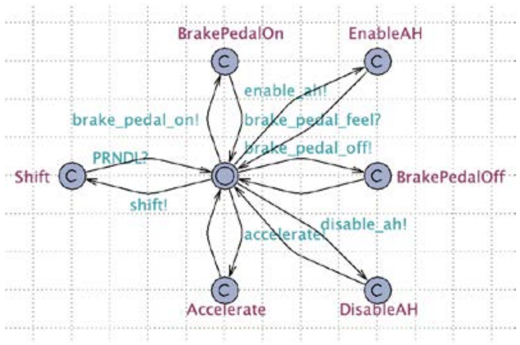
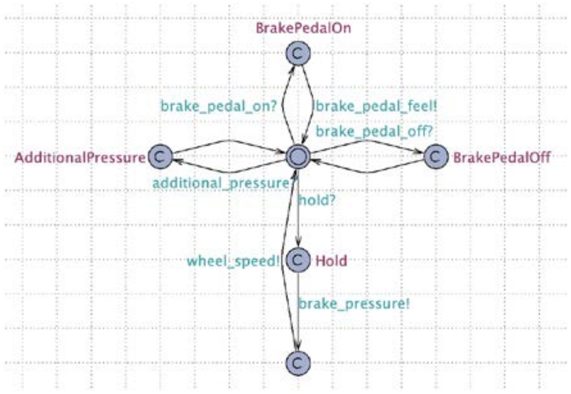
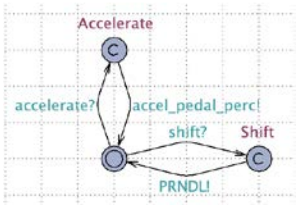
int[0, 2] mode = 0;
int[0, 1] brake = 0;
int[0, 4] gear = 0;
int[0, 9] brake_pressure_level = 0;
int[0, 9] wheel_speed_level = 0;
int[0, 9] accel_pedal_level = 0;
chan enable_ah, disable_ah;
broadcast chan accelerate, brake_pedal_on, brake_pedal_off, shift;

cont. Table 8

chan ah_enabled, ah_disabled, hold, additional_pressure, release;
chan brake_pedal_feel, brake_pressure, wheel_speed;
broadcast chan accel_pedal_perc, PRNDL;	

mode	A variable indicating that the feature is in Disable, Enable, or Hold mode.
brake	A variable indicating whether the brake pedal was pressed.
gear	A variable that represents the current car gear.
brake_pressure_level	A variable that indicates how much the accelerator pedal is pressed.
wheel_speed_level	A variable that indicates the level of brake pressure.
accel_pedal_level	A variable indicating the level of rotation of the wheel.
channels	A variable representing actions that can occur in the entire system
Control Algorithm for Auto Hold Module	
	

cont. Table 8

Model for Driver
 BrakePedalOn EnableAH enable_ah! brake_pedal_on! brake_pedal_feel? brake_pedal_off! PRNDL? Shift BrakePedalOff shift! accelerate! disable_ah! Accelerate DisableAH
Model for Braking System
 BrakePedalOn brake_pedal_on? brake_pedal_feel! brake_pedal_off? AdditionalPressure BrakePedalOff additional_pressure? hold? wheel_speed! Hold brake_pressure!
Model for Propulsion System
 Accelerate accelerate? accel_pedal_perc! shift? Shift PRNDL!

Source: our own study.

Tables [9, 10] list the safety properties and show the verification results of each model checker.

Table 9. Safety Properties and Results using NuSMV

SP1	LTLSPEC (ahm.mode = 1 & ahm.brake = 1 -> ahm.j = 8);	true
SP2	CTLSPEC AG !(ahm.j = 8 & ps.l = 1);	true
SP3	CTLSPEC AG !(ahm.j = 8 & ahm.mode = 0);	true
SP4	CTLSPEC AG !(ahm.j = 8 & ahm.mode = 1 & ahm.brake != 1);	true
SP5	CTLSPEC AG !(ahm.j = 8 & dr.i = 4);	true
SP6	CTLSPEC AG !(ahm.mode = 2 & ahm.accel_pedal_level >= 1);	true
SP7	N/A	N/A
SP8	CTLSPEC AG !(ahm.j = 10 & ahm.mode = 2 & ahm.wheel_speed_level >= 1);	true
SP9	CTLSPEC AG !(ahm.j = 10 & ahm.mode != 2);	true
SP10	N/A	N/A
SP11	LTLSPEC (ahm.j = 14 -> ahm.accel_pedal_level >= 1);	true
SP12	LTLSPEC (ahm.j = 14 -> ahm.mode = 0);	true
SP13	CTLSPEC AG !(ahm.j = 14 & ahm.mode = 2 & ahm.accel_pedal_level = 0);	true
SP14	CTLSPEC AG !(ahm.j = 14 & ahm.accel_pedal_level = 0);	true
SP15	N/A	N/A
SP16	N/A	N/A

Source: our own study.

Table 10. Safety Properties and Results using UPPAAL

SP1	AutoHoldModule.mode == 1 && AutoHoldModule.brake == 1 --> AutoHoldModule.Hold	true
SP2	A[]!(AutoHoldModule.Hold && PropulsionSystem.Accelerate)	true
SP3	A[]!(AutoHoldModule.Hold && AutoHoldModule.mode == 0)	true
SP4	A[]!(AutoHoldModule.Hold && AutoHoldModule.mode == 1 && AutoHoldModule.brake != 1)	true
SP5	A[]!(AutoHoldModule.Hold && Driver.DisableAH)	true

cont. Table 10

SP6	A[]!(AutoHoldModule.mode == 2 && AutoHoldModule.accel_pedal_level >= 1)	true
SP7	A[]!(AutoHoldModule.Hold && x > 5 && x < 3)	N/A
SP8	A[]!(!AutoHoldModule.AdditionalPressure && AutoHoldModule.mode == 2 && AutoHoldModule.wheel_speed_level >= 1)	true
SP9	A[]!(AutoHoldModule.AdditionalPressure && AutoHoldModule.mode !=2)	true
SP10	N/A	N/A
SP11	AutoHoldModule.Release --> AutoHoldModule.accel_pedal_level >= 1	true
SP12	AutoHoldModule.Release --> AutoHoldModule.mode == 0	true
SP13	A[]!(AutoHoldModule.Release && AutoHoldModule.mode == 2 && AutoHoldModule.accel_pedal_level == 0)	true
SP14	A[]!(AutoHoldModule.Release && AutoHoldModule.accel_pedal_level == 0)	true
SP15	N/A	N/A
SP16	N/A	N/A

Source: our own study.

6. DISCUSSIONS

We discuss two case studies and empirical results using two model checkers. There are a total of 4 safety constraint conditions for the Door Interlock System. Among them, 3 safety properties except SP3 could be written with NuSMV, and all 4 could be written with UPPAAL. The reason NuSMV could not write SP3 of the Door Interlock System is because it does not provide a clock concept systematically. For the same property, UPPAAL systemically supports the clock concept and uses TCTL as the property specification language, so the time constraint could be sufficiently expressed. There is one other thing that stands out. Since NuSMV provides almost completely the syntax and semantics of CTL or LTL as a modelling and property specification language, in the case of SP4, it was possible to write safety constraints as safe properties sufficiently using the Until operator. However, in UPPAAL, it was difficult to express the same property because the property specification language is syntactically limited and does not support such things as the Until operator.

In the other case study, the Auto-Hold System, out of a total of 16 safety constraints, 12 safety properties were created with NuSMV, and 13 safety properties could be created with UPPAAL. SP15 and SP16 were excluded because they were not suitable for modelling the control algorithm as an off-nominal situation, and SP10 could not be written because we were unable to access the specification of the system. Since SP7 included time constraints, modeling and formalising were possible only with UPPAAL.

As introduced in the previous section, NuSMV and UPPAAL are commonly used in much of the literature. To the best of our knowledge, however, there have been no attempts to use these two tools at the same time. We obtained the following results by using these two tools to model the same system.

1) The specification language of NuSMV can express the specification abundantly, both syntactically and semantically, so it was very easy to write safety constraints expressed in natural language in logical formula.
2) Since UPPAAL systematically supports clocks, it is very suitable for modelling systems with time constraints.

We used the model checking technique in STPA to discover and derive risk scenarios. As a guide that is generally provided, people directly analyse by constructing a context table with all possible values that the process model can have for control actions. The context table is similar to examining all the states of the system, but by having the model checker take over this task, it can be automatic and less error-prone without human intervention. And if you discover a counterexample through the model checker, you can analyse the trace and help refine the control algorithm of the controller.

In summary, the advantages of using the model checking technique are as follows.

1) Since the model checker verifies whether the safety property is violated through a full investigation, it is possible to reduce the human intervention and reduce the occurrence of mistakes.
2) Even if an actual implemented system does not exist, safety properties can be verified more quickly in the early stages of development or modelling, so a guide can be provided to system engineers to follow safety constraints afterwards.

7. CONCLUSIONS

In this paper, we introduced research on applying model checking, which is one of the formal verification techniques, to STPA. For this, the model checkers chosen were NuSMV and UPPAAL. In two case studies, the control algorithm and process model of the system were modelled using each model checker, and the identified UCAs were written as safety properties. In the early stage of modelling each system, a control algorithm that achieves all safety properties was obtained through the counterexample of the model checker. As a future study, we will conduct research to integrate the model checking technique into the STPA-compliant software development process. So, we will quantitatively analyse how much the model checking technique can help STPA. And it will be necessary to select and apply a larger and more complex system than the system discussed in this paper.

8. ACKNOWLEDGEMENTS

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-00122, Safety Analysis and Verification Tool Technology Development for High Safety Software Development).

REFERENCES

Abdulkhaleq, A., Wagner, S., Leveson, N., 2015, *A Comprehensive Safety Engineering Approach for Software-Intensive Systems Based on STPA*, Procedia Engineering, vol. 128, pp. 2–11.

Baier, C., Katoen, J.-P., Larsen, K.G., 2014, *Principles of Model Checking*, MIT Press, Cambridge.

Dakwat, A.L., Villani, E., 2018, *System Safety Assessment Based on STPA and Model Checking*, Safety Science, vol. 109, pp. 130–143.

De Souza, F.G., de Melo Bezerra, J., Hirata, C.M., de Saqui-Sannes, P., Apvrille, L., 2020, *Combining STPA with SysML Modeling*, 2020 IEEE International Systems Conference (SysCon).

Placke, M.S., 2014a, *Application of STPA to the Integration of Multiple Control Systems: A Case Study and New Approach.*

Tsuji, M., Takai, T., Kakimoto, K., Ishihama, N., Katahira, M., Iida, H., 2020, *Prioritizing Scenarios Based on Stamp/STPA Using Statistical Model Checking*, 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW).

Zhong, D., Sun, R., Gong, H., Wang, T., 2022, *System-Theoretic Process Analysis Based on SysML/Marte and NuSMV*, Applied Sciences, vol. 12(3).

Internet sources

Leveson, N.G., Thomas, J.P., 2018, *STPA Handbook*, http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf (accessed 23.08.2022).

Placke, M.S., 2014b, *Engineering a Safer World*, http://sunnyday.mit.edu/safer-world.pdf (accessed 23.08.2022).