The Journal of Systems & Software 206 (2023) 111840

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Continuous verification with acknowledged MAPE-K pattern and time logic-based slicing: A platooning system of systems case study☆,☆☆,★

Jiyoung Song [a,*], Doo-Hwan Bae [b]

[a] Computer engineering, Hannam University, 70 Hannamro, Daedeok-Gu, Daejeon, 34430, Republic of Korea
[b] School of Computing, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon, 34141, Republic of Korea

ARTICLE INFO

Article history:
Received 19 August 2022
Received in revised form 15 June 2023
Accepted 5 September 2023
Available online 9 September 2023

Keywords:
System of systems
Continuous verification
MAPE-K pattern
PCTL
Model slicing

ABSTRACT

A system of Systems (SoS) has emerged to achieve goals beyond the capabilities of a single system. Platooning is a representative SoS where vehicles are driven in a group for energy efficiency. A leader of a platoon can control followers, but the followers can also leave the platoon independently. During follower leave, energy efficiency and the independent operation of followers may conflict. To resolve the conflicts of the platooning SoS and operate safe platooning maneuvers, continuous verification of platooning is required. Continuous verification is performed repeatedly in a control loop that allows system monitoring and verification. However, there are two problems in the existing approaches: there are no suitable control loop patterns to support the resilient reconfiguration, and the SoS verification cost is high. We propose an approach, called continuous verification of platooning (CVP), that solves these two problems. CVP includes an acknowledged MAPE-K pattern for resilience and a fast and accurate slicing for low verification costs. The acknowledged MAPE-K pattern and slicing algorithm proposed in the paper can be independently used for other systems and models. In the case of the acknowledged MAPE-K pattern, we applied it to a mass casualty incident response SoS, which is another acknowledged type of SoS in the paper, and showed its effectiveness. Our experiments on CVP showed that the pattern reduced the incidence rate of 10 types of failure by 97.3%, and ensured the leave of followers. We also proved the correctness of slicing and demonstrated experimentally that it reduces the verification costs by 68.62.

Editor's note: Open Science material was validated by the Journal of Systems and Software Open Science Board.

1. Introduction

A system of Systems (SoS) can achieve goals that are beyond the capabilities of a single system (Maier, 1998; Nielsen et al., 2015). Platooning is a representative SoS in which vehicles are automatically driven in a group for energy efficiency (Tsugawa et al., 2016). The constituent systems of an SoS have different levels of operational and managerial independence. According to Maier's suggestion (Maier, 1998), if the component systems of SoS are fully independently managed and operated, it is a collaborative type of SoS. According to the new standard proposed by Dahmann and Baldwin (2008), the component system operates independently, but if a manager specified in the SoS exists and can control other systems, it is an acknowledged type. A platooning SoS has the characteristics of both collaborative and acknowledged types. In this paper, we consider platooning SoS as an acknowledged type by focusing on the characteristics of a manager that controls the other systems.

In the platooning SoS, the vehicle at the front of the platoon has the operational command of all vehicles (*i.e.*, leader), and the others follow the command (*i.e.*, followers). However, followers also have a high level of independence to leave for their own destinations and change the leader. This independence of followers can lead to conflicts with goal achievement. For example, common goals of the platooning SoS include safe operation of platoons themselves and increasing fuel efficiency by reducing air resistance. If followers leave a platoon to overtake vehicles and to leave for their individual destinations, they conflict with the leader's common goals. That is, there may be conflicts in which the leader does not allow the followers to diverge in

☆ This research was supported by (No. 2015-0-00250, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System) supervised by the IITP(Institute of Information & Communications Technology Planning & Evaluation).
☆☆ This work was supported by 2023 Hannam University Research Fund.
★ Editor: Heiko Koziolek.
* Corresponding author.
E-mail addresses: jysong@hnu.kr (J. Song), bae@se.kaist.ac.kr (D.-H. Bae).

https://doi.org/10.1016/j.jss.2023.111840
0164-1212/

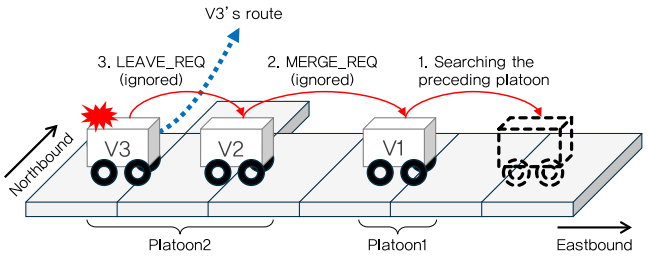


Fig. 1. Example of deadlock failure.

consideration of surrounding traffic conditions and optimum fuel efficiency. The conflicts between the leader and followers must be resolved because if the followers pursue their own independence and leave their platoon without permission, casualties may occur.

Conflict resolution and safe platoon reconfiguration can be ensured through continuous verification. Continuous verification (Chang et al., 1997) constantly models and verifies changing systems and environments and is mainly used under monitor-analyze-plan-execute over a shared knowledge (MAPE-K) (Kephart et al., 2003). In particular, effective decisions can be made based on verification results in the planning phase. In this phase, the verification is performed using model based software engineering. The model based software engineering enables to abstract large system and to simulate the abstracted system in advance. The verification result based on simulation can assure the safety of decision made. Let us assume that we have a platooning SoS and use it in verification. For example, the result of verifying the status of the leader and followers can be used by the followers to determine when to send a leave request, while ensuring the success of the leave and fuel efficiency. However, there are two major problems in applying the existing continuous verification (Ghezzi, 2010; Ghezzi and Sharifloo, 2013; Moreno et al., 2015) to the platooning SoS: (1) irresilient reconfiguration and (2) high verification cost.

First, resilient reconfiguration based on the independent verification of the vehicle is difficult. According to Hyun et al. (2021a), 10 failure types can occur during the operation of the monitoring and execution algorithm of the platooning simulator VENTOS (Amoozadeh et al., 2015). Fig. 1 shows one example among these 10 failure types, in which the leader's merge and follower's leave request cause deadlock. Leader $V2$ of *Platoon*2 constantly sends a merge request to leader $V1$ of *Platoon*1, but the merge request is repeatedly ignored because $V1$ is always searching the preceding platoon when the merge request arrives. In the meantime, the leave requested by follower $V3$ is also constantly ignored owing to the leaders being busy, resulting in a deadlock. One of the causes of this failure is that the follower's independence such as leaving a platoon is not guaranteed. Therefore, a distributed MAPE-K pattern that allows followers to verify their own goals independently, is required rather than a single MAPE-K in the existing continuous verification (Ghezzi, 2010; Ghezzi and Sharifloo, 2013; Moreno et al., 2015).

Second, the verification cost of the platooning SoS is high owing to frequent reconfiguration and model size. Many researchers have proposed approaches to reduce SoS verification costs (Mignogna et al., 2013; Hyun et al., 2019; Song et al., 2022), such as statistical model checking (SMC) (Legay et al., 2010), which can bypass the state explosion problem (Valmari, 1996), and model slicing (Androutsopoulos et al., 2013), which can reduce the size of the verification model itself. SMC can be applied to platooning SoS verification, but the correctness of time-related goal-based slicing is not guaranteed in existing SoS model-slicing approaches (Song et al., 2022). For platooning SoSs, certain requirements require a time specification. For example, when the platoon is driving in a changing environment, to guarantee the success of the leave maneuver, how far ahead of the desired time should the follower request a leave? Such time-related property of system under uncertainty is specified using a bounded until property of probabilistic computational tree logic (PCTL) (Hansson and Jonsson, 1994) To use slicing for verification efficiency, the correctness of bounded until property-based slicing must be demonstrated.

We propose an approach called continuous verification of platooning SoS (CVP) that solves two problems. CVP contains an acknowledged MAPE-K (ACK) pattern and bounded until property-based model slicing. The key idea of the ACK pattern is to give monitoring and verification components to all vehicles to check failure-inducing behaviors and clarify control connections between them. A model-slicing technique is included to solve the second problem. The key to the second problem is to prove that slicing correctness is guaranteed under assumptions and to suggest modeling guidelines that can satisfy the assumptions. This assumption is defined based on whether each constituent system participates in the SoS and whether its behaviors are executable.

We evaluated the effectiveness of CVP through experiments and proofs. To show the effectiveness of the proposed approach, experiments were conducted by applying it to other domains besides platooning SoS. The target domain is a mass casualty incident SoS (MCI SoS) where acknowledged type SoSs are allowed. In the case of MCI SoS, there is central control, but members have the autonomy to independently respond to rapidly changing situations. A detailed description of the MCI scenario and configuration is described in the experimental section. The following sections use a platooning SoS as an example to help understand the approach. The results of the experiment showed that the pattern reduced the incidence rate of 10 types of failure of existing platooning by 97.3% and ensured the leave of followers. We also proved the correctness of slicing and demonstrated it experimentally. In our experiments, slicing reduced verification costs by 68.62%.

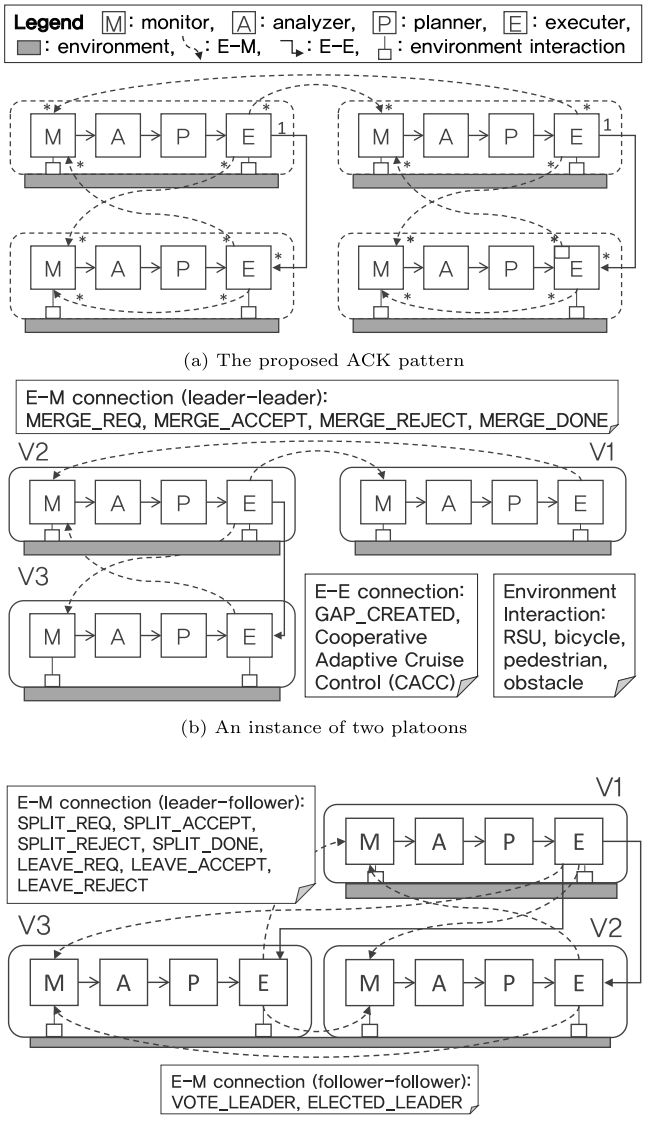
This paper makes the following contributions to the continuous verification of the platooning SoS:

- To solve the irresilient reconfiguration problem, we propose an ACK pattern that enables independent verification based on shared knowledge between vehicles and clarifies control connections (Section 2).
- In the analysis component of the ACK pattern, we propose an analysis approach for signals that cause failure-inducing state transitions and update the platooning SoS models (Section 3).
- To further reduce verification costs, we perform bounded until property-based model slicing along with utilizing SMC and prove its correctness under assumptions (Section 4).
- We evaluate the effectiveness of the ACK pattern and show that it improves the resilient reconfiguration by reducing the occurrence of 10 failure types found in the existing VENTOS by 97.30% (Section 5.1).
- For model slicing, we evaluate correctness and efficiency. The verification results of the original model and slice were shown to be the same using Welch's t-test. Slicing reduced the verification cost by 68.62% over time (Section 5.3).

We conclude this paper with a discussion about related work (Section 6) and present concluding remarks (Section 7).

2. Overview of the ACK pattern

CVP consists of an ACK pattern, model slicing and verification. This section describes how the vehicles of platooning SoS and



(a) The proposed ACK pattern

(b) An instance of two platoons

(c) The result of merging the platoons in Fig. 2b

Fig. 2. The proposed ACK pattern and its instances.

their internal components are arranged and connected when the pattern is applied. A MAPE-K pattern for platooning requires a hierarchy between leaders and followers. Since platoons on the road travel in a single line in one direction, they mainly rely on the monitoring information of the platoon leader to understand the road conditions. In addition, the platoon leader must be able to control in case of a sudden stop. Followers, on the other hand, need independence from the platoon leader to get benefits such as fuel efficiency and plan routes independently. The proposed pattern is designed to satisfy all of the above requirements. For example, a leader can obtain monitoring information from followers, but conversely, does not deliver road condition information to followers. Followers can plan their platoon leave through independent analysis and planning components.

Fig. 2(a) gives an overview of the ACK pattern. The functions of the components appearing in the pattern follow those of the basic MAPE-K framework components (Ghezzi and Sharifloo, 2013; Moreno et al., 2015). The monitoring component (M) collects raw data from environments. The analysis component (A) analyzes and models environmental changes to determine whether a new plan is needed. The planning component (P) verifies modified models and establishes a strategy for the next execution. The execution component (E) executes the new plan within the environments.

As shown in Fig. 2(a), leaders are connected in a horizontal structure, and a leader and followers are connected in a vertical structure. The roles of leader and follower can be changed at any time by reconfiguration. Therefore, both leaders and followers have M, A, P, and E components to execute them according to their operational and managerial independence. The ACK pattern includes connections between vehicles and the environment, between components within the vehicle, and between components in different vehicles.

The connections between vehicles include E-M and E-E connections. An E-M connection can represent the connection of two leaders of different platoons. In this case, merge-related status data (*e.g.*, MERGE_REQ) are transmitted. In the case of an E-M connection within a platoon, data such as the election of a new leader are transmitted. E-E connection represents the connection between leaders and followers within a platoon, and exchanges data for functions that require synchronization, such as vehicle gap while driving (e.g. GAP_CREATED). A detailed description of each connection is given in the following paragraphs and additional data examples are depicted in Fig. 2(a). In addition, there is monitoring of other manual vehicles, obstacles, and pedestrians as interactions with the environment.

Interactions between vehicle and environment. In Fig. 2(b), components M and E of the vehicle interact with the environment. Monitoring the environment is particularly important as leaders should control their followers at a higher level and make decisions regarding safety and fuel efficiency. Component M monitors the communication between the road-side unit and the on-board unit mounted on the vehicle. In addition, component M monitors the environment to prevent accidents with not only fixed obstacles, such as road dents and black ice, but also cyclists and pedestrians that may appear unexpectedly. The execution of vehicles provide feedback to the environment, such as traffic flow changes and road deformation.

Component connections between vehicles. The configuration of the platoon is flexibly changed according to the situation. Thus, the connections between vehicles are represented by dotted lines. The knowledge shared by other vehicles differs depending on the component to which it is connected. Platoon leaders also pursue fuel efficiency, so if they find a leading platoon, they merge and try to become more fuel-efficient followers.

Fig. 2(b) shows an instance of the platoons shown in Fig. 1. $V1$ is the leader of *Platoon*1, and *Platoon*2 consists of $V2$ and $V3$. The connections between leaders are E-M connections. When a merge request is sent from E of $V2$ to M of $V1$, $V1$ transmits merge accept and reject messages from E of $V1$ to M of $V2$ in consideration of the current state and the optimal size of the platoon. The connections between the leader and followers after the merge maneuver are depicted in Fig. 2(c).

The leader and followers can leave the platoon even after it is configured through the merge maneuver. Similar to the merge maneuver, messages such as leave request, leave accept, leave reject, and leave done can be exchanged through E-M connections. When the leader leaves, it broadcasts the leader's leave to the followers. The remaining followers perform an additional leader vote maneuver and agree on whether to elect the leading follower as the leader. All messages related to leader leave and change are transmitted through E-M connections.

During platoon driving, the leader controls the inter-vehicle distance adjustment, speed increase and decrease, route finding, and lane change. The followers are driven according to the controls. The leader synchronizes its own execution with the

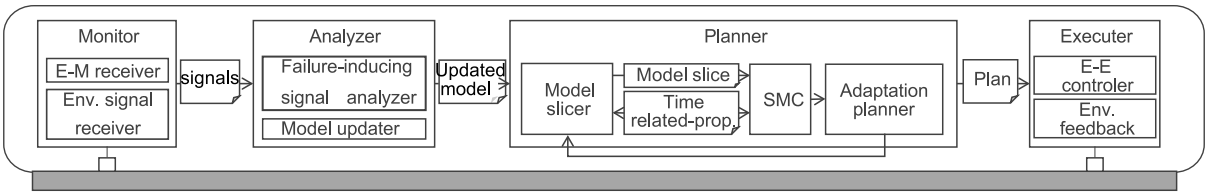


Fig. 3. MAPE components.

Time Step	Veh ID	fromState	toState	Command Sent	R-ID	S-PltId	R-PltId
30.00	V3	sendMerge Req	waitForMerge Req	MERGE_ REQ	V2	P3	P2
30.00	V2	Platoon Leader	sendMerge Accept	–	–	–	–
30.00	V2	sendMerge Accept	wailtFor MergeDone	MERGE_ ACCEPT	V3	P2	P3
30.10	V3	waitFor MergeReply	Merge Accepted	–	–	–	–
30.10	V3	Merge Accepted	waitFor Catchup	–	–	–	–
30.12	V3	waitFor Catchup	sendMerge Done	–	–	–	–
30.12	V3	sendMerge Done	Platoon Follower	MERGE_ DONE	V2	P2	P2
...							
40.00	V1	Platoon Leader	sendMerge Req	–	–	–	–
40.00	V2	sendMerge Req	waitForMerge Req	MERGE_ REQ	V1	P2	P1
40.01	V1	Platoon Leader	sendMerge Req	–	–	–	–
...							
50.00	V3	sendLeave Req	waitForLeave Reply	LEAVE_ REQ	V2	P2	P2
50.00	V1	Platoon Leader	sendMerge Req	–	–	–	–

Fig. 4. Sample signals and state transitions.

follower's execution to prevent accidents during sharp deceleration and acceleration over a narrow inter-vehicle distance. We express the synchronization of component E of the leader and the E components of the followers as E-E connections. Overall, the component connections between vehicles consists of E-M and E-E connections.

Component connections inside the vehicle. When knowledge is shared through interaction with the environment and E-M connections between vehicles, analysis, planning, and execution are carried out inside the vehicle based on it. Fig. 3 shows the roles and composition of components M, A, P, and E. Based on the monitoring knowledge, component A updates the SoS model and analyzes failure-inducing state transitions and maneuvers. The details of the analysis of the platooning SoS are given in Section 3. In component P, the updated model is sliced and verified using SMC. Based on the verification result, the leader decides whether to accept the follower's merge request and followers decide when to send a leave request to the leader. Finally, component E sends a message according to the decision and synchronizes the execution. The details of the slicing performed on component P are presented in Section 4.

3. Analyzing platooning SoS

In this section, we describe what component A does with its signals obtained from the environment and E-M connections. Component A analyzes whether the signals cause failure-inducing state transitions and maneuvers, and updates the SoS changes to the model. The platooning model and protocol we proposed follow the work of Amoozadeh et al. (2015). For example, the basic operations that occur in platooning are merge and split operations, and a leave operation can be performed using the two split operations. However, in Hyun et al.'s study (Hyun et al., 2021a), when platooning was simulated based on the simplified operations of Amoozadeh et al. it was shown that ten driving failures and accidents could occur. We expanded the research of Amoozadeh et al. when implementing the platooning model to be used in the A component to reduce such failures and accidents.

In the three-vehicle platooning SoS presented in Fig. 1, suppose a scenario in which $V3$ requests a merge to $V2$, $V2$ requests a merge to $V1$, and $V3$ requests a leave to $V2$ Fig. 4 depicts the signals and state transitions that occur during the scenario execution. The figure shows the time, vehicle id, previous state, current status, sent signal, receiver id, sender platoon id, and receiver platoon id from the leftmost column. After $V3$ sends *MERGE_REQ* at the 30.00 time step, the merge is completed at the 30.12 time step and $V3$ belongs to *Platoon2*. $V2$ then sends merge requests to $V1$ from the 40.00 to 50.00 time steps as well. However, they are ignored because $V1$ keeps transitioning from *platoonLeader* to *sendMergeReq* to search the preceding platoon and $V1$ can accept the merge requests only in *platoonLeader*. It is reasonable to transition states to find a platoon, but this can lead to compounding-error problems (Da Silva and Dobránszki, 2017), such as $V3$'s failure to leave *Platoon2*. Therefore, in the absence of a preceding platoon and in the rejection of signals, the merge request interval needs to be extended to reduce failure-inducing state transitions so that other maneuvers can be executed.

We modeled the platooning SoS to analyze the above state transitions and maneuvers. As shown in Fig. 5, the probabilistic platooning SoS model is built based on three maneuvers: merging, splitting, and leaving (Amoozadeh et al., 2015). We found one cause of the failures described in the study by Hyun et al. (2021b) through an analysis of the existing state machines (Amoozadeh et al., 2015). The cause is that an infinite loop that continuously requests merge to the preceding platoon occurs. The infinite loop from *platoonLeader* to *waitForMergeReply* in merge states starts itself immediately even though the merge request is rejected and ignored. That is, in the existing state machines, the model is designed by prioritizing the operational independence of the platoon leaders over that of the followers.

The model we propose has the following two characteristics to ensure the operational independence of the followers and to enable realistic scenario verification.

First, before entering busy states from *state_platoon Leader*, we check whether a preceding platoon exists in *waitForBeacon* and have received *REJECT*, so that we can prioritize the operational independence of the followers. If the vehicle checks those signals, it is set to either send a signal at a longer interval or not.

Second, all maneuvers are integrated in the probabilistic model and the execution probabilities are assigned to each maneuver. All vehicles can become leaders, followers, or non-participants

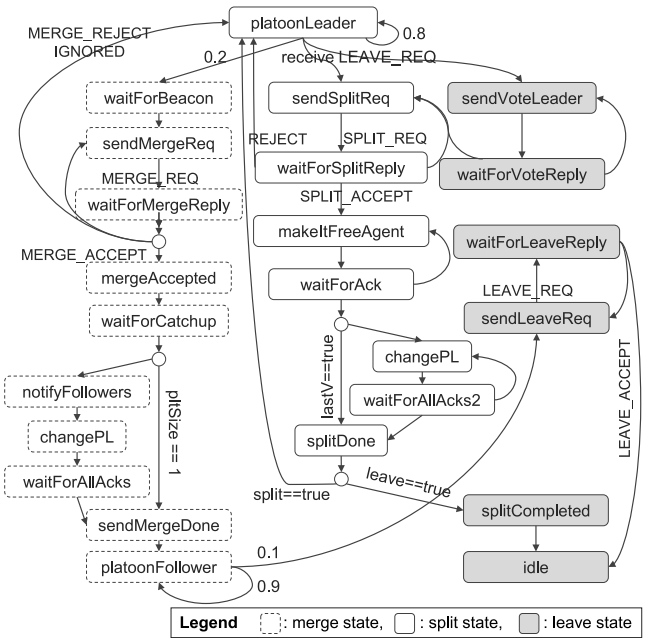


Fig. 5. An abstracted probabilistic platooning model.

(*i.e.*, idle) at any time, and maneuvers are intervened and executed. We assign the probability that maneuvers will be used in *platoonLeader*, *platoonFoll ower*, and *idle*. For example, in Fig. 5, if *platoonLeader* does not receive any request, there will be two transitions with 0.8 and 0.2 probabilities from *platoon Leader* to itself and $waitForBeacon$, respectively. By learning the monitoring signals obtained while platooning is executed, this probability can be closer to the realistic scenario.

Even with the proposed probabilistic model, we cannot prevent the leader from transitioning to the busy state; therefore, we perform verification to ensure the independence of the followers. For example, we verify that the success of the leave can be guaranteed when the follower makes a leave request to the leader several minutes or meters before leaving for another destination. In Section 4, we introduce an approach for the model-based, efficient, and accurate verification performed in component P.

4. Slicing and verifying platooning SoS

In our proposed component P, slicing and SMC are used for verification efficiency. We model the platooning SoS as a discrete-time Markov chain (DTMC) because state changes and reconfiguration of platooning SoS shown in the previous section occur discretely and probabilistically. PCTL (Hansson and Jonsson, 1994) is mainly used to specify time-related properties for probabilistic models. However, the correctness of the slicing algorithm for the bounded until property of PCTL has not been proven (Song et al., 2022). In this section, we provide a background on the syntax and semantics of PRISM and PCTL, and the PRISM DTMC slicing algorithm. We prove the correctness of the slicing algorithm under an assumption. We present modeling guidelines for the PRISM DTMC SoS to ensure that this assumption is satisfied.

4.1. Background of PRISM slicing

PRISM syntax and semantics. The syntax of PRISM is formally described in Fig. 6. A model consists of modules and their commands. Variables have Boolean, integer, and double. Expressions include variables, constants, binary and unary operations. Commands are formatted according to the conditions and actions. The

$$
\begin{aligned}
T &::= Bool \mid Integer \mid Double \\
uop &::= - \mid \neg \\
bop &::= \times \mid \div \mid + \mid - \mid < \mid \leq \mid \geq \mid > \mid \\
&\quad = \mid \neq \mid \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \\
\varepsilon &::= const \mid v \mid uop\ \varepsilon \mid \varepsilon\ bop\ \varepsilon \\
Init \ni init &::= v : T \texttt{ = } const \\
L \ni l &::= String \\
D \ni cond &::= \varepsilon \\
Atom \ni atom &::= \texttt{skip} \mid v' = \varepsilon \\
A \ni act &::= atom \mid act_1\ \&\ \cdots\ \&\ act_i \mid \\
&\quad p_1 : act_1 + \cdots + p_j : act_j \\
C \ni c &::= \texttt{[]}\ cond\ \texttt{->}\ act \mid \texttt{[}l\texttt{]}\ cond\ \texttt{->}\ act \\
Module \ni \texttt{module} &\triangleq \langle \texttt{list init, list } C \rangle \\
\mathcal{M} \ni M &\triangleq \texttt{list } Module
\end{aligned}
$$

Fig. 6. PRISM syntax.

semantics of PRISM is described in Fig. 7. Transitions from a given state s in a PRISM model are enumerated by the function $next : \mathcal{M} \times S \rightarrow 2^{\mathbb{A}}$. There are two types of action labels in $next(M, s)$: local ($\mathtt{a}_{i,j}$) and synchronized ($\mathtt{sa}_l$). A local action label is executed when its condition is satisfied. Synchronized action labels located in different modules are executed when all of their conditions are met. The probabilities of action labels are normalized by the cardinality of $next(M, s)$. Fig. 8 is an example PRISM DTMC model that abstracts Fig. 5.

PCTL syntax and semantics. The syntax and semantics of the PCTL are formally described in Fig. 9. There are two types of formulas in PCTL: a state formula ϕ, and a path formula ψ. ϕ consists of atomic propositions, logical connectives and operators. Two state formulas can build the path formula with an $U^{<t}$ operator. The semantics of ϕ for a state s are defined by satisfaction relations $[\![\phi]\!]_M(s)$. The satisfaction relation for ϕ represents whether the given state s satisfies the state formula or not. The semantics of ψ for a state s are defined by satisfaction relations $[\![\psi]\!]_M(s)$. ψ with $U^{<t}$, checks states along a specific path. $Path(M, s_0, t)$ is the set of paths derived by t transitions in M starting from s_0. $\chi(\phi_1, \phi_2, t, s_0)$ is the set of paths in $Path(M, s_0, t)$ satisfying $\phi_1 U \phi_2$.

$$
\begin{aligned}
\chi(\phi_1, \phi_2, t, s_0) \triangleq \{ & s_0 \cdots s_t \in Path(M, s_0, t) \mid \\
& s_t \models \phi_2 \ \wedge \forall i \in 0 \leq i < t, \\
& s_i \models \phi_1 \wedge s_i \not\models \phi_2 \} \ .
\end{aligned}
$$

$\phi_1 U^{<t} \phi_2$ means that states satisfy ϕ_1 until reaching a state that satisfies ϕ_2 within t transitions. For example, $true\ U^{\leq 20}\ V2\_state = 27$ means that the follower $V2$ becomes *idle* within 20 time step. The probability measure for $[\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(s)$ is calculated as follows.

$$
[\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(s_0) = \sum_{s_0 \ldots s_j \in \chi(t, s_0)} \prod_{i=0}^{t-1} [\![M]\!](s_i, s_{i+1}) \ .
$$

PRISM slicing algorithm. PRISM DTMC model slicing involves drawing the dependence graph $\mathbf{D}(M) = (V, E)$, calculating the set, $\mathbf{I}$ of influencer variables, and transforming the model. V of the dependency graph consists of condition-labels $\mathtt{c}_{i,j}$ and atomic

• Action semantics $[\![act]\!] : S \times S \to [0,1]$

$$
\begin{aligned}
[\![\texttt{skip}]\!](s,s') &= \begin{cases} 1 & (if\ s' = s) \\ 0 & (otherwise) \end{cases} \\
[\![v' = \varepsilon]\!](s,s') &= \begin{cases} 1 & (if\ s' = s[v' \mapsto [\![\varepsilon]\!](s)]) \\ 0 & (otherwise) \end{cases} \\
[\![act_1\ \&\cdots\&\ act_i]\!](s,s') &= \sum_{t \in S} [\![act_1]\!](s,t) \cdot [\![act_2 \\
&\quad \&\cdots\&\ act_i]\!](t,s') \\
[\![p_1{:}act_1 + \cdots + p_j{:}act_j]\!](s,s') &= \sum_{k=1}^{j} p_k \cdot [\![act_k]\!](s,s')
\end{aligned}
$$

• Model semantics $[\![M]\!] : S \times S \to [0,1]$

– A set of actions, $next : \mathcal{M} \times S \to 2^{\mathtt{A}}$, $\quad I : \mathbb{N} \to \texttt{option}\ \mathbb{N}$

$$
\begin{aligned}
next(M,\ s) = \{&\mathtt{a}_{i,j} \mid \exists i \in \{1,\cdots,|M|\}, \\
&\exists j \in \{1,\cdots,|M[i].C|\}, \\
&\exists cond \in D, \quad \exists act \in A, \\
&M[i].C[j] = \texttt{[]}\ cond\texttt{->}act\ \wedge\ [\![cond]\!](s)\} \\
\cup \{&\mathtt{sa}_I \mid \exists l \in L, \\
&(\forall i \in \{1,\cdots,|M|\}, \forall j \in \{1,\cdots,|M[i].C|\}, \\
&I(i) = \texttt{Some}\ j \to \exists cond \in D, \quad \exists act \in A, \\
&M[i].C[j] = \texttt{[}l\texttt{]}\ cond\texttt{->}act\ \wedge\ [\![cond]\!](s)) \\
\wedge\ &(\forall i \in \{1,\cdots,|M|\},\ \exists j \in \{1,\cdots,|M[i].C|\}, \\
&\exists cond \in D, \quad \exists act \in A, \\
&M[i].C[j] = \texttt{[}l\texttt{]}cond\texttt{->}act \to\ I(i) \neq \texttt{None}) \\
\wedge\ &(I \neq \lambda\_.\ \texttt{None})\}
\end{aligned}
$$

– An action, $lookup : \mathcal{M} \times \mathtt{A} \to A$, $\quad lookup : \mathcal{M} \times \mathtt{SA} \to A$

$$
\begin{aligned}
lookup(M, \mathtt{a}_{i,j}) = \ & act\ s.t.\ \exists i,j \in \mathbb{N}, \quad \exists cond \in D, \\
& M[i].C[j] = \texttt{[]}\ cond\texttt{->}act \\
lookup(M, \mathtt{sa}_I) = \ & act_{i_1}\ \&\cdots\&\ act_{i_{|dom(I)|}}\ s.t.\ \exists l \in L, \\
& \forall i \in dom(I), \\
& \exists cond \in D, M[i].C[I[i]] = \texttt{[}l\texttt{]}\ cond\texttt{->}act_i
\end{aligned}
$$

$$
[\![M]\!](s,s') = \frac{\sum_{\mathtt{a} \in next(M,s)} [\![lookup(M,\mathtt{a})]\!](s,s')}{|next(M,s)|}
$$

Fig. 7. PRISM semantics.

action-labels $\mathtt{a}_{i,j,k}$:

$$
\begin{aligned}
V = \ & \{\mathtt{c}_{i,j} \mid \exists i \in \{1,\ldots,|M|\},\ \exists j \in \{1,\ldots,|M[i].C|\}\} \\
& \cup \{\mathtt{a}_{i,j,k} \mid \exists i \in \{1,\ldots,|M|\},\ \exists j \in \{1,\ldots,|M[i].C|\}, \\
& \quad \exists k \in \mathbb{N},\ \exists atom \in Atom,\ f(M[i].C[j],k) = atom\}\ .
\end{aligned}
$$

$f(C,n)$ returns the nth atomic action in the given C. $\mathtt{c}_{i,j}$ is a condition label mapped to *cond* of $M[i].C[j]$ and $\mathtt{a}_{i,j,k}$ is an atomic action label mapped to the kth atomic action in $M[i].C[j]$, respectively.

E consists of control, data, and synchronization dependence as follows:

$$
\begin{aligned}
E = \ & \{(v_1,\ v_2) \mid \exists i,j,k \in \mathbb{N}, \quad v_1 = \mathtt{c}_{i,j}\ \wedge\ v_2 = \mathtt{a}_{i,j,k}\} \quad \text{(CONTROL)} \\
& \cup \{(v_1,\ v_2) \mid Write(v_1) \cap Read(v_2) \neq \emptyset\ \} \quad \text{(DATA)} \\
& \cup \{(v_1,\ v_2) \mid \exists i_1,i_2,j_1,j_2 \in \mathbb{N}, \quad \exists l \in L,
\end{aligned}
$$

```
1  dtmc
2
3  module  PlatoonLeader
4     isPV1: bool init true;
5     leader_state: [0..27] init 0;
6     // 0: platoonLeader, 1:waitForBeacon,
          18: sendSplitReq
7     [] isP & leader_state=0 -> 0.8:
          (leader_state'=0) +0.2:
          (leader_state'=1);
8     [f1] isP & leader_state=0 ->
          (leader_state'=18);
9     [f2] isP & leader_state=0 ->
          (leader_state'=18);
10    ...
11 endmodule
12
13 module PlatoonFollower1
14    follower1_state: [0..27] init 26;
15    // 26: platoon follower, 27: idle
16    isLeave1: bool init true;
17
18    [f1] isLeave1 = true ->
          (follower1_state'=27);
19    ...
20 endmodule
21
22 module PlatoonFollower2
23    follower2_state: [0..27] init 10;
24    isLeave2: bool init false;
25
26    [f2] isLeave2 = true ->
          (follower2_state'=27);
27    ...
28 endmodule
```

Fig. 8. An abstracted platooning PRISM DTMC.

$$
\begin{aligned}
& \exists cond_1, cond_2 \in D, \quad \exists act_1, act_2 \in A, \\
& v_1 = \mathtt{c}_{i_1,j_1} \wedge\ v_2 = \mathtt{c}_{i_2,j_2} \wedge\ i_1 \neq i_2 \\
& \wedge\ M[i_1].C[j_1] = [l]cond_1 - > act_1 \\
& \wedge\ M[i_2].C[j_2] = [l]cond_2 - > act_2\ \}. \qquad \text{(SYNC)}
\end{aligned}
$$

Read: $V \to 2^{\mathbf{V}}$ and *Write*: $V \to 2^{\mathbf{V}}$ functions map a vertex v to the variables to be read and written, respectively.

Given $\mathbf{D}(M)$ and a path formula ψ, we can calculate the set, $\mathbf{I}$, of influencers. *Influencers* are variables that influence R which is the set of variables in ψ. $Inf : \mathbf{D} \times 2^{\mathbf{V}} \to 2^{\mathbf{V}}$ function maps the dependency graph $D = (V,E)$ and R to influencers as follows:

$$
\begin{aligned}
Inf(D,R) = \ & R \cup \{v \mid \exists u, v \in D.V, \\
& \quad v \in Read(v) \cup Write(v)\ \wedge \\
& \quad v \xrightarrow[D.E]{*} u\ \wedge\ R \cap Write(u) \neq \emptyset\ \}\ .
\end{aligned}
$$

The influencer calculation is recursively performed backwards.

Given the calculated $\mathbf{I}$ and M to the slicing function *SLI*, M is transformed into a model slice according to the definitions in Fig. 10. For example, if we slice Fig. 8 based on *true* $U^{\leq 20}$ *V2_state* $= 27$, $\mathbf{I}$ will contain *V1_state*, *isPV1*, *V2_state*, and *isPV2* and the model slice will contain the whole lines of modules *PlatoonLeader* and *PlatoonFollower1*, and a condition of the 26th line of *PlatoonFollower2*.

- PCTL's Syntax

$$\phi \ ::= \ \mathtt{tt} \mid ap \mid \neg\phi \mid \phi_1 \ \wedge \ \phi_2 \qquad \text{(STATE)}$$
$$\psi \ ::= \ \phi_1 \ U^{\leq t}_{>p} \ \phi_2 \mid \phi_1 \ U^{\leq t}_{\geq p} \ \phi_2 \qquad \text{(PATH)}$$

- PCTL's Denotational Semantics for State Formula

$$\begin{aligned} [\![\mathtt{tt}]\!]_M(s) &\triangleq \mathtt{skip} \\ [\![ap]\!]_M(s) &\triangleq ap \in L(s) \\ [\![\neg\phi]\!]_M(s) &\triangleq not\ [\![\phi]\!]_M(s) \\ [\![\phi_1 \wedge \phi_2]\!]_M(s) &\triangleq [\![\phi_1]\!]_M(s)\ and\ [\![\phi_2]\!]_M(s) \end{aligned}$$

- PCTL's Denotational Semantics for Path Formula

$$\begin{aligned} [\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(\pi) &\triangleq \exists i \leq t\ s.t.\ [\![\phi_2]\!]_M(s_i) \wedge \\ &\quad \forall j : 0 \leq j < i,\ [\![\phi_1]\!]_M(s_j) \\ [\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(s) &\triangleq \mu^M_s(\pi \in Paths(M,s) \text{---} \pi_0 = s \wedge \\ &\quad [\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(\pi)) \end{aligned}$$

Fig. 9. PCTL's syntax and denotational semantics.

- Transformation of variable initialization lists $SLI : Init \times 2^{\mathbf{V}} \rightarrow Init$

$$SLI(v:\ T\ init\ const,\ \mathbf{I}) \ = \ \begin{cases} v:\ T\ init\ const & (\text{if} \quad v \in \mathbf{I}) \\ \bot & (\text{otherwise}) \end{cases}$$

- Transformation of commands $SLI : C \times 2^{\mathbf{V}} \rightarrow \text{Option } C$

$$\begin{aligned} SLI(\mathtt{skip},\ \mathbf{I}) &= \mathtt{skip} \\ SLI(v' = \varepsilon,\ \mathbf{I}) &= \begin{cases} v' = \varepsilon & (\text{if} \quad v \in \mathbf{I}) \\ \mathtt{skip} & (\text{otherwise}) \end{cases} \\ SLI(act_1\ \&\cdots\&\ act_i,\ \mathbf{I}) &= SLI(act_1,\ \mathbf{I})\ \&\cdots\& \\ &\qquad SLI(act_i,\ \mathbf{I}) \\ SLI(p_1 : act_1 + \cdots + p_j : act_j) &= p_1 : SLI(act_1,\ \mathbf{I}) + \cdots + p_j : SLI(act_j,\ \mathbf{I}) \\ SLI(\mathtt{[]}\ cond\text{->}act,\ \mathbf{I}) &= \begin{cases} \mathtt{[]}\ cond\text{->}SLI\ (act,\ \mathbf{I}) \\ \qquad (\text{if} \quad Var(cond) \subseteq \mathbf{I}) \\ \bot \quad (\text{otherwise}) \end{cases} \\ SLI(\mathtt{[}l\mathtt{]}\ cond\text{->}act,\ \mathbf{I}) &= \begin{cases} \mathtt{[}l\mathtt{]}\ cond\text{->}SLI\ (act,\ \mathbf{I}) \\ \qquad (\text{if} \quad Var(cond) \subseteq \mathbf{I}) \\ \bot \quad (\text{otherwise}) \end{cases} \end{aligned}$$

Fig. 10. *SLI* transformation for commands in *M*.

4.2. Correctness of bounded until-based slicing

Proof of slicing correctness. We here prove the correctness of the bounded until property-based PRISM model slicing under an assumption. We define preliminaries as follows: Let S be the set of states of M, S' be the set of states of $SLI(M, \mathbf{I})$, and $s'_0 = s_0|_{\mathbf{I}}$ be the initial state of $SLI(M, \mathbf{I})$. $NL(M)$ is the set of all action-labels of M.

$$\mathtt{n} \in NL(M) \triangleq \bigcup_{s \in S} next(M, s)\ .$$

$SL(M)$ is the action label set assigning values to variables in $\mathbf{I}$.

$$\mathtt{ns} \in SL(M) \triangleq \{n \in NL(M) \mid Write(lookup(M, n)) \ \cap \ \mathbf{I} \ \neq \ \emptyset\}\ .$$

$AL(M)$ is the set excluding $SL(M)$ from $NL(M)$.

$$\mathtt{na} \in AL(M) \triangleq NL(M) \backslash SL(M)\ .$$

We prove Theorem 1 under an assumption that $\forall 0 \leq k \leq l \leq t, \forall s_0 \cdots s_l \in \chi(\phi_1, \phi_2, l, s_0), next(M, s_k) \cap AL(M) \neq \emptyset$. The assumption means that transitions that are not included in the model slice should not be available from the states in $Path(M, s_0, t)$ satisfying $\phi_1 U \phi_2$.

Theorem 1. *Let M be a PRISM model, $\phi_1\ U^{\leq t}\ \phi_2$ be a bounded until property, D be the dependency graph of M, R be the set of variables in $\phi_1\ U\ \phi_2$, $\mathbf{I} = Inf(D, R)$, s_0 be a state of M, and $s'_0 = s_0|_{\mathbf{I}}$ be the corresponding state in the slice, and $\forall 0 \leq k \leq l \leq t$, $\forall s_0 \cdots s_l \in \chi(\phi_1, \phi_2, l, s_0)$, $next(M, s_k) \cap AL(M) \neq \emptyset$. We have $[\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(s_0) = [\![\phi_1\ U^{\leq t}\ \phi_2]\!]_{SLI(M,\mathbf{I})}(s'_0)$.*

Proof.

$$[\![\phi_1 U^{\leq t} \phi_2]\!]_M(s_0)$$
$$= \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \\ \displaystyle\sum_{k=1}^{t} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \cdots \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s \models \phi_2\}}} \\ \displaystyle\prod_{i=0}^{k-1} \left(\frac{\sum_{\mathtt{ns} \in next(M,s_i) \cap SL(M)} [\![lookup(M, \mathtt{ns})]\!](s_i, s_{i+1})}{|next(M, s_i) \cap SL(M)|} \right) \\ (otherwise) \end{cases}$$

(by Lemma 2)

$$= \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \\ \displaystyle\sum_{k=1}^{t} \sum_{\substack{s'_1 \in \\ \{s' \in S' \mid s' \models \phi_1 \\ \wedge s' \not\models \phi_2\}}} \sum_{\substack{s_1 \in \\ \{s \in S \mid s'_1 = s|_{\mathbf{I}}\}}} \cdots \sum_{\substack{s'_{k-1} \in \\ \{s' \in S' \mid s' \models \phi_1 \\ \wedge s' \not\models \phi_2\}}} \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s'_{k-1} = s|_{\mathbf{I}}\}}} \\ \displaystyle\sum_{\substack{s'_k \in \\ \{s' \in S' \mid s' \models \phi_2\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s'_k = s|_{\mathbf{I}}\}}} \left(\prod_{i=0}^{k-1} \sum_{\substack{\mathtt{ns} \in \\ next(M,s_i) \\ \cap SL(M)}} \frac{[\![lookup(M, \mathtt{ns})]\!](s_i, s_{i+1})}{|next(M, s_i) \cap SL(M)|} \right) \\ (otherwise) \end{cases}$$

$$\left(\because \bigcup_{\substack{q' \in \{s' \in S' \mid \\ s' \models \phi_1 \wedge s' \not\models \phi_2\}}} \{s \in S | q' = s|_{\mathbf{I}}\} = \{s \in S | s \models \phi_1 \wedge s \not\models \phi_2\} \wedge \bigcup_{q' \in \{s' \in S' | s' \models \phi_2\}} \{s \in S | q' = s|_{\mathbf{I}}\} = \{s \in S | s \models \phi_2\} \right)$$

$$= \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \sum_{k=1}^{t} \sum_{\substack{s'_1 \in \\ \{s' \in S' \mid s' \models \phi_1 \wedge s' \not\models \phi_2\}}} \cdots \sum_{\substack{s'_{k-1} \in \\ \{s' \in S' \mid s' \models \phi_1 \wedge s' \not\models \phi_2\}}} \sum_{\substack{s'_k \in \\ \{s' \in S' \mid s' \models \phi_2\}}} \\ \left(\sum_{\substack{s_1 \in \\ \{s \in S \mid s'_1 = s|_{\mathbf{I}}\}}} \cdots \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s'_{k-1} = s|_{\mathbf{I}}\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s'_k = s|_{\mathbf{I}}\}}} \right. \\ \left. \prod_{i=0}^{k-1} \sum_{\substack{\mathrm{ns} \in \\ next(M,s_i) \\ \cap SL(M)}} \frac{[\![lookup(M, \mathtt{ns})]\!](s_i, s_{i+1})}{|next(M, s_i) \cap SL(M)|} \right) & (otherwise) \end{cases}$$

(by the distributive property)

$$= \begin{cases} 1 & (s'_0 \models \phi_2) \\ 0 & (s'_0 \not\models \phi_1 \wedge s'_0 \not\models \phi_2) \\ \sum_{t=1}^{\infty} \sum_{\substack{s'_1 \in \\ \{s' \in S' \mid s' \models \phi_1 \wedge s' \not\models \phi_2\}}} \cdots \sum_{\substack{s'_{t-1} \in \\ \{s' \in S' \mid s' \models \phi_1 \wedge s' \not\models \phi_2\}}} \sum_{\substack{s'_t \in \\ \{s' \in S' \mid s' \models \phi_2\}}} \\ \left(\prod_{i=0}^{t-1} \sum_{\substack{\mathrm{ns} \in \\ next(SLI(M,\mathbf{I}),s'_i) \\ \cap SL(SLI(M,\mathbf{I}))}} \frac{[\![lookup(SLI(M,\mathbf{I}), \mathtt{ns})]\!](s'_i, s'_{i+1})}{|next(SLI(M,\mathbf{I}), s'_i) \cap SL(SLI(M,\mathbf{I}))|} \right) \\ (otherwise) \end{cases}$$

($\because s_0|_{\mathbf{I}} = s'_0 \ \wedge$ Lemma 5 of Song et al., (2022))

$= [\![\phi_1 U \phi_2]\!]_{SLI(M,\mathbf{I})}(s'_0)$ (by Lemma 2) □

Lemma 2. $\forall s_0 \in S$,

$$[\![\phi_1 U^t \phi_2]\!]_M(s_0) = \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \sum_{k=1}^{t} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \cdots \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s \models \phi_2\}}} \\ \prod_{i=0}^{k-1} \left(\frac{\sum_{\mathrm{ns} \in next(M,s_i) \cap SL(M)} [\![lookup(M, \mathtt{ns})]\!](s_i, s_{i+1})}{|next(M, s_i) \cap SL(M)|} \right) \\ (otherwise) \end{cases}$$

Proof.

$$[\![\phi_1 U^{\leq t} \phi_2]\!]_M(s_0) = \sum_{k=0}^{t} \sum_{\substack{s_0 \cdots s_k \in \\ \chi(\phi_1,\phi_2,k,s_0)}} \prod_{i=0}^{k-1} [\![M]\!](s_i, s_{i+1})$$

(by definition of bounded until property)

$$= \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \sum_{k=1}^{t} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \cdots \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s \models \phi_2\}}} \\ \prod_{i=0}^{k-1} [\![M]\!](s_i, s_{i+1}) & (otherwise) \end{cases}$$

(by definition of χ)

$$= \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \sum_{k=1}^{t} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \cdots \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s \models \phi_2\}}} \\ \prod_{i=0}^{k-1} \frac{\sum_{\mathrm{n} \in next(M,s_i)} [\![lookup(M, \mathtt{n})]\!](s_i, s_{i+1})}{|next(M, s_i)|} & (otherwise) \end{cases}$$

(by definition of M)

$$= \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \sum_{k=1}^{t} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \cdots \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s \models \phi_2\}}} \\ \prod_{i=0}^{k-1} \left(\frac{\sum_{\mathrm{na} \in next(M,s_i) \cap AL(M)} [\![lookup(M, \mathtt{na})]\!](s_i, s_{i+1})}{|next(M, s_i) \cap AL(M)| + |next(M, s_i) \cap SL(M)|} \right. \\ \left. + \frac{\sum_{\mathrm{ns} \in next(M,s_i) \cap SL(M)} [\![lookup(M, \mathtt{ns})]\!](s_i, s_{i+1})}{|next(M, s_i) \cap AL(M)| + |next(M, s_i) \cap SL(M)|} \right) \\ (otherwise) \end{cases}$$

($\because SL(M) \cap AL(M) = \emptyset$)

$$= \begin{cases} 1 & (s_0 \models \phi_2) \\ 0 & (s_0 \not\models \phi_1 \wedge s_0 \not\models \phi_2) \\ \sum_{k=1}^{t} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \cdots \sum_{\substack{s_{k-1} \in \\ \{s \in S \mid s \models \phi_1 \wedge s \not\models \phi_2\}}} \sum_{\substack{s_k \in \\ \{s \in S \mid s \models \phi_2\}}} \\ \prod_{i=0}^{k-1} \left(\frac{\sum_{\mathrm{ns} \in next(M,s_i) \cap SL(M)} [\![lookup(M, \mathtt{ns})]\!](s_i, s_{i+1})}{|next(M, s_i) \cap SL(M)|} \right) \\ (otherwise) \end{cases}$$

(by the assumption, $\forall 0 \leq k \leq l \leq t$, $\forall s_0 \cdots s_l \in \chi(\phi_1, \phi_2, l, s_0)$, $next(M, s_k) \cap AL(M) \neq \emptyset$.) □

Modeling guidelines and slicability. To perform the model slicing based on the bounded until property, transitions that are not included in the model slice should be unavailable from the states in $\chi(\phi_1, \phi_2, t, s_0)$, as the assumption implies. We present modeling guidelines that make this assumption possible and how to check slicability.

The modeling guidelines are as follows: Vehicles that do not participate and are not viable for achieving the platooning SoS goal should be isolated from the SoS model. To clearly distinguish the participation and execution of vehicles, one constituent system should be modeled as one or more modules independent of other systems when modeling SoSs with PRISM. In addition, each module should include a variable that can indicate whether to participate, that is, a participation index. Taking Fig. 8 as an example, each leader and follower is modeled as one module, and each module has a variable called *isSomething* as a participation index. Commands in a module with a participation index of *false* should be modeled to have unsatisfiable *cond*s in order to conform the assumption.

Communication channels with independent systems such as E-M connections are modeled with variables in multiple modules and independent action-labels. As shown in the 8th and 18th lines of Fig. 8, the E-M connection is modeled with the [*f*1]

Table 1
Failure types and their descriptions (Hyun et al., 2021b).

CASE1	CASE2	CASE3	CASE4	CACSE5
Merge & merge: Constantly request Merge to the original leader	Split & merge: Constantly request Merge to the original leader	LLeave & merge: Constantly request Merge to the original leader	FLeave & merge: Constantly request Merge to the original leader	Split Optsize: Constantly request Merge to the new platoon leader
CASE6	**CASE7**	**CASE8**	**CASE9**	**CASE10**
LLeave Optsize: Constantly request Merge to the new platoon leader	LLeave Optsize: Constantly request Merge to the left platoon leader	MFLeave Optsize: Constantly request Merge to the left platoon leader	MFLeave Optsize: Constantly request Merge to the new platoon leader	EFLeave Optsize: Constantly request Merge to the left platoon leader

action label and the participant index variables for the leader and followers. The E-E connection is a control that is synchronized from the leader, so it should be modeled with the same action label. In modeling communication channels, the conditions of the local and synchronized action labels should not be *true* because unconditional action execution can affect the cardinality of the *next* function.

Model slicability based on bounded until can be confirmed by checking whether all participation indices in the modules that are not included in the model slice have *false* values. If there are modules and transitions that can participate in addition to the slice, the value of the *next* function changes during model probability calculation, so the probability verification result of the original model and that of the model slice may be different.

5. Evaluation

We experimentally evaluated the effectiveness of the ACK pattern on two domains platooning SoS and mass casualty incident response (MCIR) SoS (Section 5.1 and Section 5.2). We also evaluated the correctness and efficiency of bounded until based slicing (Section 5.3). The materials used in the experiment are all publicly available on Zenodo.[1]

5.1. Evaluation of platooning ACK pattern

In this section, we will demonstrate whether the platooning example used in this paper actually shows the effectiveness of increasing the goal achievement rate by the ACK MAPE-K pattern. The platooning SoS is a representative safety critical system, and the most important goal is not to have operating failures that can develop into human casualties among various goals. In addition, the success of the system is determined by the final arrival of the subsystems to their respective destinations. Therefore, we selected failure rate reduction (Section 5.1.1) and subsystem separation success rate (Section 5.1.2) as metrics to show the effectiveness.

Setup. To evaluate the ACK pattern, we prepared the existing VENTOS platooning simulator (Amoozadeh et al., 2015), PLTBench (Hyun et al., 2021b) (which is a failure scenario bench), the proposed platooning simulator that applied the ACK pattern, and PRISM version 4.4 (Kwiatkowska et al., 2002) as a verification tool in the pattern. The simulators and verification tools were run on an Intel i7-7700 at 3.6 GHz with 16 GBs of memory on a 64-bit Windows 10 machine. We classified the failure scenarios that occur when performing 1000 simulations with the existing VENTOS according to the failure types of PLTBench.

The 10 failure types of PLTBench are listed in Table 1. Case 1 means that during the merge operation of two platoons, another rear platoon leader requests a merge operation and fails for good. Case 2 means that during the split operation, the rear platoon leader continuously requests the original front leader to merge and the request fails. Case 3 means that during the leader leave operation (LLeave), the rear platoon leader continuously requests the original front leader to merge and the request fails. Case 4 means that during the follower leave operation (FLeave), the rear platoon leader continuously requests the original front leader to merge and the request fails. Case 5 means that during the split operation, the rear platoon leader requests the newly split platoon to merge and the request fails. Case 6 means that during the leader leave operation, the rear platoon leader requests the new platoon leader to merge and the request fails. Case 7 means that during the leader leave operation, the new platoon leader requests the left leader to merge and the request fails. In a platoon, there are a leader, middle followers, and the last follower. Case 8 means that during the middle follower leave operation (MFLeave), the intermediate platoon leader requests the left manual vehicle to merge and the request fails. Case 9 means that during the middle follower leave operation, the rear platoon leader requests the intermediate leader to merge and the request fails. Case 10 means that during the split operation in the last follower leave operation (EFLeave), the rear platoon leader requests the left manual vehicle to merge and the request fails.

To test whether the follower's leave success can be guaranteed, there are restrictions for the follower leave scenario. The road and environment are not changed, and only platoon configurations change. There are 4 lanes, the maximum vehicle speed is 30 km/h, and each scenario includes four to five maneuvers. One of those maneuvers involves the leader or follower leaving the platoon and heading toward another destination.

When SMC was performed with PRISM under the ACK pattern, the simulation method was a confidence interval with 0.01 confidence level. Whenever verification was performed, 1000 samples were generated.

5.1.1. How much can the ACK pattern reduce reconfiguration failures that present in PLTBench?

Method. We compared the number of failure-inducing state transitions (*e.g.*, recursive transitions from *state_platoon Leader* to *state_sendMergeReq*) and the average number of failures in each type of failure scenarios (Hyun et al., 2021b).

Results. Table 2 shows the results of executing the existing VENTOS platooning and the ACK pattern-applied platooning SoS. When the ACK pattern was applied to all failure types, failure-inducing state transitions were reduced by at least 87.6%, with an average of 93.3%. In one scenario, the incidence rate of each type of failure was reduced by at least 70%, with an average of 97%. This result shows that it is a compounding-error problem (Da Silva and Dobránszki, 2017), where small state transitions and missing requests accumulate, leading to SoS failures.

[1] https://zenodo.org/record/8314278 and https://zenodo.org/record/8314259

Table 2
Experimental results of effectiveness of the ACK pattern.

Failure type	CASE1	CASE2	CASE3	CASE4	CASE5	CASE6	CASE7	CASE8	CASE9	CASE10
ST # (VENTOS)	605.60	1085.50	443.00	1550.00	492.50	253.00	548.74	354.93	712.21	731.15
ST # (CVSoS-A)	55.47	21.00	55.00	24.00	25.50	29.50	29.99	27.17	40.99	51.06
Reduction rate (%)	90.84	98.1	87.6	98.5	94.8	88.3	94.5	92.3	94.2	93
Failure # (VENTOS)	202.91	342.50	21.00	239.00	288.50	196.00	258.87	39.93	46.52	80.07
Failure # (CVSoS-A)	1.53	0.00	5.00	0.00	0.00	0.50	0.98	0.04	0.17	1.36
Reduction rate (%)	99.24	100	76.2	100	100	99.7	99.6	99.9	99.6	98.3
Distance to complete LEAVE (m)	174.16	118.37	379.30	148.32	95.04	128.74	161.57	144	183.74	224.06

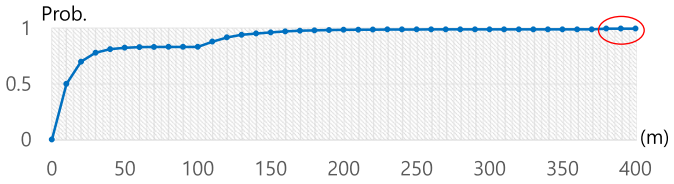


Fig. 11. Calculated leave-success rates in component P.

5.1.2. Can ACK pattern ensure the follower's goal achievement rate?
Method. We calculated the distance required to guarantee the follower's leave success based on the verification results. We then compared the calculated distance and distance from leave requests to leave successes in the failure scenarios.

Results. Component P calculates how far before the leave should be requested by increasing the distance parameter until the leave-success rate reaches 1.0 through SMC. The calculated leave-success rates are shown in Fig. 11. The verification results indicate that leave success can be guaranteed if a leave request is sent before 380 m. As shown in the last row of Table 2, we compare the distance from the leave request to the separation in the failure scenarios. We found that leaves were completed before 380 m in all failure cases. Therefore, the decision made by component P guarantees success within the constraint described in the setup paragraphs.

5.2. Evaluation of MCIR ACK pattern

In the previous section, we used a platooning SoS as a representative example used in the paper to demonstrate the effectiveness of the ACK MAPE-K pattern. In addition, we want to show that the proposed ACK pattern can be utilized independently for other acknowledged SoSs. We apply the proposed ACK pattern to MCIR SoS and demonstrate its effectiveness. An acknowledged SoS shows subsystems collaborating according to the upper system to achieve a common goal. In MCI situations, the overriding goal is to increase the life-saving rate. Therefore, the goal achievement rate of the MCIR SoS was selected as a metric to show the effectiveness of the ACK pattern (Section 5.2.1).

Setup. To generalize the proposed approach, we applied the ACK pattern to the MCIR SoS domain as well. Because we cannot invoke an MCI situation, the proposed approach was applied on the simulation of an example MCI case as follows.

An example MCI case. A fire broke out in the 24-storey building of flats. The fire was started by a malfunctioning fridge-freezer on the fourth floor. It spread rapidly up the building's exterior, bringing fire and smoke to all the residential floors. This was due to the building's cladding, the external insulation and the air gap between which enabled the stack effect. The fire burned for about 60 h before finally being extinguished. More than 250 Fire Brigade firefighters and 70 fire engines were involved from stations across London in efforts to control the fire, and rescue residents. More than 100 ambulance Service crews on at least 20 ambulances attended, joined by specialist paramedics from the Ambulance Service's Hazardous Area Response Team. The police and air ambulance also assisted the rescue effort.

An MCI SoS goal and constituent systems. Constituent systems of an acknowledged SoS cooperate with each other for a common goal. In the case of the aboce MCI SoS scenario, the goal is to rescue people from a burning building. Several constituent systems are required to achieve the common goal. The constituent systems include helicopters and firefighting systems, which are systems that will rescue people in the air and on land. A transport system to transport rescued people to the hospital is needed, and the medical system that provides medical services should be interlocked as well. A control tower exists so that all these systems can cooperate. However, each system has autonomy in dealing with its area of expertise.

Park et al. (2019) proposed dynamic simulation approach that modeled the MCI response as an SoS (Grenfell Tower Inquiry, 2018). The proposed simulator reflects the actual fire pumps' arrival times, fire spread speed, firefighters, ambulances, and hospitals. We injected stimuli and the execution plan generated by CVSoS into the dynamic simulator during the simulation. As shown in Fig. 12, the continuous verification was conducted by (1) constantly reflecting the changed simulated MCI world to a PRISM model, (2) verifying the probability that all people would be rescued, and (3) injecting stimuli based on verification result. We focuses primarily on frame-counts of 600 to 1000 in the fire scenario. During those frame counts, the fire spread speed dramatically increased. We assumed that stimuli based on the verification results were injected every 100 frames.

5.2.1. Can ACK pattern increase the MCIR goal achievement rate?
Method. To show the effectiveness of CVSoS, we compared the number of rescued people in the MCIR scenario to the CVSoS-applied scenarios. There were 10, 40, 70, and 100% scenarios in the CVSoS-applied scenarios. The 40% scenario showed that CVSoS added firefighters every 100 frames when the verification result of goal achievement was lower than 40%.

Results. Fig. 13 shows the number of firefighters increasing over time. The real scenario shows that the fire pump did not arrive from frames 600 to 1000, when the fire began to spread rapidly.

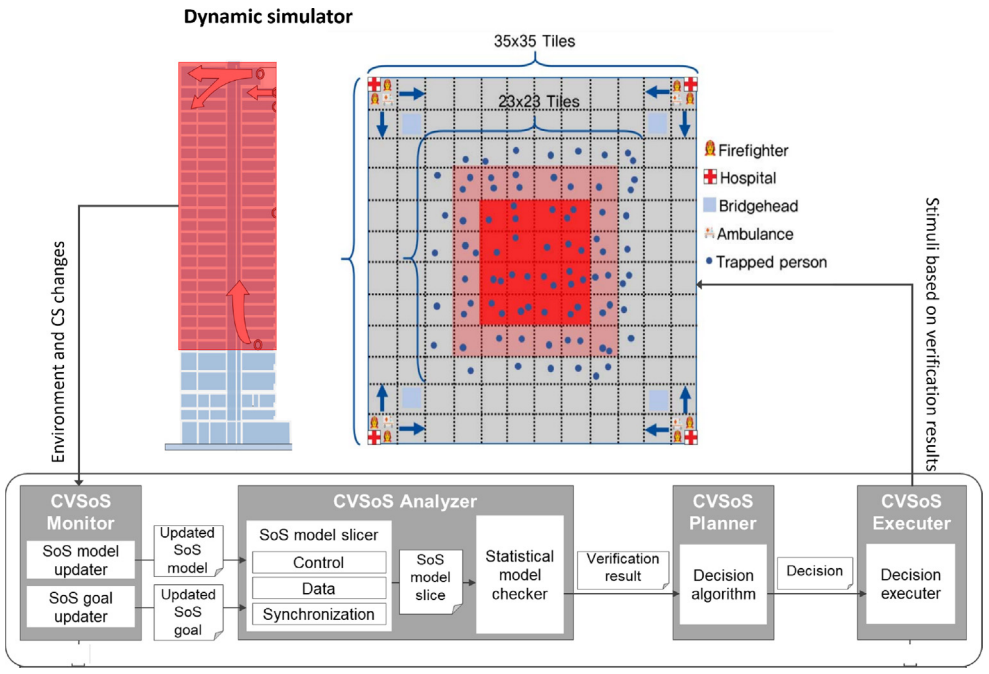


Fig. 12. Continuous verification setup using the dynamic simulator for MCIR.

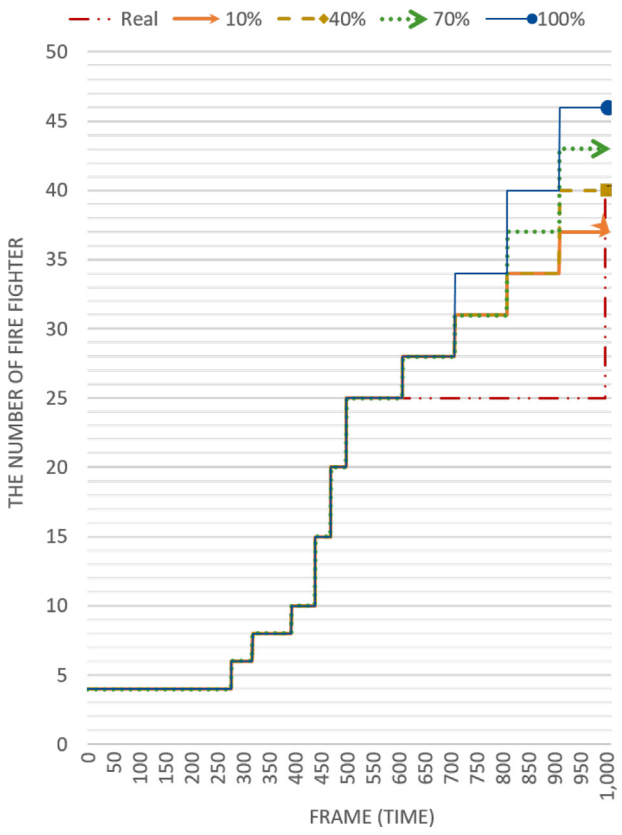


Fig. 13. The number of firefighters increasing over time.

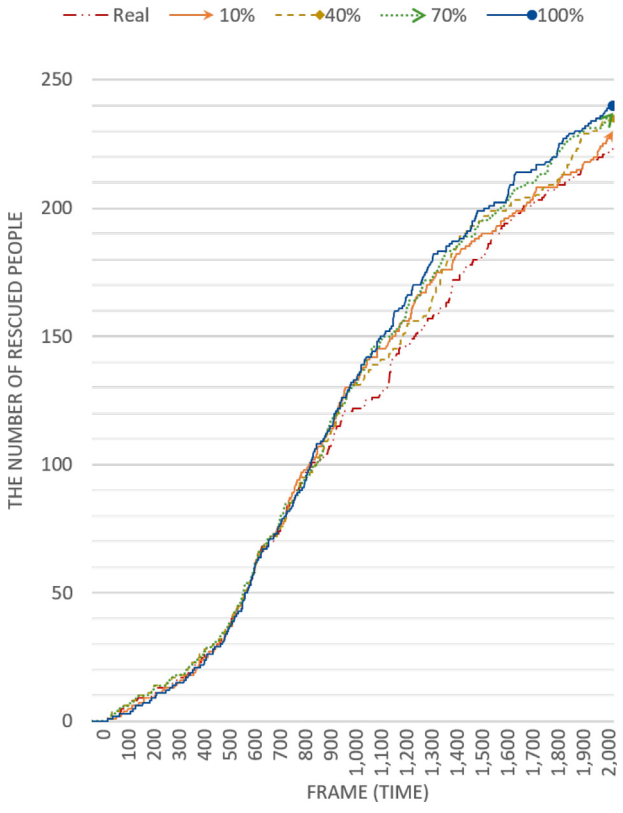


Fig. 14. The number of rescued people increasing over time.

When the proposed continuous verification approach was applied to the frames in which the fire spread rapidly, it is shown that the number of firefighters steadily increased. The number of firefighters increased slowly in the 10% scenario, and it increased fast in the 100% scenario.

Fig. 14 presents a graph showing the number of rescued people in the real scenario vs. the CVSoS-applied scenarios (10, 40, 70, and 100%) until 1976 frames passed. In the real scenario, 223 people were rescued. In the 100% scenario, 17 more could be saved at most. Interestingly, even in the 10% scenario, seven more

Table 3
Benchmark models and their properties.

Subject model	Property
BRP (Kwiatkowska et al., 2012)	$[\![\texttt{tt}\ U^{\leq 10}\ s{=}5]\!]_M(s)$
Leader_sync (Kwiatkowska et al., 2012)	$[\![\texttt{tt}U^{\leq 10}\ elected4]\!]_M(s)$
Two dice (Knuth and Yao, 1976)	$[\![\texttt{tt}\ U^{\leq 5}\ s=3]\!]_M(s)$
P2P (5) (Song et al., 2022)	$[\![\texttt{tt}\ U^{\leq 30}\ data_1+\cdots+data_3 \geq MAX]\!]_M(s)$
P2P (100) (Song et al., 2022)	$[\![\texttt{tt}\ U^{\leq 50}\ data_1+\cdots+data_5 \geq MAX]\!]_M(s)$
Platooning (5)	$[\![\texttt{tt}\ U^{\leq 30}\ V2\_state=26]\!]_M(s)$
Platooning (10)	$[\![\texttt{tt}\ U^{\leq 30}\ V2\_state=26]\!]_M(s)$

Table 4
Verification results of the original models and model slices.

Subject model	PMC verification result (%)		SMC verification result (%)		Welch's t-test
	Original	Slice	Original	Slice	1.960
BRP	98.96	98.96	99.1 (±0.77)	99.1 (±0.77)	0
Leader_sync	97.56	97.56	97.3 (±1.32)	97.4 (±1.30)	0.05
Two dice	25	25	24.6 (±3.51)	25.1 (±3.53)	0.10
P2P SoS (5)	64.40	64.40	63.7 (±3.92)	63.6 (±3.92)	0.01
P2P SoS (100)	–	–	84.8 (±2.93)	84.5 (±2.95)	0.07
Platooning (5)	99.80	99.80	99.8 (±0.36)	99.8 (±0.36)	0
Platooning (10)	–	–	99.9 (±0.26)	99.8 (±0.36)	0.23
Remarks			Accept H_0*		

*$H_0 : \mu_1 = \mu_2$ and $H_1 : \mu_1 \neq \mu_2$.

Table 5
Verification time of the original models and model slices.

Subject model	PMC model building time (s)		PMC verification time (s)		SMC verification time (s)	
	Original	Slice	Original	Slice	Original	Slice
BRP	0.32	0.26	0.00	0.00	0.08	0.05
Leader_sync	0.29	0.15	0.05	0.05	0.15	0.05
Two dice	0.06	0.00	0.0	0.0	0.02	0.00
P2P SoS (5)	0.67	0.37	0.20	0.04	0.04	0.03
P2P SoS (100)	–	–	–	–	0.31	0.07
Platooning (5)	0.13	0.00	0.0	0.0	0.05	0.01
Platooning (10)	–	–	–	–	0.16	0.01
Remarks	62.36%**		16%**		68.62%**	

** Average time reduction rates.

people could be rescued with three fewer firefighters than in the real scenario.

5.3. Evaluation of bounded until based slicing

Setup. We selected the PRISM DTMC models used in existing studies as benchmarks. All selected models passed the slicability check introduced in Section 4. The SMC verification setup is the same as that introduced in Section 5.1. The models and their bounded until verification properties are summarized in Table 3. The numbers in parentheses each indicate the number of constituent systems.

5.3.1. Is the proposed slicing experimentally correct?

Method. We statistically compared the verification results of the original models and the model slices. Because the variances of the test results differ, Welch's t-test (Welch, 1947) was performed to determine whether the means were the same or different. We tested when the degree of freedom was 1000 and the confidence level was 95%. H_0 is that the means of the two populations are the same, and H_1 is that the means of the two populations are different.

Results. The experimental results are listed in the verification result column of Table 4. When we performed exhaustive probabilistic model verification (PMC) on the benchmark models, the original model and model slice showed the same verification results. However, there are cases in which PMC is not applicable, depending on the model size. The state explosion problem (Valmari, 1996) does not occur when SMC is performed; however, the verification results show the mean and distributions. Welch's t-test was performed to test whether the verification means of the original model and the model slices were the same or different. H_0 was adopted because all t-values were less than 1.960. That is, the means of the verification results of the original model and the model slice are the same.

5.3.2. Does the proposed slicing incur lower verification cost?

Method. We compared the time required to build and verify the original model and the model slice with PMC and compared the time required to verify them with SMC.

Results. The experimental results are presented in the PMC model building time, PMC verification time, and SMC verification time columns in Table 5. When performing PMC, the PRISM model was built with multi-terminal BDDs (MTBDDs) (Kwiatkowska et al., 2002). When model slicing was performed, the number of variables included in the model was reduced. Accordingly, as shown in the PMC model building time column, the time was reduced by 62.36% and the PMC verification time was reduced by 16%. When summing up these times, the bounded until-based slicing reduces verification time by about 80% on average. The PRISM models and their slices are built as multi-terminal binary decision diagrams (MTBDDs) (Fujita et al., 1997) matrices when we verify them. Analyzing the reduction in PMC verification time of individual models, the time decreased in proportion to the decrease in the MTBDD model matrix size. In the case of the SMC

verification time in Table 5, the verification time decreased by 68.62% on average because of the bounded until-based slicing. The time for slicing the model is less than 0.01 s, so it has little effect on the reduction rate of the verification time. Therefore, the verification time can be reduced if bounded until-based slicing is applicable.

5.4. Threats to validity

Although we carefully designed and conducted experiments to demonstrate the effectiveness of the proposed approach, several issues still remain. We will point out high dependency on simulators, dealing with SoS uncertainties, and strict assumptions as follows.

High simulator dependence. Because we used the VENTOS simulator, we have to take into account the error due to the real situation not reflected in the simulation. Additionally, we cannot guarantee that the proposed approach will be effective in all platooning situations because it has not been applied to simulators other than our scenarios. However, to demonstrate the effectiveness of the proposed method as best as possible, we applied it to a mass casualty incident response system and revealed the experimental results of various cases. The MCIR case has the same problem. In the case of the MCIR SoS scenario, all assumptions about constituent systems and environmental conditions follow the papers of Park et al. (2019). Also, that the MCIR scenario assumes the acknowledged type follows the setting of Jin et al.'s study (Jin et al., 2018). It is necessary to consider the above assumptions when reproducing the experiment or interpreting the experimental results. We cannot guarantee that the proposed approach will be effective in all MCI situations because it has not been applied to simulators other than the described scenario. This threat can be complemented by specifically modeling many scenarios to which the pattern will be applied.

In an effort to bridge the gap between reality and simulator, we will discuss in more detail the simulation of packet loss in platooning SoS. In practice, packet loss can occur at any time while the platooning SoS is operating. VENTOS, which we extended to experiment with platooning SoS, supports specifying arbitrary packet loss rates. In our implementation, each platoon has a state (*e.g.*, state_sendMergeReq), and sometimes to move to a specific state, a packet containing a specific message (*e.g.*, MERGE_REQ) must be received. We can tune message dropout by adjusting the packet loss rate. However, in order to see the effect of the proposed approach in the experiment, we experimented by setting the packet loss rate specified in the control group that is, the experimental data of Hyun et al.'s study (Hyun et al., 2021a). The effect according to the packet loss rate is shown by Lei et al.'s study (Lei et al., 2011) which uses the same experimental environment as ours. As the packet loss ratio increases, the disturbance of the leading vehicle is amplified more through the platoon upstream, in other words, the platoon is more string unstable.

Dealing with SoS uncertainties. One of the main characteristics of SoS is uncertainty, which causes unexpected behaviors (Sousa-Poza et al., 2008). Unexpected behaviors can have effects for both good and bad. Systems participating in SoS create a synergistic effect, so there is a good point that the goal achievement rate is higher than expected, but on the contrary, unexpected failures can occur. In this paper, various uncertainties can be set as parameters in the simulator, and the result, that is, the failure rate can be measured. However, this study is far from eliminating the uncertainties themselves or their causes that may occur in an SoS. Logan conducted a study that could classify uncertainties (Logan, 2009). Depending on the classification, the parameters that can be set on the simulator are limited to known unknowns and cannot be controlled for unknown unknowns.

By increasing the synchronization rate for the real environment on the simulator, it will be possible to narrow the area of unknown unknowns and become a new research topic. Collecting all possible external uncertainties, for example external weather conditions and the difference between day and night, can also be a way to deal with uncertainty. In this study, the simulation was performed assuming a clear day without wind. Experimental results may differ or the system may not respond properly to other changes in the external environment. If a study on a metric that can numerically measure the degree of uncertainty is conducted, the above tolerant range of weather uncertainty will also be possible to express. After those preceding studies are conducted, it may be possible to determine how the SoS responds to external uncertainties and changes its internal model accordingly.

Restrict assumptions. We have strict assumptions for rules and values that are not manipulable in the simulator with parameters. For example, the platooning protocol follows the protocol of Amoozadeh et al.'s study (Amoozadeh et al., 2015). In addition, there is an assumption for applying the slicing approach: $\forall 0 \leq k \leq l \leq t, \forall s_0 \cdots s_l \in \chi(\phi_1, \phi_2, l, s_0), next(M, s_k) \cap AL(M) \neq \emptyset$. That means that transitions that are not included in the model slice should not be available from the states in $Path(M, s_0, t)$ satisfying $\phi_1 U \phi_2$. In particular, the assumption about slicing must be respected. The assumption was kept in our experiments, and the same verification results of the models and the slices could be obtained. However, if a new dependency arises in models, the probability that the assumption is not met may increase.

6. Related work

In this section, we compare and explain related studies by dividing topics into platooning as an SoS, continuous verification, MAPE-K patterns, and slicing.

Platooning and MCIR as an SoS. As mentioned by Svenson and Axelsson (2020), a platooning system is a good example of SoS. Many researchers have regarded platooning as SoS, and have also conducted research for analysis and verification. An SoS has uncertainties that arise when several systems gather together. Oquendo (Oquendo, 2020) conducted research on organizing the concept of uncertainty that can exist in platooning SoS and expressing it in the architecture design. In addition, Oquendo proposes an architecture description language for SoS by taking platooning SoS as an example.

As Oquendo suggests, platooning SoS has uncertainties. As a result, SoS may have unexpectedly high goal achievements and, at the same time, defects. For the high achievements through cooperation, Svenson and Axelsson (Axelsson, 2019; Svenson and Axelsson, 2021) analyzed platooning SoS through several models. Conversely, studies on unexpected defects of platooning SoS, a safety critical system, have also been conducted. In the study of Mallozzi et al. (2016), a verification of the dynamic and unpredictable change of platooning SoS was performed. Specifically, the verification was performed using an UPPAAL model checker for some properties to see if the platooning protocol could cope with uncertainties. However, the model checking technique cannot be free from the state explosion problem (Valmari, 1996) when applied to a large system like SoS. Hyun et al. 's study (Hyun et al., 2023) confirmed that several failures occurred when platooning was simulated according to the existing platooning protocol (Amoozadeh et al., 2015), and reported ten failures. In this paper, we experimented to see if the proposed approach could actually reduce the above failures.

MCIR SoS is a system in which several systems voluntarily gather to cope with man-made and natural disasters such as building fires and forest fires (Alm et al., 2006; Shen et al., 2014). MCIR SoS has been modeled and verified as directed, acknowledged, and collaborative types according to various degrees of operational and managerial independence (Seo et al., 2016). In most models (Jin et al., 2018), it is shown that the control tower plays an important role in an emergency situation. However, it is realistic that some independence of subsystems is naturally recognized because detailed control from one command center to the lowest subsystem is not possible. Therefore, in this study, the type of MCIR SoS was specified as acknowledged and experimented.

Self-adaptation in an SoS. One of the important focuses of our study is to allow SoSs to adapt themselves according to changes in the constituent systems and environment. A MAPE-K control loop is a framework designed to adapt to changes and has been applied to various distributed systems. A study by Vromant et al. (2011) applied MAPE-K to self-adapt to situations where cameras are added or disabled in a traffic system Weyns and Andersson (2013) proposed local, regional, and collaborative adaptation architectures for SoSs. Our proposed study is similar to that of Weyns et al. in helping self-adaption of SoS. A further consideration in our study is that there is a hierarchical order between constituent systems as a characteristic of the acknowledged
SoS.

In Edwards et al.'s study (Edwards et al., 2009), two meta-level components were utilized, and self-adaptation between hierarchical mobile robots was addressed. The difference from our approach is that in the analysis phase of Edwards et al.'s study, the fault toleration strategy is determined in the form of a branch statement based on the obtained data. In our study, several uncertainty parameters were applied to statistical verification to further consider the uncertainty of SoS. In Raheja et al.'s study (Raheja et al., 2010), preemption was performed to schedule time-critical adaptation. Scheduling is based on an algorithm that maximizes time-related utility for a set of concurrently executing adaptations. Platooning SoSs have a similar motivation in that they require scheduling in advance. However, Raheja et al.'s approach that plants probes in other systems is not applicable to SoSs. An SoS is a collection of independent component systems, and probes cannot be implanted by invading other constituent systems.

Continuous verification. The concept of continuous verification was first introduced by Chang et al. (1997) as a repeated verification of requirements without being regarded as a separate activity after design and implementation. Since then, continuous verification has been applied to embedded software (Cordeiro et al., 2010), self-adaptive systems (Ghezzi, 2010; Yang et al., 2014; Moreno et al., 2015), and SoSs (Tim, 2016). Continuous verification has similarities with runtime verification in that it verifies that properties are being violated against the running system (Fitzgerald and Stol, 2017; Leucker and Schallhart, 2009). In this study, we proposed a continuous verification approach, not runtime verification, considering the dynamic reconfiguration and evolutionary characteristics of SoSs (Nielsen et al., 2015). These features are a long-term development process in which new constituent systems are added, deleted, or changed in SoS. This corresponds to the concept that continuous verification applies verification activities including formal methods throughout the development process (Fitzgerald and Stol, 2017).

When continuous verification is applied to self-adaptive systems, it is combined with MAPE-K (Ghezzi, 2010). However, the single MAPE-K applied in the studies by Yang et al. (2014) and Moreno et al. (2015) is difficult to apply to distributed SoSs, and even if applied, it is applied to control the environment. Tim (2016) proposed a system engineering life cycle that enables continuous verification of SoS, but did not consider the vertical and horizontal structure between SoS components and verification efficiency. The continuous verification approach proposed by Song et al. (2022) is limited to a collaborative type SoS, and cannot be applied to platooning SoSs. In this study, we proposed a MAPE-K pattern for distributed platooning SoSs and a continuous verification approach.

In terms of monitoring done at runtime, the MAPE-K loop is similar to monitoring in service-oriented architecture (SoA). In many studies, each component of MAPE-K has been serviced and combined with SoA. For example, Vizcarrondo et al. (2017) developed MAPE-K components as services acting as an automatic manager of middleware. Ali and Solis (2015) proposed a service-oriented architecture that supports self-adaptation when reconfiguring the software architecture at runtime, such as mobile applications. The architecture is implemented hierarchically and it has distributed MAPE-K loop to respond to change.

In particular, there are studies by Cook et al. (2007) and Baresi and Guinea (2013) related to monitoring. Cook et al. (2007) conducted a study on the temporal behavior of SoS, namely the correct specification of runtime monitoring of execution. A study by Baresi and Guinea (2013) proposed an approach for collecting and analyzing runtime data in a multi-tiered system. In our study, when platooning is simulated, message exchanges between leaders and followers constituting the hierarchy are recorded in chronological order, as supported by the VENTOS simulator, and used for verification. As in Cook's study, specification using business processing execution language may help expand our study. In particular, if there is a delay in the message exchange between the leader and the followers, careful verification will be possible.

Distributed MAPE-K patterns. Distributed MAPE-K patterns (Weyns et al., 2013) applicable to distributed systems have been proposed. They can be categorized into vertical and horizontal structures. The vertical structure (Gambi et al., 2009; Kephart et al., 2003) includes hierarchical control, main/subordinates, and regional planning. The horizontal structure (Georgiadis et al., 2002) includes coordinated control and information sharing patterns. In hierarchical control and main/subordinates structures, subsystems are executed asynchronously or synchronously according to commands from upper systems. In regional planning, the upper system receives monitoring and analysis knowledge of the subsystems, performs only the planning function, and transmits the decision to the execution unit. Coordinated control and information sharing patterns have a horizontal structure, but they synchronize all controls and share monitoring information so that independent execution is impossible. According to Seo et al. (2016), platooning SoS is a hybrid of acknowledged and collaborative types. The leaders of platoons have the characteristics of a collaborative type with high operational and managerial independence. Between leaders and followers, there are execution components where management is forced by the leaders, so it has the characteristics of the acknowledged type. To apply the above distributed MAPE-K patterns to the platooning SoS, a hybrid type of the above patterns is required, independent execution should be possible to ensure the independence of each system, and the constituent systems should be able to deviate from the SoS. Additionally, the collaborative MAPE-K pattern proposed by Song et al. (2022) has a horizontal structure; therefore it is difficult to apply it to platoon leaders and followers.

Model slicing. Program slicing was originally used for program analysis (Weiser, 1981) but has since been used for software maintenance (Gallagher and Lyle, 1991), comprehension (Canfora et al., 1998), and re-engineering (Cimitile et al., 1996). In this study, slicing was used to increase the verification efficiency. Existing model slicing research has mainly focused on reducing the state of extended finite state machines (Korel et al., 2003; Androutsopoulos et al., 2009, 2011), UML state machines (Colangelo et al., 2006; Lano and Rahimi, 2011), and timed automata (Janowska and Janowski, 2006). However, because SoS requires modeling the uncertainty of the environment and other systems, it is usually implemented as a probabilistic model. Therefore, a slicing technique for the probabilistic model is required. In the study by Hur et al. (2014), the slicing technique for the probabilistic program was applied, but the observed dependence existing in the probabilistic program does not necessarily exist in other probabilistic models. In the PRISM DTMC slicing study by Song et al. (2022), slicing was performed considering the synchronization dependence between modules. When the slicing is executed, the correctness of until property-based slicing is guaranteed, but bounded until property-based slicing is not. However, in platooning SoSs, slicing based on the bounded until property is necessary for time-related goal specification. Therefore, we proposed model guidelines and a method to check the slicability to enable bounded until property-based slicing.

7. Conclusion

In this paper, we proposed CVP for continuous verification of the platooning SoS. CVP consists of an ACK pattern that enables the continuous verification of acknowledged-type platooning and bounded until based slicing that enables accurate and efficient verification. In addition to the platooning SoS, the ACK pattern is applicable to SoSs with acknowledged-type characteristics. To show the applicability of the ACK pattern, we applied it to another acknowledged type of SoS, a MCIR SoS, in the experiment and showed its effectiveness. Bounded until-based slicing can guarantee slicing accuracy and efficiency independently for PRISM DTMC models to which the model guideline is applied. We plan the following two future works.

We will propose failure coverage in units of maneuvers and state transitions, based on our continuous verification results. Platooning SoSs is an emerging field, and safety standards and verification are still lacking. Continuous verification studies such as CVP can provide basic research for the safe platooning SoSs.

We plan to extend the use of slicing to the field of model synthesis. Slicing can be used for various purposes not only for improving verification efficiency but also for model analysis. We will slice various models based on model requirements and combine them into an executable model. If the functions of other models can be reused, the effectiveness of model generation will increase.

CRediT authorship contribution statement

Jiyoung Song: Conceptualization, Data curation, Formal analysis, Investigation, Methodology / Study design, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Doo-Hwan Bae:** Conceptualization,Writing – original draft, Writing – review & editing, Funding acquisition, Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared the Zenodo link in the manuscript

References

Ali, N., Solis, C., 2015. Self-adaptation to mobile resources in service oriented architecture. In: 2015 IEEE International Conference on Mobile Services. pp. 407–414. http://dx.doi.org/10.1109/MobServ.2015.62.

Alm, A.M., Gao, T., White, D., 2006. Pervasive patient tracking for mass casualty incident response. In: AMIA Annual Symposium Proceedings, Vol. 2006. American Medical Informatics Association, p. 842.

Amoozadeh, M., Deng, H., Chuah, C.-N., Zhang, H.M., Ghosal, D., 2015. Platoon management with cooperative adaptive cruise control enabled by VANET. Veh. Commun. 2 (2), 110–123.

Androutsopoulos, K., Binkley, D., Clark, D., Gold, N., Harman, M., Lano, K., Li, Z., 2011. Model projection: Simplifying models in response to restricting the environment. In: 2011 33rd International Conference on Software Engineering. ICSE, IEEE, pp. 291–300.

Androutsopoulos, K., Clark, D., Harman, M., Krinke, J., Tratt, L., 2013. State-based model slicing: A survey. ACM Comput. Surv. 45 (4), 1–36.

Androutsopoulos, K., Clark, D., Harman, M., Li, Z., Tratt, L., 2009. Control dependence for extended finite state machines. In: International Conference on Fundamental Approaches to Software Engineering. Springer, pp. 216–230.

Axelsson, J., 2019. Business models and roles for mediating services in a truck platooning system-of-systems. In: 2019 14th Annual Conference System of Systems Engineering. SoSE, IEEE, pp. 113–118.

Baresi, L., Guinea, S., 2013. Event-based multi-level service monitoring. In: 2013 IEEE 20th International Conference on Web Services. IEEE, pp. 83–90.

Canfora, G., Cimitile, A., De Lucia, A., 1998. Conditioned program slicing. Inf. Softw. Technol. 40 (11–12), 595–607.

Chang, T.-f., Danylyzsn, A., Norimatsu, S., Rivera, J., Shepard, D., Lattanze, A., Tomayko, J., 1997. "Continuous verification" in mission critical software development. In: Proceedings of the Thirtieth Hawaii International Conference on System Sciences, Vol. 5. IEEE, pp. 273–284.

Cimitile, A., Lucia, A.D., Munro, M., 1996. A specification driven slicing process for identifying reusable functions. J. Softw. Maint. Res. Pract. 8 (3), 145–178.

Colangelo, D., Compare, D., Inverardi, P., Pelliccione, P., 2006. Reducing software architecture models complexity: A slicing and abstraction approach. In: International Conference on Formal Techniques for Networked and Distributed Systems. Springer, pp. 243–258.

Cook, T.S., Drusinksy, D., Shing, M.-T., 2007. Specification, validation and runtime monitoring of soa based system-of-systems temporal behaviors. In: 2007 IEEE International Conference on System of Systems Engineering. IEEE, pp. 1–6.

Cordeiro, L., Fischer, B., Marques-Silva, J., 2010. Continuous verification of large embedded software using SMT-based bounded model checking. In: 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems. IEEE, pp. 160–169.

Da Silva, J.A.T., Dobránszki, J., 2017. Compounding error: The afterlife of bad science. Acad. Quest. 30 (1), 65.

Dahmann, J.S., Baldwin, K.J., 2008. Understanding the current state of US defense systems of systems and the implications for systems engineering. In: 2008 2nd Annual IEEE Systems Conference. pp. 1–7. http://dx.doi.org/10.1109/SYSTEMS.2008.4518994.

Edwards, G., Garcia, J., Tajalli, H., Popescu, D., Medvidovic, N., Sukhatme, G., Petrus, B., 2009. Architecture-driven self-adaptation and self-management in robotics systems. In: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. IEEE, pp. 142–151.

Fitzgerald, B., Stol, K.-J., 2017. Continuous software engineering: A roadmap and agenda. J. Syst. Softw. 123, 176–189.

Fujita, M., McGeer, P.C., Yang, J.-Y., 1997. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Form. Methods Syst. Des. 10 (2), 149–169.

Gallagher, K.B., Lyle, J.R., 1991. Using program slicing in software maintenance. IEEE Trans. Softw. Eng. 17 (8), 751–761.

Gambi, A., Pezze, M., Young, M., 2009. SLA protection models for virtualized data centers. In: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. IEEE, pp. 10–19.

Georgiadis, I., Magee, J., Kramer, J., 2002. Self-organising software architectures for distributed systems. In: Proceedings of the First Workshop on Self-Healing Systems. pp. 33–38.

Ghezzi, C., 2010. Adaptive software needs continuous verification. In: 2010 8th IEEE International Conference on Software Engineering and Formal Methods. IEEE, pp. 3–4.

Ghezzi, C., Sharifloo, A.M., 2013. Dealing with non-functional requirements for adaptive systems via dynamic software product-lines. In: Software Engineering for Self-Adaptive Systems II. Springer, pp. 191–213.

Grenfell Tower Inquiry, 2018. Grenfell tower-fire safety investigation: The fire protection measures in place on the night of the fire, and conclusions as to: the extent to which they failed to control the spread of fire and smoke; the extent to which they contributed to the speed at which the fire spread. Fire Saf. Eng.

Hansson, H., Jonsson, B., 1994. A logic for reasoning about time and reliability. Form. Asp. Comput. 6 (5), 512–535.

Hur, C.-K., Nori, A.V., Rajamani, S.K., Samuel, S., 2014. Slicing probabilistic programs. ACM SIGPLAN Notices 49 (6), 133–144.

Hyun, S., Liu, L., Kim, H., Cho, E., Bae, D.-H., 2021a. An empirical study of reliability analysis for platooning system-of-systems. In: 2021 IEEE International Symposium on Autonomous Vehicle Software. IEEE AVS 2021, IEEE.

Hyun, S., Liu, L., Kim, H., Cho, E., Bae, D.-H., 2021b. An empirical study of reliability analysis for platooning system-of-systems. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion. QRS-C, IEEE, pp. 506–515.

Hyun, S., Song, J., Jee, E., Bae, D.-H., 2023. Timed pattern-based analysis of collaboration failures in system-of-systems. Journal of Systems and Software 198, 111613.

Hyun, S., Song, J., Shin, S., Bae, D.-H., 2019. Statistical verification framework for platooning system of systems with uncertainty. In: 2019 26th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 212–219.

Janowska, A., Janowski, P., 2006. Slicing of timed automata with discrete data. Fund. Inform. 72 (1–3), 181–195.

Jin, M., Shin, D., Bae, D.-H., 2018. Abc+ extended action-benefit-cost modeling with knowledge-based decision-making and interaction model for system of systems simulation. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing. pp. 1698–1701.

Kephart, J., Kephart, J., Chess, D., Boutilier, C., Das, R., Kephart, J.O., Walsh, W.E., 2003. An architectural blueprint for autonomic computing. In: IBM White paper. pp. 2–10.

Knuth, D., Yao, A., 1976. Algorithms and Complexity: New Directions and Recent Results. Academic Press (Chapter The complexity of nonuniform random number generation).

Korel, B., Singh, I., Tahat, L., Vaysburg, B., 2003. Slicing of state-based models. In: International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.. IEEE, pp. 34–43.

Kwiatkowska, M., Norman, G., Parker, D., 2002. PRISM: Probabilistic symbolic model checker. In: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Springer, pp. 200–204.

Kwiatkowska, M., Norman, G., Parker, D., 2012. The PRISM benchmark suite.

Lano, K., Rahimi, S.K., 2011. Slicing techniques for UML models. J. Object Technol. 10 (11), 1–49.

Legay, A., Delahaye, B., Bensalem, S., 2010. Statistical model checking: An overview. In: International Conference on Runtime Verification. Springer, pp. 122–135.

Lei, C., van Eenennaam, E.M., Wolterink, W.K., Karagiannis, G., Heijenk, G., Ploeg, J., 2011. Impact of packet loss on CACC string stability performance. In: 2011 11th International Conference on ITS Telecommunications. pp. 381–386. http://dx.doi.org/10.1109/ITST.2011.6060086.

Leucker, M., Schallhart, C., 2009. A brief account of runtime verification. J. Log. Algebr. Program. 78 (5), 293–303.

Logan, D.C., 2009. Known knowns, known unknowns, unknown unknowns and the propagation of scientific enquiry. J. Exp. Bot. 60 (3), 712–714.

Maier, M.W., 1998. Architecting principles for systems-of-systems. Syst. Eng. J. Int. Counc. Syst. Eng. 1 (4), 267–284.

Mallozzi, P., Sciancalepore, M., Pelliccione, P., 2016. Formal verification of the on-the-fly vehicle platooning protocol. In: International Workshop on Software Engineering for Resilient Systems. Springer, pp. 62–75.

Mignogna, A., Mangeruca, L., Boyer, B., Legay, A., Arnold, A., 2013. SoS contract verification using statistical model checking. In: Larsen, K.G., Legay, A., Nyman, U. (Eds.), Proceedings 1st Workshop on Advances in Systems of Systems. AiSoS 2013, Rome, Italy, 16th March 2013, In: EPTCS, vol. 133, pp. 67–83. http://dx.doi.org/10.4204/EPTCS.133.7.

Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B., 2015. Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, pp. 1–12.

Nielsen, C.B., Larsen, P.G., Fitzgerald, J., Woodcock, J., Peleska, J., 2015. Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput. Surv. 48 (2), 1–41.

Oquendo, F., 2020. Fuzzy architecture description for handling uncertainty in IoT systems-of-systems. In: 2020 IEEE 15th International Conference of System of Systems Engineering. SoSE, IEEE, pp. 000555–000562.

Park, S., Belay, Z.M., Bae, D.-H., 2019. A simulation-based behavior analysis for mci response system of systems. In: 2019 IEEE/ACM 7th International Workshop on Software Engineering for Systems-of-Systems (SESoS) and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems. WDES, IEEE, pp. 2–9.

Raheja, R., Cheng, S.-W., Garlan, D., Schmerl, B., 2010. Improving architecture-based self-adaptation using preemption. In: Self-Organizing Architectures: First International Workshop, SOAR 2009, Cambridge, UK, September 14, 2009, Revised Selected and Invited Papers. Springer, pp. 21–37.

Seo, D., Shin, D., Baek, Y.-M., Song, J., Yun, W., Kim, J., Jee, E., Bae, D.-H., 2016. Modeling and verification for different types of system of systems using prism. In: 2016 IEEE/ACM 4th International Workshop on Software Engineering for Systems-of-Systems. SESoS, IEEE, pp. 12–18.

Shen, W.-f., Jiang, L.-b., Jiang, G.-y., Zhang, M., Ma, Y.-f., He, X.-j., 2014. Development of the science of mass casualty incident management: reflection on the medical response to the Wenchuan earthquake and Hangzhou bus fire. J. Zhejiang Univ. Sci. B 15 (12), 1072.

Song, J., Kang, J., Hyun, S., Jee, E., Bae, D.-H., 2022. Continuous verification of system of systems with collaborative MAPE-K pattern and probability model slicing. Inf. Softw. Technol. 106904.

Sousa-Poza, A., Kovacic, S., Keating, C., 2008. System of systems engineering: an emerging multidiscipline. Int. J. Syst. Syst. Eng. 1 (1–2), 1–17.

Svenson, P., Axelsson, J., 2020. Situation awareness and decision making for constituent systems. In: 2020 IEEE 15th International Conference of System of Systems Engineering. SoSE, pp. 361–366. http://dx.doi.org/10.1109/SoSE50414.2020.9130539.

Svenson, P., Axelsson, J., 2021. Should I stay or should I go? How constituent systems decide to join or leave constellations in collaborative SoS. In: 2021 16th International Conference of System of Systems Engineering. SoSE, IEEE, pp. 179–184.

Tim, L., 2016. Tool and techniques–DANSE. INSIGHT 19 (3), 55–58.

Tsugawa, S., Jeschke, S., Shladover, S.E., 2016. A review of truck platooning projects for energy savings. IEEE Trans. Intell. Veh. 1 (1), 68–77.

Valmari, A., 1996. The state explosion problem. In: Advanced Course on Petri Nets. Springer, pp. 429–528.

Vizcarrondo, J., Aguilar, J., Exposito, E., Subias, A., 2017. MAPE-K as a service-oriented architecture. IEEE Lat. Am. Trans. 15 (6), 1163–1175. http://dx.doi.org/10.1109/TLA.2017.7932705.

Vromant, P., Weyns, D., Malek, S., Andersson, J., 2011. On interacting control loops in self-adaptive systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 202–207.

Weiser, M., 1981. Program slicing. In: Proceedings of the 5th International Conference on Software Engineering. IEEE Press, pp. 439–449.

Welch, B.L., 1947. The generalization of 'STUDENT'S'problem when several different population varlances are involved. Biometrika 34 (1–2), 28–35.

Weyns, D., Andersson, J., 2013. On the challenges of self-adaptation in systems of systems. In: Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems. pp. 47–51.

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., Göschka, K.M., 2013. On patterns for decentralized control in self-adaptive systems. In: Software Engineering for Self-Adaptive Systems II. Springer, pp. 76–107.

Yang, W., Xu, C., Liu, Y., Cao, C., Ma, X., Lu, J., 2014. Verifying self-adaptive applications suffering uncertainty. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. ACM, pp. 199–210.

Jiyoung Song is an assistant professor at the Computer Engineering Dept., Hannam University in Republic of Korea. She served as a Postdoctoral Researcher in Electronics and Communications Research Institute in Republic of Korea in 2022. She received her Ph.D.degree in Computer Science from KAIST. Her research interests include software analysis, software testing, software verification, and system of system engineering.

Doo-Hwan Bae is a Professor at the School of Computing, KAIST in Republic of Korea. He had served as the Head of the School of Computing, KAIST from 2012 to 2016. He received his BS degree in the Seoul National University, Korea in 1980 and his Ph.D. degree in Computer and Information Sciences in the University of Florida in 1992. Since 1995, he has been with the School of Computing (formerly Department of Computer Science until 2014), KAIST. His research interests include model-based software engineering, quality-driven software development, and modeling & verification of system of systems.