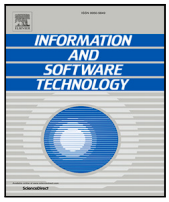
Information and Software Technology 147 (2022) 106904

Contents lists available at ScienceDirect

Information and Software Technology

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



Continuous verification of system of systems with collaborative MAPE-K pattern and probability model slicing

Jiyoung Song *, Jeehoon Kang, Sangwon Hyun, Eunkyoung Jee, Doo-Hwan Bae

Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea

ARTICLE INFO

MSC:
93C40
68N30

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

ABSTRACT

The phenomenon of cooperation among independent systems to achieve common goals has been growing. In this regard, the concept of *system of systems* (SoS), wherein numerous independent systems cooperate with each other, has been proposed. The key characteristic of an SoS is the *operational and managerial (O/M) independence* of each constituent system (CS). Each CS of a collaborative SoS with high O/M independence provides different levels of *internal-knowledge* sharing and is entitled to voluntary participation in the SoS (*i.e.*, *dynamic reconfiguration*). To increase goal-achievement rate, we need to verify SoS considering the knowledge-sharing and dynamic reconfiguration constraints.

The dynamic reconfiguration of SoSs can be managed using continuous verification, which involves environment monitoring, modeling systems for operation in changing environments, and verifying the model runtimes. However, O/M independence introduces the following challenges: (1) the low knowledge-sharing level causes inaccurate modeling, which leads to inaccurate verification results, and (2) dynamic reconfiguration requires frequent re-verification at runtime, which incurs high verification costs.

In this paper, we propose a continuous-verification-of-SoS (CVSoS) approach to solve these two challenges. To address the low knowledge-sharing level, we propose the *collaborative MAPE-K* pattern. The key to collaborative MAPE-K is the retrieval of knowledge from the other collaborating CSs. To address dynamic reconfiguration, we propose a new slicing algorithm for SoS models. This algorithm promotes *synchronization dependence*, which is essential for representing interactions between CSs. Furthermore, we demonstrate the accuracy of this algorithm.

We evaluated CVSoS across multiple SoS domains, which revealed that the SoS goal-achievement rate increases by up to 64% using the collaborative MAPE-K pattern and that slicing the benchmark and SoS models improved the verification time by an average of 67%.

1. Introduction

The prevalence of the phenomenon involving cooperation among independent systems to achieve common goals has been increasing. For instance, e-commerce—wherein a number of buyers and sellers communicate with each other—has been developed to achieve the shared goal of product sale and delivery. Additionally, although self-driving vehicles can operate independently, they can drive cooperatively in a platoon to improve efficiency. Moreover, although medical, traffic management, power-grid, and IoT systems are usually operated independently, their operations are coordinated to improve goal-achievement in certain scenarios [1].

The concept of a *system of systems* (SoS) has been proposed to model the cooperation needed among numerous independent systems to achieve common goals [2]. The defining characteristic of an SoS is the *operational and managerial (O/M) independence* of each constituent system (CS). This O/M independence allows CSs to freely participate in an SoS and adapt to changes in the SoS configuration. SoSs can be categorized into directed, acknowledged, collaborative, or virtual types, based on their characteristics. This study focuses on collaborative SoSs; they exhibit high O/M independence, have an SoS goal, and offer limited centralized management [1]. Knowledge-sharing among the CSs of such SoSs cannot be coerced owing to the limited centralized management offered. CSs of collaborative SoSs exhibit low knowledge-sharing levels, which depend on their O/M rules [3]. Jin et al. [3] experimentally demonstrated that the absence of knowledge-sharing lowers the SoS goal-achievement rate.

To improve SoS goal-achievement rates, each CS should adapt to the changes in other CSs according to the O/M independence characteristic.

* Corresponding author.
E-mail address: jysong@se.kaist.ac.kr (J. Song).

https://doi.org/10.1016/j.infsof.2022.106904
Received 11 September 2021; Received in revised form 24 February 2022; Accepted 8 March 2022
Available online 19 March 2022

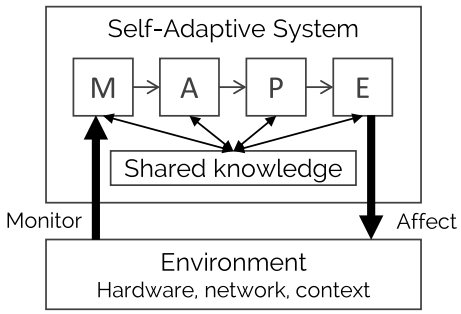


Fig. 1. Self-adaptive system adapting to environmental changes using MAPE-K [4].

To facilitate such adaptation while satisfying system requirements, continuous verification [5,6] was proposed in the context of self-adaptive systems to ensure that the self-adaptive systems behave as expected during runtime. Inspired by reinforcement learning [7], continuous verification iterates through three steps: environmental-change monitoring, analysis of monitoring-data, and verifying systems based on the analysis [5].

A combination of the continuous verification process and the monitor-analyze-plan-execute over a shared knowledge (MAPE-K) [4] is used herein, as shown in Fig. 1. Continuous verification using MAPE-K [6,8] consists of the monitoring component (M), which collects raw data from environments; the analysis component (A), which analyzes and models environmental changes to determine whether a new plan is needed; the planning component (P), which verifies modified models and establishes a strategy for the next execution, and the execution component (E), which executes the new plan within the environments. Each component processes the data obtained from the environments and other components to produce knowledge [9]. This knowledge is shared with the other components.

The concept of continuous verification using MAPE-K can be applied to each CS for better goal-achievement in dynamically reconfigured SoSs. However, O/M independence impedes continuous verification by each CS because of the following two characteristics owing to O/M independence: (1) the low *internal knowledge* level of CSs, and (2) *dynamic reconfiguration* of SoSs. The low internal knowledge-sharing level leads to inaccurate SoS modeling, and frequent SoS reconfiguration incurs high verification costs. Therefore, we propose a continuous-verification-of-SoS (CVSoS) approach that solves both these problems. We describe the problems in the remainder of this section.

1.1. Problem 1: Inaccurate probabilistic model

When traditional continuous verification using MAPE-K is applied to each CS, the (M) and (A) components cannot model the SoS accurately because each CS contains internal knowledge undisclosed to the remaining CSs. In the extreme case, a black-box CS conceals all its internal knowledge, including knowledge concerning resources, state transitions, and errors [10]. Consequently, no information is available to the (M) components of other CSs. The monitored behavior of the remaining CSs can be represented as probabilities, and the (A) component can model the SoS using a probabilistic model. The incorrectly estimated probability due to insufficient information, in turn, may cause ineffective verification of SoS goals by the (P) component.

Several approaches have been proposed to improve the accuracy of modeling surrounding systems and environments in the context of self-adaptive systems [8,11,12]. Decentralized MAPE-K patterns [11] have been proposed to model controllable systems and allow knowledge-sharing between them; this is achieved by designing them to support synchronization or share a hierarchy. For example, znn.com, a multimedia news provider, was implemented using subsystems and a single hierarchical management system that collected knowledge from the subsystems [13]. However, the decentralized MAPE-K pattern under the hierarchy cannot be applied because CSs are not controlled by other CSs. Additionally, unlike subsystems of the decentralized MAPE-K pattern, CSs are not designed to share their internal knowledge.

While modeling uncontrollable environments, raw data obtained from sensors are actively utilized. Considering other CSs in such environments, attaching sensors and applying sensor data-quality improvement techniques can aid in accurate model representation. However, this approach yields inaccurate models and causes privacy concerns. According to Kodesh [14], 40% of the data obtained from IoT sensors is spurious; hence, the data is likely to develop the wrong model. Furthermore, unanticipated tracking of CSs through sensors can interfere with their O/M independence [15]. Thus, sensor data-quality improvement is excluded from the scope of this study.

1.2. Problem 2: High verification cost

Owing to O/M independence, each CS is free to leave and participate in an SoS, which causes dynamic reconfiguration of the SoS. Consequently, when the SoS configuration is updated, the verification result of the previous configuration loses its utility, thereby necessitating a promptly updated verification result.

Continuous verification approaches have been proposed to verify the collective goals of distributed systems in the presence of dynamic reconfigurations [6,8]. These approaches generally combine MAPE-K [16] and probabilistic model checking (PMC) [17]; the system is represented using a probabilistic model, and its goals are verified with respect to the model using PMC. These approaches can be applied to an SoS; however, PMC suffers from the state explosion problem [18]. This problem is aggravated by the following factors: (1) dynamic reconfiguration may necessitate frequent re-verification during runtime, and (2) the model updated via collaborative MAPE-K tends to be larger than the original if the SoS being reconfigured is large and complex.

To reduce the verification cost, statistical model checking (SMC) has been proposed [19,20]. SMC executes finite simulations of the estimated model until adequate statistical evidence has been collected [21]. Although SMC mitigates the state explosion problem, Ballarini et al. [22] reported that the simulation times for SMC tools such as PRISM [23], PLASMA-lab [24], and UPPAAL-SMC [25] increase exponentially with the number of CSs. This indicates that SMC alone is insufficient to achieve the desired degree of scalability for an SoS.

1.3. Contributions and outline

This paper makes the following contributions to the continuous verification of an SoS that consists of a collaborative MAPE-K pattern and model-slicing.

- To solve the inaccurate probabilistic model problem, we propose a *collaborative MAPE-K* pattern that enables sharing of internal knowledge from a CS's (E) component to other CSs' (M) components (Section 2).
- To further reduce verification costs, we perform *model slicing* alongside utilizing SMC. To address synchronization among CSs during slicing, we introduce a novel *synchronization dependence* class (Section 2).
- To demonstrate the increase in modeling accuracy of each component owing to knowledge-sharing, we clarify the roles of the (M), (A), (P), and (E) components. Each component is described in detail, with examples (Section 3).
- We propose a new slicing algorithm based on synchronization dependence arising from cooperation among CSs. We prove our slicing algorithm's correctness (Section 5).
- We review the discrete time Markov chain (DTMC) model and the PRISM programming language [23] to describe SoS models. Hence, we present the first formal semantics of PRISM's *synchronized commands* that model inter-CS interactions. We also review probabilistic computation tree logic (PCTL) to describe SoS goals (Section 4).

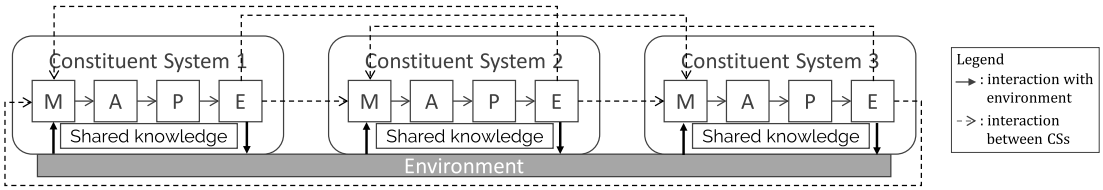


Fig. 2. Collaborative MAPE-K pattern.

- We evaluate the accuracy of SoS models and the effectiveness of the collaborative MAPE-K on multiple SoS domains. The experimental results show that the collaborative MAPE-K pattern improves SoS goal achievement by 64% at most, and model slicing reduces the verification cost by up to 67% (Section 6).

Finally, we discuss related works (Section 7), future work (Section 8), and provide a conclusion to this paper (Section 9).

2. Overview of CVSoS

This section provides an overview of the proposed CVSoS, which can solve the two problems mentioned in Section 1. In Section 2.1, we introduce CVSoS and how CVSoS solves the problem caused by O/M independence. To aid in understanding, we describe the peer-to-peer (P2P) file sharing system from the SoS perspective (Section 2.2) and use it as a running example throughout this paper.

2.1. Continuous verification of SoS

We propose a continuous verification approach for SoS that consists of a collaborative MAPE-K pattern that increases the accuracy of the SoS model alongside model slicing, which can mitigate the high verification costs. We apply the proposed approach to the collaborative type SoS. The collaborative type SoS has the notion of centralized management, but the central authority is very limited or absent [1]. Because CSs are not controlled from the center, the O/M independence of each CS is high. The subjects of application of the proposed approach are CS managers who set concrete O/M rules and SoS managers who support the operation of CSs but have little authority.

Collaborative MAPE-K pattern. The proposed collaborative MAPE-K pattern shown in Fig. 2 enables CSs to share their internal knowledge. The key differences from the prior MAPE-K patterns are twofold.

First, the proposed pattern allows internal knowledge sharing through one-way communication from a CS's (E) component to the other CSs' (M) components. Unlike the decentralized patterns [11], the (E)-(M) connection does not synchronize the (E) components of all CSs, and the higher-level CS does not control the lower-level CS components. For example, as shown in Fig. 2, CS1, CS2, and CS3 are horizontally arranged, and the (E)-(M) connection does not have any coercion to the other CSs. That is, the (E)-(M) connection is created for the purpose of sharing knowledge, rather than transmitting an execution command to the (E) component of another CS.

Although there are (E)-(M) connections among CSs, the (E)-(M) connections do not expose all O/M rules of CSs because (E) components transmit selected knowledge. For example, CS1 in Fig. 2 receives execution-related knowledge from CS2 and CS3, but it cannot know the results of data processing from M, A, and P components of CS2 and CS3.

Second, the (A) component models the other CSs as gray boxes. In the decentralized MAPE-K patterns, the (A) component models the other CSs as white boxes because their capabilities are fully declassified. On the other hand, in our approach, the other CSs are modeled as gray boxes because some of their capabilities are hidden owing to their O/M independence. Then, the capabilities of the other CSs are estimated as a probabilistic model. The detailed description of the collaborative pattern is given in Section 3.

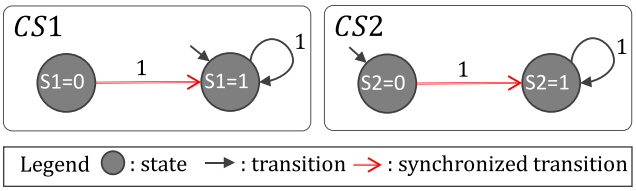


Fig. 3. Synchronization example between two CSs.

The estimated probabilistic SoS model and its goal are given to the (P) component to perform SMC. Based on the statistical verification result, the next execution plan of each CS is generated. The detailed explanations on the SoS models and its goal specification are given in Section 4.

CS managers and conceptual central management apply the proposed collaborative MAPE-K pattern to collaboration type SoS. First, (E)-(M) connections are indicated by dotted lines. The basis of these connections is designed by the central management, but CS managers set whether they will actually be connected with other CSs (*e.g.*, dynamic reconfiguration) and how much knowledge they will share. The second (A) and (P) are executed under the O/M rules of the CS managers. An example CS manager is described in Section 2.2.

Model slicing. To further reduce verification costs, we perform *model slicing* in addition to utilizing SMC. Program slicing [26] calculates a subset of statements that affects variables of interest. The slicing criteria are the SoS goals because each CS verifies that it can achieve the SoS goals. By slicing the part that affects the goal to be verified in the SoS model, the state space—as well as the simulation time—of the SoS model is significantly reduced.

However, a conventional slicing algorithm that uses control and data dependence does not preserve the semantics of the SoS model because those dependences do not present synchronization that shows cooperation between CSs within the SoS models. Therefore, rendering the conventional slicing algorithm is unusable for SoS verification.

For example, consider the synchronization between the two CSs in Fig. 3, represented as a state diagram of a DTMC. The initial states of $CS1$ and $CS2$ are $s1 = 1$ and $s2 = 0$, respectively. The two CSs are simultaneously transitioned to the $s1 = 1$ and $s2 = 1$ states if both $s1$ and $s2$ are zero. Suppose that the size of the model is reduced by conventional backward slicing to calculate the probability of reaching $s2 = 1$ from the initial state. Following the transition recursively in reverse from $s2 = 1$, a model slice that includes only $s2 = 0$ and $s2 = 1$ of $CS2$ is generated. The probability of reaching $s2 = 1$ in the initial state of the model slice is one. However, to reach $s2 = 1$, synchronized transitions should be executed. A correct slice that considers the synchronization of the two transitions should comprise the whole model. Because the initial state of the correct slice is $s1 = 1$ and $s2 = 0$, the synchronization transition cannot be executed; hence the correct probability of reaching $s2 = 1$ is zero.

To address such synchronization in slicing, we introduce an additional class of dependence called *synchronization dependence*. Our slicing technique correctly and effectively slices probabilistic SoS models. The details on model slicing are given in Section 5.

2.2. A running example—P2P file sharing SoS

A P2P file-sharing protocol can be classified as a collaborative SoS because each CS has high O/M independence and it shares its characteristics: (1) the presence of a common SoS goal, (2) the low level of knowledge sharing, and (3) dynamic reconfiguration. Recently, a SHA1 collision attack called BitErrant [27] caused by the low level of knowledge sharing has occurred in a P2P file sharing system. Next, we describe the overall construction issues of P2P file sharing SoS and concrete BitErrant attack example from the perspective of SoS.

SoS goal, construction, and operation. The goal of a P2P file-sharing SoS is to download a file without malware within a tolerable waiting time [28]. The P2P file-sharing SoS consists of multiple server CSs such as *peers* and *trackers*. A *client* is a peer that requests files and other *peers* who have the requested file will provide the file. A *tracker* is a server that holds the addresses of peers that own the requested files. The P2P file-sharing protocol divides a file into pieces, which is shared by peers. When a client executes a file request (*e.g.*, a `.torrent` file), a tracker searches for peers having the target. The tracker randomly configures a peer list and sends it to the client. The client configures the initial SoS based on the peer list. The peers participating in the SoS act according to O/M rules, such as rejecting the file transfers and controlling transfer rates. When the peers participating in the SoS send file fragments, the client compares the hash of the received file fragment with the hash value written in the `.torrent` file. If the two hash values are the same, the file fragment is received.

Low level of knowledge sharing. We can observe the low level of knowledge sharing (even hiding) of internal knowledge of the P2P file-sharing SoS in the file transfer process. For example, a BitErrant attack can occur during the process of receiving file fragments from peers. The BitErrant attack induces the client to download the malware by making the hash of the file fragment, including the malware, equal to the hash value of the clean file. The Hacker's intentional hiding and non-sharing of information among peers are manifestations of O/M independence. They also hinder the achievement of the goal of P2P file sharing SoS. As in the BitErrant example, hiding knowledge or sharing low-level knowledge by O/M rules makes modeling other CSs difficult.

Through different levels of knowledge sharing, the achievement of SoS goals can be different. CS managers can classify private and public levels according to the type of system knowledge. For example, if the peer servers' managers publicly discloses more detailed knowledge, such as location and file transfer rate of peers as well as BitErrant attack records, malware-free files can be downloaded faster by constructing the peer list based on detailed knowledge. Using sharing knowledge can identify CSs' states that hinder the achievement of the SoS goal. If there are peers who only download files and do not redistribute files, tracker and application managers can modify the policy to encourage file redistribution of peers who only pursue personal interests when updating the application version.

Dynamic reconfiguration. The P2P file sharing SoS shows characteristics of dynamic reconfiguration while the client downloads a file. While clients try to download files, the peers may not wish to provide files according to their O/M rules because doing so consumes their network and memory resources. Thus, peer configuration is dynamically reconfigured if any peer deletes files and leaves the SoS. Therefore, even if the peer list without hackers were given, the success of the file downloading is not guaranteed.

3. Collaborative MAPE-K pattern

In this section, we propose a collaborative MAPE-K pattern to address the problem of inaccurate probabilistic SoS models (Section 1.1). To tackle problems in SoS engineering, the collaborative MAPE-K improves the accuracy of the probabilistic SoS model in two ways: (1) in the (M) component, it collects other CSs' internal knowledge; and (2) in the (A) component, it updates other changing CSs as gray boxes immediately. When including these two components, the four components of collaborative MAPE-K can be explained using the P2P file-sharing SoS. The implementations of M, A, P, and E components in collaborative MAPE-K pattern are shown in Fig. 4.

Monitor. In the (M) component, a knowledge collector mainly obtains environment data from sensors and probes, and the internal knowledge of the CSs. An example internal knowledge in SoSs is CS participation caused by SoS reconfiguration. If the internal knowledge of CSs is obtained, O/M independence may be violated. Therefore, CS monitoring is proposed in the form of selectively receiving the internal knowledge through an (E)-(M) connection under mutual cooperation. The selection criteria for knowledge should be based on agreed policies among CSs and the characteristics of each SoS.

By the file-sharing protocol [29] (*i.e.* an SoS policy), a tracker can obtain the internal knowledge such as the file possessions of each peer. If the existing protocol is modified by applying our approach shown in Fig. 4, peers can also provide other knowledge to the tracker, such as online status, past malicious file transfers, volume, and time. If peers share malicious file download records to the tracker, it helps find and bypass suspected hackers.

CSs reach consensus to achieve SoS goals. In the case of P2P file-sharing SoS, file sharing should be actively performed. In order to achieve the goal, all CS servers agree that if the file transmission speed is set to 0 bit/s to other CS servers, the file reception speed will also be forced to 0 bit/sec. When the CS servers do not agree with this policy, they do not upgrade the version or use another file sharing application. The CS servers basically agree on policies including data protection, but the policies may have potential conflicts with higher-level policies. For example, if file-sharing is involved in a crime such as copyright infringement, there can be controversy over the disclosure of IP addresses.

Internal knowledge is shared without infringing the O/M independence of peers. Supposing that peers do not disclose the file transfer available time set per user according to the agreed policies among peers, the peers instead can provide knowledge, such as the time for uploading and downloading files for a certain period. Based on the knowledge provided, the tracker can infer the file-transfer available time to preserve the O/M independence of peers. The provided knowledge is delivered to the (A) component for modeling and analysis.

Analyze. In the proposed collaborative MAPE-K pattern, the (A) component models gray-boxed CSs. Unlike the lower systems in hierarchical systems, CSs are gray-boxed to each other. The (A) component infers capabilities of the gray-boxed CSs based on knowledge and reflects the capabilities to the SoS model. The knowledge selection criteria of the monitoring phase are important to the capability inference performed by (A) component. Because cooperation between CSs is important to achieve goals, the CS participation becomes an important capability. If the necessary knowledge for capability inference is not obtained, model accuracy will decrease. The result of capability inference is reflected in an estimated probabilistic model.

For example, if a peer has accepted five out of 10 data-transfer requests, the acceptance execution probability of the peer is 0.5. Apart from participation, there are other capabilities inferred, such as the maximum data-transfer speed, available time, and stability. When malicious file-download records are reported to a tracker, it can list the suspicious peers. Based on this list, the tracker can model the probability of each peer sending a malicious file.

Plan. The (P) component plans the consecutive executions affecting other CSs. The updated SoS model is given to the SoS model slicer, which slices the model based on the SoS goal to reduce verification costs. The solution to the high verification cost caused by the large and complex probabilistic models is addressed in Section 5. When the

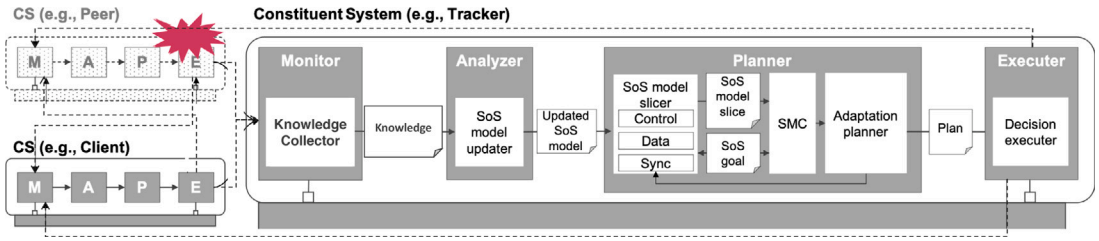


Fig. 4. P2P File-sharing SoS with collaborative MAPE-K pattern.

$$
\begin{array}{rcl}
T & ::= & \mathit{Bool} \mid \mathit{Integer} \mid \mathit{Double} \\
\mathit{uop} & ::= & - \mid \neg \\
\mathit{bop} & ::= & \times \mid \div \mid + \mid - \mid < \mid \leq \mid \geq \mid > \mid = \mid \neq \mid \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow \\
\varepsilon & ::= & \mathit{const} \mid v \mid \mathit{uop}\ \varepsilon \mid \varepsilon\ \mathit{bop}\ \varepsilon \\
\mathit{Init} \ni \mathit{init} & ::= & v : T = \mathit{const} \\
L \ni l & ::= & \mathit{String} \\
D \ni \mathit{cond} & ::= & \varepsilon \\
\mathit{Atom} \ni \mathit{atom} & ::= & \texttt{skip} \mid v' = \varepsilon \\
A \ni \mathit{act} & ::= & \mathit{atom} \mid \mathit{act}_1\ \&\ \cdots\ \&\ \mathit{act}_i \mid p_1 : \mathit{act}_1 + \cdots + p_j : \mathit{act}_j \\
C \ni c & ::= & [\,]\ \mathit{cond} \rightarrow \mathit{act} \mid [l]\ \mathit{cond} \rightarrow \mathit{act} \\
\mathit{Module} \ni m & \triangleq & \langle \texttt{list}\ \mathit{Init},\ \texttt{list}\ C \rangle \\
\mathcal{M} \ni M & \triangleq & \texttt{list}\ \mathit{Module}
\end{array}
$$

Fig. 5. PRISM syntax.

model slice is generated, an SMC engine verifies whether the current SoS model can achieve the SoS goal or not. Based on the verification results, the adaptation planner generates an execution plan, which is reflected to the SoS model and given to the model slicer. Model slicing and verification are repeated until the SoS goal is expected to be satisfiable according to the verification result.

For example, in the case of a tracker, the verification result can be used to determine the optimal peer list containing peers having high transfer speed and without malicious files. When a client request a peer list, the adaptation planner configures the candidate peers, which are repeatedly verified until it is estimated that files are downloadable.

Execute. The determined plans are executed in the (E) component. Hence, some of the CS's knowledge, such as the peer list, is declassified and voluntarily provided to the other CSs' (M) components. For example, if a client receives malware from peers, the peers are reported to a tracker as suspicious peers through the (E)-(M) communication channel. Subsequently, the continuous verification cycle is repeated.

4. SoS model and goal specification

The frequent re-verification on accurate SoS models obtained from the collaborative MAPE-K pattern (Section 3), induces higher verification costs (Section 1.2). We address this problem in Section 5. In this section, we describe the syntax and semantics of SoS models (Section 4.1) and goals (Section 4.2).

4.1. SoS models in DTMC

We model SoS as a DTMC [30], because (1) collaborative MAPE-K pattern probabilistically estimates the state of the other CSs; and (2) SoS reconfiguration is discretely performed. To represent DTMC, we use the PRISM [23] programming language for reactive modules [31] because it is widely used to model internal interaction among CSs. We model CSs as PRISM modules and CS-interactions as *synchronized commands* of multiple modules.

We review the syntax and semantics of PRISM using the P2P file-sharing example. Thus, we formalize the semantics of synchronization modeled with synchronized commands, which has not been clearly covered in prior works [32,33].

Syntax. The syntax of PRISM is defined in Fig. 5. A model, $M \in \mathcal{M}$, is a list of modules; a module, $m \in \mathit{Module}$, has a variable initialization list and a command list; and a command, $c \in C$, is either non-synchronous or synchronized with a label, l. A local command consists of a Boolean expression, cond, and an action, act, updating states; and a synchronized command additionally has a label, where the commands having the same label in different modules are meant to be synchronized with each other. There are three types of actions: atomic actions (atom), composite actions ($\mathit{act}_1\ \&\ \cdots\ \&\ \mathit{act}_i$), and probabilistic choices ($p_1 : \mathit{act}_1 + \cdots + p_j : \mathit{act}_j$). Atomic actions include self-loops (`skip`) and value assignments ($v' = \varepsilon$). Composite actions and probabilistic choices are built from smaller actions. Variables have base types, such as *Bool*, *Integer*, and *Double*. Expressions include variables, constants, and binary and unary operations.

Semantics. The denotational semantics of PRISM is defined in Fig. 6. Given a model M, let $V = \{v_1, \ldots, v_l\}$ be the set of variables appearing in M. A state $s \in S$ is a tuple $(x_1, \ldots, x_l)$ where, for all i, x_i is a value for the variable v_i. The initial state s_0 is defined by the variable initialization list.

A state is transitioned by an action act. The probability of the action is denoted as $[\![\mathit{act}]\!](s, s')$. The `skip` action represents a self-transition and its probability is one if the current and the next states are the same.

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

$$[\![\texttt{skip}]\!](s,s') = \begin{cases} 1 & (if\ s' = s) \\ 0 & (otherwise) \end{cases}$$

$$[\![v' = \varepsilon]\!](s,s') = \begin{cases} 1 & (if\ s' = s[v' \mapsto [\![\varepsilon]\!](s)]) \\ 0 & (otherwise) \end{cases}$$

$$[\![act_1\ \&\cdots\&\ act_i]\!](s,s') = \sum_{t\in S} [\![act_1]\!](s,t)\cdot [\![act_2\ \&\cdots\&\ act_i]\!](t,s')$$

$$[\![p_1{:}act_1 + \cdots + p_j{:}act_j]\!](s,s') = \sum_{k=1}^{j} p_k \cdot [\![act_k]\!](s,s')$$

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

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

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

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

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

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

Fig. 6. PRISM semantics.

A state is updated by a value assignment action, $v' = \varepsilon$. If the updated state is equal to the next state, the probability of the action becomes one. The $act_1\ \&\ \cdots\ \&\ act_n$ composite action executes acts sequentially. The probability of the composite action is the sum of the probability of transitions from s state to s' when executed in order from act_1 to act_i. In a probabilistic choice, $p_1{:}act_1 + \cdots + p_j{:}act_j$, act is executed with the corresponding probability, p.

We present for the first time the formal semantics of synchronized commands in PRISM. To enumerate all possible transitions, we define a function, $next : \mathcal{M} \times S \to 2^{\mathtt{A}}$. $\mathtt{A}$ is the set of *action-labels* that can be executed in the given state s. There are two types of action-labels in $next(M, s)$: local ($\mathtt{a}_{i,j}$) and synchronized ($\mathtt{sa}_I$). The result of $next$ contains a local action-label if and only if (iff) the corresponding action's condition is met; and a synchronized action-label, $\mathtt{sa}_I$, iff the corresponding actions of multiple modules are simultaneously executable. The synchronized action-label $\mathtt{sa}_I$ provides the semantics to execute a set of actions in commands having the same label, l, under three conditions: (1) every module that have commands with l must select one of the commands with l; (2) s must satisfy all $cond$s of the selected commands with l; and (3) if no command with l is selected, nothing is executed. Action-labels are mapped for act by the $lookup$ function, and their probabilities are normalized by the size of $next(M, s)$.

```
1  module Client
2    // 0: downloadWait, 1: download(=seed)
```

```
3    s_d1: int 0;
4    s_d2: int 0;
5    [WAIT1] s_d1=0 -> (s_d1'=1);
6    [WAIT2] s_d2=0 -> (s_d2'=1);
7    [TX1] s_d1=1 -> (s_d1'=1);
8    [TX2] s_d2=1 -> (s_d2'=1);
9  endmodule
10
11 module Seeder1
12   OM1: double 0.5; // O/M independence
13   C1: double 0.5;  // corrupted portion
14   s_u1: int 0;     // 0:stopped, 1:seedWait
15   p1: bool true;
16   data1: int 0;
17   corrupt1: int 0;
18   [] s_u1=0 & !p1 -> true;
19   [] s_u1=0 & p1 -> (s_u1'=1);
20   [WAIT1] s_u1=1 -> OM1:true +
         1-OM1:(s_u1'=0)&(p1=false);
21   [TX1] true -> C1:(corrupt1'=corrupt1+1) +
         1-C1:(data1'=data1+1);
22 endmodule
23
24 module Seeder2
25   OM2: double 0.6;
26   C2: double 0.8;
27   s_u2: int 0;
28   p2: bool true;
29   data2: int 0;
30   corrupt2: int 0;
31   [] s_u2=0 & !p2 -> true;
32   [] s_u2=0 & p2 -> (s_u2'=1);
33   [WAIT2] s_u2=1 -> OM2:true +
         1-OM2:(s_u2'=0)&(p2=false);
34   [TX2] true -> C2:(corrupt2'=corrupt2+1) +
         1-C2:(data2'=data2+1);
35 endmodule
```

Listing 1: An abstract P2P file-sharing SoS PRISM model

Example. Listing 1 presents the PRISM model for the P2P file sharing SoS (Section 2.2). When a client requests a file to seeders, the seeders accept the request according to their O/M rules (`OM1` in line 12 and `OM2` in line 25). The seeders have different corrupted data rates (`C1` in line 13 and `C2` in line 26). The peers are modeled with **`module-endmodule`** keywords that have four states[1]: `stopped`, `downloadWait`, `download (=seed)`, and `seedWait` [34]. Variables `s_d1~2` in lines 3–4 indicate states of `Client`, and variables `s_u1` in line 14 indicates states of `Seeder1`. `p1` in line 15 indicates the existence of the requested file. If `Seeder1` does not have the file, `p1` is initialized to `false`. The `downloadWait` state of `Client` (line 5) is synchronized with the `seedWait` state of `Seeder1` (line 20) through the label, `WAIT1`. When `Seeder1` accepts the file request, the `download` state of `Client` (line 7) and the `seed` state of `Seeder1` (line 21) are synchronized through the label, `TX1`. Based on the corrupted data rate `C1` in line 21, uncorrupted data (`data1`) or corrupted data (`corrupt1`) are transferred. The increment of corrupted data represents transferring fragments of malware. The other seeder `Seeder2`, which functions the same as `Seeder1`, appears in lines 24–35.

4.2. SoS goals in PCTL

SoS goals are specified in a PCTL formula [35] that represents a probabilistic distribution over DTMC paths. We review the PCTL syntax and denotational semantics [35], presented in Fig. 7.

[1] The states of peers are modeled based on an open-source P2P application, Transmission. `checkWait` and `check` states that verify previous file are excluded.

Syntax. There are two types of formulas in PCTL: a state formula, ϕ, and a path formula, ψ. The state formula is built from atomic propositions and logical connectives and operators. The path formula is built from the two state formulas with an $U^{<t}$ operator. There are other operators which can be represented by until properties, such as next (X) and weak until (W).

Semantics. The semantics of state formulas for a state s in a model M are defined by a satisfaction relations $[\![\phi]\!]_M(s)$. The satisfaction relations for the state formulas, such as $\mathtt{tt}$, ap, $\neg\phi$, and $\phi_1 \wedge \phi_2$, represent whether the given state s satisfies the state formulas. $L : S \rightarrow 2^{AP}$ is a labeling function assigning atomic propositions to states. If ap is included in atomic propositions assigned to the given state s, $[\![ap]\!]_M(s)$ becomes `skip`.

The semantics of path formulas for a state s in a model, M, are defined by satisfaction relations $[\![\psi]\!]_M(s)$. By the $U^{<t}$ operator, the path formulas check states along a specific path. $\phi_1 U^{<t} \phi_2$ means that states satisfy ϕ_1 until reaching a state that satisfies ϕ_2 within t transitions. When t is finite, the probability measure for $[\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(s)$ is calculated as follows, where $\chi(t, s_0)$ is the set of finite sequences. $s_0 \ldots s_j$, starting from s_0, such that $\forall i \in [0, j)$ and $j \leq t$, we have $[\![\phi_1]\!]_M(s_i)$, $\neg [\![\phi_2]\!]_M(s_i)$, and $[\![\phi_2]\!]_M(s_j)$:

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

When t is ∞, $[\![\phi_1\ U^{\leq t}\ \phi_2]\!]_M(s)$ is equal to $[\![\phi_1\ U\ \phi_2]\!]_M(s)$. To apply the probability measure to an infinite sequence, we need a set of failure states, Q, and a set of success states, R; Q can never reach s' and R eventually reaches s' without passing Q. Each sequence in χ starting from s_0 becomes finite by reaching a state in Q or R. The details of calculating Q and R are given in the study of Hansson et al. [35]. Through the satisfaction relations for a state, s, and a path, π, PCTL formulas are inductively well-founded.

Example. The SoS goal of a P2P file sharing SoS is to download a file without malware within a tolerable time. We specify a property to find an optimal peer list that satisfies the P2P SoS goal in PCTL. Among the two seeders in Listing 1, if `Seeder1` and `Seeder2` have the file requested by `Client`, and the size of a peer list requested by `Client` is two, we can check whether the two seeders can transfer uncorrupted data through an example property. According to the syntax and semantics of PCTL, the example property *P1* is specified as

$$[\![\mathtt{corrupt1} + \mathtt{corrupt2} < 1\ \ U^{\leq \infty}\ \mathtt{data1} + \mathtt{data2} \geq \mathtt{MAX}]\!]_M(s_0) \ .$$

P1, presents the probability that corrupted data are not transferred from `Seeder1` and `Seeder2` until the client receives the entire data. If the probability is larger than a criterion set by a tracker, the tracker configures a peer list with `Seeder1` and `Seeder2`.

5. Slicing SoS models

In this chapter, we propose slicing approach that can reduce SoS verification costs. The target of model slicing is all SoS models that can be modeled with PRISM DTMC except for the virtual type SoS whose goal cannot be verified. Modeling the SoS model with PRISM DTMC has been proposed in several studies [36,37]. As discussed in Section 1.2, the conventional slicing algorithm using control and data dependence does not preserve the semantics of SoS model. We introduce an additional synchronization dependence class among CSs interacting via synchronized commands. Our slicing algorithm consists of three steps: (1) drawing the *dependency graph*; (2) calculating the set of variables that may affect the SoS goals, *influencers*; and (3) transforming the model. Finally, we prove the correctness of our slicing algorithm.

• PCTL's Syntax

$$\phi \ ::= \ \texttt{tt} \mid ap \mid \neg\phi \mid \phi_1 \ \wedge \ \phi_2 \qquad \text{(STATE)}$$

$$\psi \ ::= \ \phi_1 \ U^{\leq t}_{>p} \ \phi_2 \mid \phi_1 \ U^{\leq t}_{\geq p} \ \phi_2 \qquad \text{(PATH)}$$

• PCTL's Denotational Semantics for State Formula

$$[\![\texttt{tt}]\!]_M(s) \ \triangleq \ \texttt{skip}$$

$$[\![ap]\!]_M(s) \ \triangleq \ ap \in L(s)$$

$$[\![\neg\phi]\!]_M(s) \ \triangleq \ not \ [\![\phi]\!]_M(s)$$

$$[\![\phi_1 \wedge \phi_2]\!]_M(s) \ \triangleq \ [\![\phi_1]\!]_M(s) \ and \ [\![\phi_2]\!]_M(s)$$

• PCTL's Denotational Semantics for Path Formula

$$[\![\phi_1 \ U^{\leq t} \ \phi_2]\!]_M(\pi) \ \triangleq \ \exists i \leq t \ s.t. \ [\![\phi_2]\!]_M(s_i) \wedge \forall j : 0 \leq j < i, \ [\![\phi_1]\!]_M(s_j)$$

$$[\![\phi_1 \ U^{\leq t} \ \phi_2]\!]_M(s) \ \triangleq \ \mu^M_s(\pi \in Paths(M,s) \text{ — } \pi_0 = s \wedge [\![\phi_1 \ U^{\leq t} \ \phi_2]\!]_M(\pi))$$

Fig. 7. PCTL's syntax and denotational semantics.

5.1. Drawing a dependency graph

Given a PRISM model M, we first draw its dependency graph $\mathbf{D}(M) = (V, E)$, where V consists of condition-labels $\mathtt{c}_{i,j}$ and atomic action-labels $\mathtt{a}_{i,j,k}$:

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

where $f(C,n)$ looks up the nth atomic action in the given command. $\mathtt{c}_{i,j}$ 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 atomic action appearing kth in $M[i].C[j]$, respectively.

E consists of the conventional control and data dependence as well as the new synchronization dependence as follows:

$$\begin{aligned} E \ = & \{(v_1, \ v_2) \mid \exists i,j,k \in \mathbb{N}, \quad v_1 = \mathtt{c}_{i,j} \ \wedge \ v_2 = \mathtt{a}_{i,j,k}\} && \text{(CONTROL)} \\ & \cup \{(v_1, \ v_2) \mid Write(v_1) \cap Read(v_2) \neq \emptyset \ \} && \text{(DATA)} \\ & \cup \{(v_1, \ v_2) \mid \exists i_1,i_2,j_1,j_2 \in \mathbb{N}, \quad \exists l \in L, \quad \exists cond_1, cond_2 \in D, \\ & \qquad \exists act_1, act_2 \in A, \quad v_1 = \mathtt{c}_{i_1,j_1} \wedge \ v_2 = \mathtt{c}_{i_2,j_2} \wedge \ i_1 \neq i_2 \\ & \qquad \wedge \ M[i_1].C[j_1] = [l]cond_1 \rightarrow act_1 \\ & \qquad \wedge \ M[i_2].C[j_2] = [l]cond_2 \rightarrow act_2 \ \} \ , && \text{(SYNC)} \end{aligned}$$

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

The control dependence exists within every command and is defined as a directional relation from a condition-label, $\mathtt{c}_{i,j}$, to all atomic action-labels, $\mathtt{a}_{i,j,k}$, in the same command. For example, a command `[WAIT1]` in line 6 of Listing 1 has a control dependence connected from a condition-label (`s_d1=0`) to an atomic action-label (`s_d1'=1`) as shown in Fig. 8(a). Data dependence is the relation from writing to reading on the same variable. Taking `s_d1'=1` as an example, there are data dependencies described with dotted arrows from where the variables are assigned new values to where they are read, `s_d1=0` and `s_d1=1` in commands `[WAIT1]` and `[TX1]` of `Client`, respectively.

We propose synchronization dependence between condition-labels. By the synchronized action-label semantics, actions are executed only when the conditions of the synchronized commands are satisfied. Therefore, the conditions have dependence. We present this synchronization dependence on conditions with bidirectional arrows. For example, in order for `Seeder1` to send data to `Client` in Fig. 8(a), both CSs should be in states where data can be sent. According to the synchronization dependence, bidirectional arrows connect conditions `s_d1=0` and `s_u1=1` with `[WAIT1]`.

5.2. Calculating influencers

For a path formula ψ, let R be the set of variables in ψ. *Influencers* are variables that influence R. $Inf \ : \ \mathbf{D} \times 2^{\mathbf{V}} \rightarrow 2^{\mathbf{V}}$ function maps the dependency graph $D = (V, E)$ and R to influencers as follows:

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

The influencer calculation is recursively performed backward until there are no more vertices to include. Then, variables to be read and written in the vertices become the set, **I**, of influencers. For example, starting from the variables, `data1` and `corrupt1`, which are not only included in $P1$ but in '`data1'=data1+1`' and '`corrupt1'=corrupt1+1`', we can recursively collect variables `s_d1`, `s_u1`, and `p1` into **I** by definition. As a result, a correct model slice includes *Seeder1*, *Seeder2*, and *Client* modules, as shown in Fig. 8(a). If influencers are collected without considering the synchronization dependence, an incorrect model slice composed of only variables `data1` and `corrupt1` is obtained as shown in Fig. 8(b).

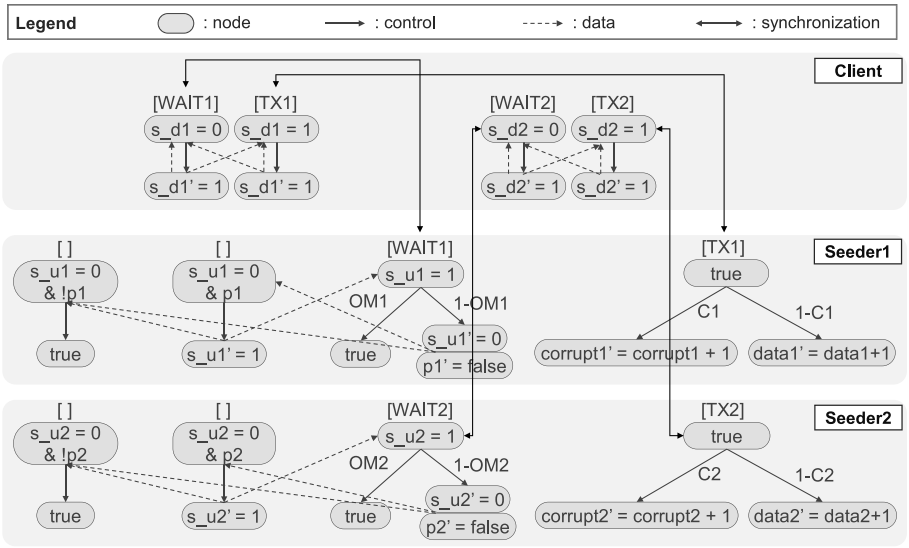
5.3. Transforming an SoS model

A PRISM model slice is generated though a slicing transformation function, SLI: $\mathcal{M} \times 2^{\mathbf{V}} \rightarrow \mathcal{M}$, defined in Fig. 9. SLI functions perform slicing based on influencers over commands of the PRISM model. As a result, a model slice contains commands that have the influencers. In the SLI transformation, $\perp$ stands for white space. That means commands that have no influencer are simply deleted. The actions without influencers are replaced by *skip*.

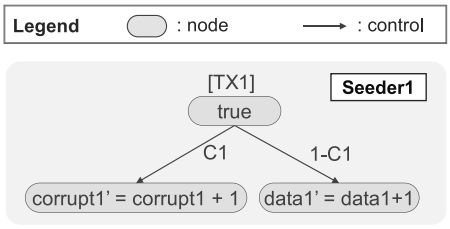
5.4. Proving correctness of the slicing algorithm

We prove correctness of the proposed slicing algorithm by dividing PCTL's until (Theorem 1) and bounded until (Theorem 2) properties. We define preliminaries as follows.

Given a PRISM model M, an initial state, s_0, and a verification property, $\phi_1 U \phi_2$, the set V of variables and the set **I** of influencers



(a) Dependency graph of Listing 1 including synchronization dependence (which is equal to a correct model slice based on $P1$)



(b) An incorrect model slice based on $P1$ only considering control and data dependence

Fig. 8. Dependency graphs of Listing 1.

are derived. 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 set of the action-labels that assigns 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 of the action-labels excluding $SL(M)$ from $NL(M)$.

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

$Path(M,s_0,t)$ 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 $\phi_1 U \phi_2$.

$$\chi(\phi_1,\phi_2,t,s_0) \triangleq \{s_0 \cdots s_t \in Path(M,s_0,t) \mid s_t \vDash \phi_2 \ \wedge \ \forall i \in 0 \leq i < t, \\ s_i \vDash \phi_1 \wedge s_i \nvDash \phi_2\}$$

. We prove Theorem 1 by proving Lemmas 3, 4, and 5. The proofs of Lemmas 3, 4, and 5 are in Appendix A.

Theorem 1. *Let M be a PRISM model, $\phi_1 \ U \ \phi_2$ be a 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. We have $[\![\phi_1 \ U \ \phi_2]\!]_M(s_0) = [\![\phi_1 \ U \ \phi_2]\!]_{SLI(M,\mathbf{I})}(s'_0)$.*

Proof. $[\![\phi_1 U \phi_2]\!]_M(s_0)$ is given in Box I.

Theorem 2. *Let M be a PRISM model where $t \in \mathbb{N}$, $\phi_1 \ U^{\leq t} \ \phi_2$ 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. 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. We show that Theorem 2 is wrong by the following counterexample. Taking Listing 1 as an example, the verification result of the original model against $[\![s\_u1 = 0 \ U^{\leq 1} \ s\_u1 = 1]\!]_M(s_0)$ is one third. However, the verification result of the model slice against $[\![s\_u1 = 0 \ U^{\leq 1} \ s\_u1 = 1]\!]_{SLI(M,I)}(s'_0)$ is one. □

6. Evaluation

We experimentally evaluated the effectiveness of collaborative MAPE-K (Section 6.1) alongside the correctness and efficiency of our slicing algorithm (Section 6.2).

6.1. Does the collaborative MAPE-K pattern provide an accurate SoS model that is effective in achieving and verifying SoS goals?

To answer this research question, we conducted experiments on two case studies: the BitTorrent SoS and the garbage collecting SoS.

• 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 $SLI : C \times 2^{\mathbf{V}} \rightarrow \text{Option } C$

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

Fig. 9. SLI transformation for commands in M.

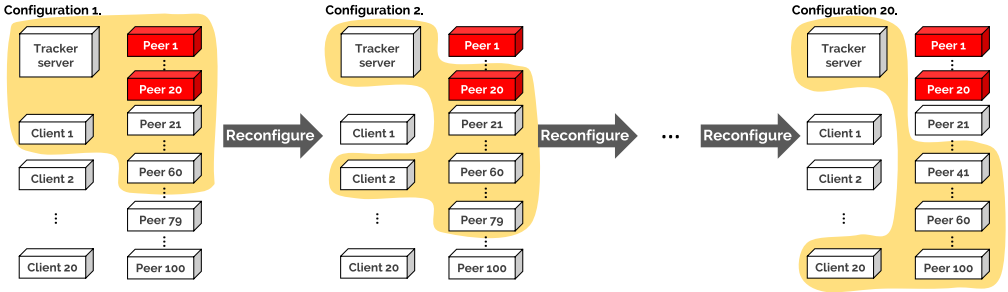


Fig. 10. A scenario of 20 re-configurations of a P2P file sharing SoS with 20 malware files.

6.1.1. BitTorrent SoS

Setup. We conducted an experiment on a BitTorrent file sharing SoS that included malware. We implemented 100 peers using Amazon EC2. Each peer used Transmission v2.77 [34] for uploading clean or malware files. The malware files were generated by BitErrant [27]. A peer having malware is considered a hacker. The size of a peer list configured by a tracker is 60, which matches the default setting of Transmission.

As shown in Fig. 10, an experimental scenario has 20 successive SoS reconfigurations according to file requests from clients. Peers 1 to 20 in Fig. 10 have malware. The first and second configurations include hackers, but the 20th configuration does not include a hacker. The experimental scenario is embodied in two sub-scenarios: the original BitTorrent scenario and the CVSoS-applied scenario. In the original scenario, the peer list was randomly configured by a tracker (*i.e.* internal knowledge was not shared). In the CVSoS-applied scenario, the peer list is configured based on a verification model with shared internal knowledge. For both scenarios, we increased the number of hackers from 10 to 40 at increments of 10. If hackers are included in the peer list, they will send malware fragments instead of a clean file. Each scenario was executed 40 times. To show the effectiveness of the CVSoS, we show the rates of having hackers and downloading malware in the peer list in both scenarios.

Results. As shown in Fig. 11(a), we compared rates of having hackers in the peer list in the original and CVSoS-applied scenarios. 10, 20, 30, and 40 in the legend of Fig. 11(a) represent the numbers of hackers. The x-axis of Fig. 11(a) represents the reconfiguration sequence of each scenario. Fig. 10 provides an example of the reconfiguration sequence. The y-axis is the probability of having hackers in the peer list.

In the original scenarios having 10 to 40 hackers, the rates of having hackers in the peer list are constant even after the SoS is reconfigured 20 times. This was caused by non-shared internal knowledge from peers to the tracker. However, in the CVSoS-applied scenarios, the rates of having hackers in the peer list decrease as the SoS reconfigures. As a result, the rates of the CVSoS-applied scenarios are lower than those of the originals. Hence, the probability of downloading malware from hackers decreases when CVSoS is applied.

In the case of the CVSoS-applied scenarios with 10 and 20 hackers, the rates gradually decrease with small decrements as the configuration repeats. In the case of the CVSoS-applied scenarios with 30 and 40 hackers, the rates were high in the early configurations, but they decreased dramatically in the end. This implies that the number of hackers incurs an increment of malware reporting from peers.

$$
\begin{aligned}
&[\![\phi_1 U \phi_2]\!]_M(s_0)\\
&= \begin{cases} 1 & (s_0 \vDash \phi_2)\\ 0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2)\\ \sum_{t=1}^{\infty} \sum_{\substack{s_1 \in \\ \{s\in S \mid s\vDash\phi_1 \wedge s\nvDash\phi_2\}}} \cdots \sum_{\substack{s_{t-1}\in \\ \{s\in S\mid s\vDash\phi_1\wedge s\nvDash\phi_2\}}} \sum_{\substack{s_t\in \\ \{s\in S\mid s\vDash\phi_2\}}} \left(\prod_{i=0}^{t-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}\\
&\quad \text{(by Lemma 3)}\\
&= \begin{cases} 1 & (s_0 \vDash \phi_2)\\ 0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2)\\ \sum_{t=1}^{\infty} \sum_{\substack{s'_1 \in \\ \{s'\in S' \mid s'\vDash\phi_1 \wedge s'\nvDash\phi_2\}}} \sum_{\substack{s_1\in \\ \{s\in S\mid s'_1=s|_{\mathbf{I}}\}}} \cdots \sum_{\substack{s'_{t-1}\in \\ \{s'\in S'\mid s'\vDash\phi_1\wedge s'\nvDash\phi_2\}}} \sum_{\substack{s_{t-1}\in \\ \{s\in S\mid s'_{t-1}=s|_{\mathbf{I}}\}}} \sum_{\substack{s'_t\in \\ \{s'\in S'\mid s'\vDash\phi_2\}}} \sum_{\substack{s_t\in \\ \{s\in S\mid s'_t=s|_{\mathbf{I}}\}}} \left(\prod_{i=0}^{t-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_{q'\in\{s'\in S'\mid s'\vDash\phi_1\wedge s'\nvDash\phi_2\}} \{s\in S \mid q'=s|_{\mathbf{I}}\} = \{s\in S\mid s\vDash\phi_1\wedge s\nvDash\phi_2\} \ \wedge \bigcup_{q'\in\{s'\in S'\mid s'\vDash\phi_2\}} \{s\in S\mid q'=s|_{\mathbf{I}}\} = \{s\in S\mid s\vDash\phi_2\} \right)\\
&= \begin{cases} 1 & (s_0 \vDash \phi_2)\\ 0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2)\\ \sum_{t=1}^{\infty} \sum_{\substack{s'_1 \in \\ \{s'\in S' \mid s'\vDash\phi_1 \wedge s'\nvDash\phi_2\}}} \cdots \sum_{\substack{s'_{t-1}\in \\ \{s'\in S'\mid s'\vDash\phi_1\wedge s'\nvDash\phi_2\}}} \sum_{\substack{s'_t\in \\ \{s'\in S'\mid s'\vDash\phi_2\}}} \left(\sum_{\substack{s_1\in \\ \{s\in S\mid s'_1=s|_{\mathbf{I}}\}}} \cdots \sum_{\substack{s_{t-1}\in \\ \{s\in S\mid s'_{t-1}=s|_{\mathbf{I}}\}}} \sum_{\substack{s_t\in \\ \{s\in S\mid s'_t=s|_{\mathbf{I}}\}}} \prod_{i=0}^{t-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}\\
&\quad \text{(by the distributive property)}\\
&= \begin{cases} 1 & (s'_0 \vDash \phi_2)\\ 0 & (s'_0 \nvDash \phi_1 \wedge s'_0 \nvDash \phi_2)\\ \sum_{t=1}^{\infty} \sum_{\substack{s'_1 \in \\ \{s'\in S' \mid s'\vDash\phi_1 \wedge s'\nvDash\phi_2\}}} \cdots \sum_{\substack{s'_{t-1}\in \\ \{s'\in S'\mid s'\vDash\phi_1\wedge s'\nvDash\phi_2\}}} \sum_{\substack{s'_t\in \\ \{s'\in S'\mid s'\vDash\phi_2\}}} \left(\prod_{i=0}^{t-1} \sum_{\substack{\mathtt{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}\\
&\quad (\because s_0|_{\mathbf{I}} = s'_0 \ \wedge \text{ Lemma 5})\\
&= [\![\phi_1 U \phi_2]\!]_{SLI(M,\mathbf{I})}(s'_0) \quad \text{(by Lemma 3)} \ \square
\end{aligned}
$$

Box I.

The increment of knowledge sharing leads to more effective hacker detection.

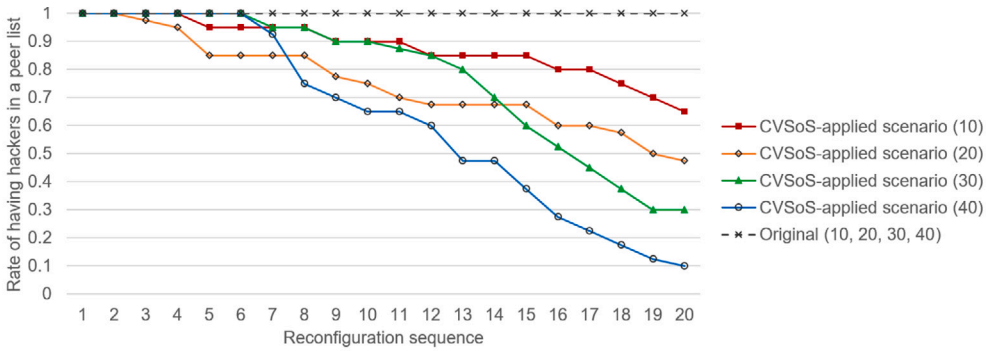
In Fig. 11(b), the graph shows rates of downloading malware. The rates from original scenarios were the same with those of the first configuration in the CVSoS-applied scenarios. The rates of downloading malware in the CVSoS-applied scenario (40) dropped 64% after 20 configurations. Hence, the SoS goal achievement rate increased 64% at most. The graphs show decreasing trends as the configuration repeats. The decreasing trends are natural because the rates of having hackers in a peer list decreases, as shown in Fig. 11(a). If a peer list is configured without hackers, malware will never be downloaded. The reason why the graphs do not decrease monotonically is that peers send file fragments. If a client receives malware-free fragments from hackers, the client can download a clean file, even if the hackers are included in a peer list.

Threat to validity. During the experiment, we controlled variables of execution time, numbers of peers and file size. However, we did not control the network latency or bandwidth of the peers located within or across regions. Thus, even if the experiment was conducted under the same conditions, it would differ from the results reported in this paper. To compensate for this, we conducted the same experiments multiple times.

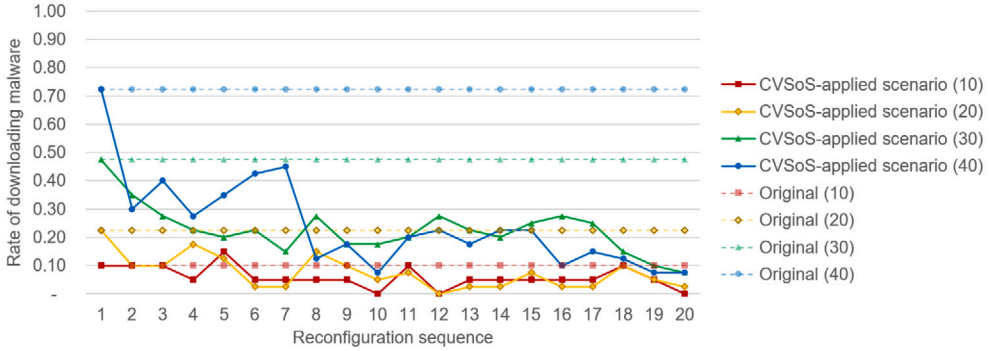
6.1.2. Garbage collection SoS in smart grid

Setup. To show that the proposed approach is applicable to various collaborative SoSs, we performed continuous verification on a garbage collection SoS in a smart grid which is the representative collaborative SoS [38]. To simulate the garbage collection SoS, we extended Park et al.'s dynamic simulator [39] based on Abusnaina et al.'s scenario [40]. The goal of the garbage collection SoS is to pick up and dispose of garbage in a short time. As shown in Fig. 12, 12 robots pick up garbage placed on the 35 by 35 map, and the robot's charging station is located in the same place as the trash bin. The robots on the map monitor and analyze the surrounding environment. Based on the analysis results, the robots make and execute a plan to go to the garbage location. The (E)-(M) connection between robots is implemented as routers.

Different levels of knowledge are shared with the (E)-(M) connections during the execution process. In the default scenario, there is no knowledge sharing between robots, and no robots are added. In the proposed pattern-applied scenarios, the robot responsible for conceptual management can request additional robots up to 20. To show that the SoS goal achievement varies according to the level of knowledge sharing, we have robots share some or all of knowledge about the total amount of garbage collected, visited locations, and garbage locations. We compared the maximum amount of garbage collected per a robot



(a) Rates of having hackers in a peer list



(b) Rates of downloading malware

Fig. 11. Experimental results as the SoS is reconfigured 20 times.

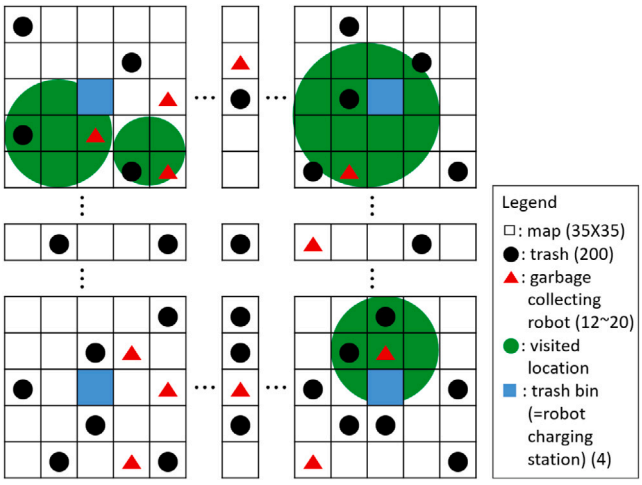


Fig. 12. Continuous verification setup for a garbage collection SoS in a smart grid.

to determine whether sharing knowledge causes the CS's individual goals to conflict with the SoS goal and causes the CS's performance degradation.

Results. Fig. 13(a) shows the distribution of time taken for robots to achieve the SoS goal of completely clearing the map. In the default scenario where CVSoS is not applied, it takes 2670.125 ticks on average. In the CVSoS-applied scenarios, the average simulation time is reduced to less than 2000 ticks, and the number of robots increases to 20 according to the verification result. The simulation time to achieve the goal varies according to the knowledge sharing level. Experimental results show that the higher the level of knowledge sharing, the faster the SoS goal can be achieved. The total amount of trash is used for requesting additional robots. Sharing the visited location ensures that the robots do not clean the same place, and the visited circles in Fig. 12 do not overlap. By sharing the location of the garbage, it is possible to achieve the SoS goal in a shorter time.

Fig. 13(b) shows that knowledge sharing can undermine the goal of maximizing garbage collection per robot. Scenarios that share less knowledge have a larger maximum garbage collection per robot. When the SoS goal and the individual CS goal collide, CSs can determine the level of knowledge sharing by verifying that the SoS goal does not significantly impair individual interests.

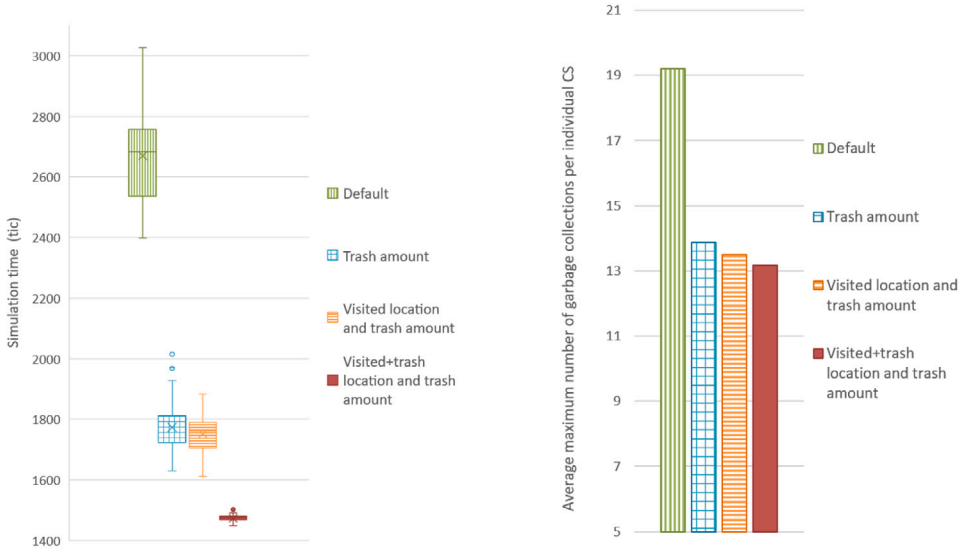
Threat to validity. In an experimental environment different from the setting described, the effectiveness of the proposed approach cannot be guaranteed. However, to demonstrate the effectiveness of the proposed method as best as possible, we applied it to multiple systems and revealed the experimental results of various cases.

6.2. *Is the proposed slicing algorithm correct and efficient?*

This research question seeks to find if the slicing algorithm gives the same verification results and if they incur lower verification costs. We answered this question by conducting experiments as follows.

Setup. We evaluated the scalability of our slicing algorithm with an experiment using the modified Kwiatkowska's PRISM DTMC model benchmark [41] and verification properties for slicing purposes; the mass casualty incident (MCI) DTMC model [39] and verification property were used in our experiments; and the P2P file sharing DTMC model had 20 to 500 peers. The target DTMC models and the verification properties are summarized in Table 1.

To evaluate our slicing algorithm's correctness, we performed PMC and SMC on the original models and their model slices. We then compared their verification results. The PMC verification result serves



(a) Simulation time to achieve an SoS goal

(b) Average maximum number of garbage collection per individual CS

Fig. 13. The experimental results of a garbage collection SoS.

Table 1
Subject DTMC models and their properties.

Subject model	Property
BRP [41]	$[\![\mathtt{tt}\ U\ s = 5]\!]_M(s)$
Leader_sync [41]	$[\![\mathtt{tt}\ U\ \text{“}elected4\text{”}]\!]_M(s)$
Two dice [42]	$[\![\mathtt{tt}\ U\ s = 7 \& d = 6]\!]_M(s)$
MCI [39]	$[\![dead < 1\ U\ rescued \geq MAX]\!]_M(s)$
P2P SoS (20)	$[\![\mathtt{tt}\ U\ data_1 + \cdots + data_5 \geq MAX]\!]_M(s)$
P2P SoS (50)	$[\![\mathtt{tt}\ U\ data_1 + \cdots + data_{20} \geq MAX]\!]_M(s)$
P2P SoS (100~500)	$[\![\mathtt{tt}\ U\ data_1 + \cdots + data_{60} \geq MAX]\!]_M(s)$

as the oracle, but it may not be available for large models, owing to state explosions. Hence, we applied a paired t-test to statistically compare the original and sliced SMC results. The null hypothesis (H_0) states that a verification result of an original model and that of the model's slice are different. For $\alpha = 0.05$, the critical value of t is 1.960. If t is larger than 1.960, we reject H_0 and state that there is no difference in SMC results. We set the sampling and path parameters of the SMC as follows: simulation method = confidence interval (CI), confidence level = 0.05, maximum path length = 10,000, and the number of samples is generated until the two CIs of the distributions are identical.

To evaluate our slicing algorithm's efficiency and scalability, we compared (1) the time taken to verify the original models; and (2) the sum of the time taken to verify the model slices, and the time taken to slice the models.

Results—Correctness. The verification results in Table 2 demonstrate the correctness of our approach by comparing the slice results to the original results. The first two columns in the verification results denote the PMC verification results of the original models and their model slices. The PMC results for the original models and model slices are identical. However, owing to the state-explosion problem, the PMC cannot be applied to all models.

The third and fourth columns in the verification results present the SMC verification results of the original models and model slices. Because the SMC verification results are approximate predictions and not accurate predictions like PMC, we applied a paired t-test to statistically compare the original and slice SMC results. The last column in Table 2 denotes the t-value. Because every t-value is larger than 1.960, we reject the null hypothesis, which means there is no difference between the verification results of original models and those of the model slices.

Results—Efficiency. Table 2 also presents the verification time for the original and model slices. We found that the model slices generally required less time than the original models. In the P2P SoS models, as the number of peers increased, the time spent on the original SMC increased exponentially. It took approximately 45 s to 8 min to verify the SoS goal while conducting SMC on the original P2P SoS models with more than 100 peers. However, conducting SMC on model slices took approximately 25 s, and the results did not show much difference even when the number of peers increased. This is because a list of 60 selected peers, which is the default setting size, is retained in the model slice when using the proposed slicing approach. Consequently, if the number of peers is increased, the verification time of the original model is exponentially increased, whereas the model slice shows almost constant verification time.

7. Related works

7.1. Continuous verification through MAPE-K

Continuous verification techniques have been proposed to ensure that requirements can be met despite environmental or system changes. The concept of combining MAPE-K to continuous verification of a single system was proposed by Ghezzi et al. [5]. Yang et al. [12] built an interactive state machine of applications and verified them considering environmental uncertainty. However, modeling and verification of interactions with other systems were not covered. Moreno et al. [8] proposed a proactive self-adaption approach with a probabilistic model checking to predict the future operations of a system and to establish a strategy. Like Yang's research, they regarded other systems as environments.

The aforementioned studies cannot be applied to SoS, because they assumed that the target system was a single white boxed system, whereas the SoS consists of multiple CSs, requiring multiple MAPE-K loops. Our collaborative MAPE-K pattern supports multiple system adaptations and can model gray-boxed CSs.

7.2. Decentralized MAPE-K

Weyns et al. [11] proposed decentralized MAPE-K patterns to meet the need for practical techniques of modeling distributed control systems for hierarchical control, main/subordinates, regional planning,

Table 2
Verification result and time comparison of the subject models and their model slices.

Subject model	Verification result (%)					Verification time	
	Original PMC	Slice PMC	Original SMC	Slice SMC	t-value > 1.960	Original (s)	Slice (s)
BRP	2.04e−4	2.04e−4	2.8e−4 (± 1.4e−4)	2.4e−4 (± 1.3e−4)	12,649.110	333.81	330.91
Leader_sync	100.00	100.00	100.00 (± 0)	100.00 (± 0)	∞	0.97	0.04
Two dice	16.67	16.67	15.6 (± 2.96)	15.1 (± 2.92)	3.591	0.053	0.04
MCI	–	–	74.3 (± 3.56)	71.5 (± 3.67)	17.692	25	15
P2P SoS (20)	–	–	1.4 (± 0.957)	1.9 (± 1.11)	11.194	0.647	0.281
P2P SoS (50)	–	–	40.3 (± 4)	44.9 (± 4.05)	26.5541	5.08	2.14
P2P SoS (100)	–	–	47.1 (± 4.07)	48.2 (± 4.07)	5.145	45.85	25.51
P2P SoS (200)	–	–	47.5 (± 4.06)	47.7 (± 4.07)	2.479	76.88	25.59
P2P SoS (300)	–	–	46.3 (± 4.06)	49.6 (± 4.07)	20.979	216.48	25.79
P2P SoS (400)	–	–	48.0 (± 4.07)	49.2 (± 4.07)	7.644	235.36	25.40
P2P SoS (500)	–	–	45.6 (± 4.06)	49.1 (± 4.07)	19.051	504	25.874

coordinated control, and information sharing patterns. The hierarchical control, main/subordinates, and regional planning patterns were the vertical structures. They had upper and lower layers between systems. The purpose of the vertical structure is to provide stability and consistency of lower layered systems via a single upper layered system. Examples of these patterns are found in the RESERVOIR project [43] and the IBM architectural blueprint [4]. The vertical structured patterns, however, are inapplicable to SoS, owing to the control of one powerful upper layered system. Hence, the upper layer restricts O/M independence by preventing systems at the lower layer from analyzing and planning themselves.

The coordinated control and information sharing patterns do not apply layers to systems, but the patterns are instead based on synchronization and information sharing between the systems. The coordinated control pattern targets systems wherein the cost of collecting information is too high, or the scale of the system is too large to process all information in one system [44]. All MAPE components synchronize their operations with the corresponding components of other systems. Hence, the SoS is dynamically reconfigured according to O/M independence, but systems in which CSs must be synchronized do not allow dynamic reconfiguration. For example, if an artificial satellite freely deviates from orbit, it can lead to a collision. Therefore the existing decentralized MAPE-K patterns are not applicable to model an SoS.

7.3. Verification cost of SoS

To detour the state explosion problem, many researchers have conducted research on SoS verification with SMC [19]. Nevertheless, the sample size and verification cost grows exponentially if the answer for SMC is required to be highly accurate [21,22].

Slicing has been frequently used for large-scale system analysis among various purposes, such as for software maintenance [45], comprehension [46], and re-engineering [47].

Gold et al. [48] proposed a generalized observational slicing technique for tree-represented modeling languages to analyze large systems. The generalized observational slicing technique does not require a complex model dependency analysis; thus, there is no burden to slice large models. However, as pointed out in the study of Song et al. [49], observational slicing may produce inaccurate model slices depending on model executions if the target model includes probability factors. Therefore, generalized observational slicing is difficult to achieve SoS accuracy with its probability factors.

Our slicing algorithm is closely related to cone of influence (COI) reduction [50] used in model abstraction. COI reduction calculates a minimal set of variables dependent on a verification property. A dependency graph is used to find dependent variables, and new dependencies can be added according to the characteristics of the language. For example, an observe dependency was proposed by Hur et al. [51]. Accurate reduction for probabilistic programs, such as Church [52] and Infer.NET [53], is possible only when the observe dependency is considered. We proposed synchronize dependency for PRISM models in addition to control and data dependence.

Although Clark et al. [54] proved that the verification result of COI reduced model based on ACTL was the same as that of original model, the correctness of COI reduction based on PCTL properties has not been proved. However, many researchers have applied the COI reduction based on PCTL properties. For example, in Ammar et al. [55], the authors reduced and verified time-triggered Ethernet protocol models using PRISM. Hamad et al. [56] performed COI reduction on PRISM models based on PCTL properties and stated that there was differences among results. We proved that by performing COI reduction on PRISM models based on PCTL until properties, it is possible to obtain the minimal influencer set, **I**, and obtain the same verification results as the original models.

Researchers have abstracted Markov decision process (MDP) models based on PCTL verification properties. However, the verification result is given as a range. For example, by applying the studies of D'argenio et al. [57] and Parker et al. [58], the minimum and maximum limits of the probability of satisfying the PCTL properties can be obtained. When we perform model abstraction on bounded until properties, the above studies can supplement ours to obtain a range of probabilities. However, because the above abstracting MDP techniques reduce the error by repeating verifications based on the original model verification result, an increase in verification efficiency cannot be expected.

8. Future work

In future works, we should account for the different classes of CSs to better understand their potential conflicting interests and their effects on SoS verification. The proposed collaborative MAPE-K pattern was designed for collaborative-type SoSs, which do not bind CSs into a hierarchy. There are other types of SoSs to consider such as acknowledged SoS which has hierarchy among CSs. In acknowledged SoSs, managing CSs guides other managed CSs to accept their commands. However, CSs do not easily accept such commands because the managed CSs may have different interests owing to its O/M independence. Hence, a new MAPE-K pattern may be necessary to manage conflicts among CSs.

We experimentally showed that the verification time of model slice is reduced when the proposed slicing is applied. To better understand the implications of this finding, future studies should address which factors should be used in the slicing algorithm (*e.g.*, the size of influencers and the number of commands) to reduce verification costs more effectively. Furthermore, we plan to generalize our algorithm so that it supports other types of verification properties, such as those with $U^{<t}$ with time bounds.

9. Conclusion

This study aimed to adapt CSs in response to the dynamically reconfigured SoSs using the proposed CVSoS to achieve the SoS goal. Owing to the O/M independence of the CSs, it is difficult to apply extant continuous verification approaches [5,8,12] to the CSs as-is. CVSoS comprises a collaborative MAPE-K pattern with model slicing. The collaborative MAPE-K pattern reflects changes of SoSs into an SoS model to increase its accuracy. With model slicing, we provided a new synchronization dependence. When combined with conventional dependence, the high verification cost of SoS models is remedied with the backward slicing of the dependency graph with respect to the SoS goal.

CVSoS described above can be externalized in two perspectives: (1) verification of evolving SoS and (2) extension of the proposed model slicing.

First, when the dynamicity of SoS is maximized, SoS evolution occurs in which goals and types change. To deal with the verification problem in a larger context, the work of this paper is a prerequisite. Hence, CVSoS should be further developed in a way that considers changes in SoS management, goals, and types for its long-term evolution. We believe that the suggestions presented as future works in previous section will be helpful with this objective.

Second, our proposed model slicing technique can be applied to any system model which have goal properties to verify. Notably, it is only applicable for the until verification property. The proof of Theorem 2 related to bounded until properties limits the generalizability of the slicing approach, and the counterexample of Theorem 2 will provide new insight into suggesting new dependence for the bounded until properties.

CRediT authorship contribution statement

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

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.

Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2022-2020-0-01795), (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), and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education(NRF-2019R1I1A1A01062946).

Appendix A. Proofs of Lemmas 3, 4, and 5

Lemma 3. $\forall s_0 \in S$,

$$[\![\phi_1 U \phi_2]\!]_M(s_0) = \begin{cases} 1 & (s_0 \vDash \phi_2) \\ 0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2) \\ \sum_{t=1}^{\infty} \sum_{s_1 \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \cdots \sum_{s_{t-1} \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \sum_{s_t \in \{s \in S \mid s \vDash \phi_2\}} \left(\prod_{i=0}^{t-1} \sum_{\mathtt{ns} \in next(M,s_i) \cap SL(M)} \frac{[\![lookup(M,\mathtt{ns})]\!](s_i, s_{i+1})}{|next(M,s_i) \cap SL(M)|} \right) & (otherwise) \end{cases}$$

Proof. $[\![\phi_1 U \phi_2]\!]_M(s_0)$ is given in Box II

Lemma 4. *Let* $T_1 = \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}$, $T_2 = \{s \in S \mid s \vDash \phi_2\}$, $T \in \{T_1, T_2\}$, *and* $s_0 \in T_1$.

$$\begin{aligned} &\sum_{t=1}^{\infty} \sum_{s_1 \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \cdots \sum_{s_{t-1} \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \sum_{s_t \in T} \\ &\prod_{i=0}^{t-2} \sum_{\mathtt{na} \in next(M,s_i) \cap AL(M)} \frac{[\![lookup(M,\mathtt{na})]\!](s_i, s_{i+1})}{|next(M,s_i)|} \\ &\quad \times \sum_{\mathtt{ns} \in next(M,s_{t-1}) \cap SL(M)} \frac{[\![lookup(M,\mathtt{ns})]\!](s_{t-1}, s_t)}{|next(M,s_{t-1})|} \\ &= \sum_{t' \in T} \sum_{\mathtt{ns} \in next(M,s_0) \cap SL(M)} \frac{[\![lookup(M,\mathtt{ns})]\!](s_0, t')}{|next(M,s_0) \cap SL(M)|} . \end{aligned}$$

Proof. We prove by continuously transforming LHS to RHS as follows:

(1) Transform LHS into matrices as it is.
We define matrices as follows:

- Let v be the $|S|$-sized row vector where s_0 has a value of 1 and the other states in P have a value of 0.
- Let w be the $|S|$-sized column vector where all $t' \in T$ have a value of 1 and the other states in S have a value of 0.
- Let $s, s' \in S$ and $A = (a_{s,s'}) \in \mathbb{R}^{|S| \times |S|}$ be the transition probability matrix by action-labels in $AL(M)$ as follows:

$$a_{s,s'} \triangleq \sum_{\mathtt{na} \in next(M,s) \cap AL(M)} \frac{[\![lookup(M,\mathtt{na})]\!](s, s')}{|next(M,s)|} .$$

- Let $s, s' \in S$ and $B = (b_{s,s'}) \in \mathbb{R}^{|S| \times |S|}$ be the transition probability matrix by action-labels in $SL(M)$ as follows:

$$b_{s,s'} \triangleq \sum_{\mathtt{ns} \in next(M,s) \cap SL(M)} \frac{[\![lookup(M,\mathtt{ns})]\!](s, s')}{|next(M,s)|} .$$

$$
\begin{aligned}
&[\![\phi_1 U \phi_2]\!]_M(s_0)\\
&=\sum_{t=0}^{\infty}\sum_{\substack{s_0\cdots s_t\in\\ \chi(\phi_1,\phi_2,t,s_0)}}\prod_{i=0}^{t-1}[\![M]\!](s_i,s_{i+1})\\
&\quad\text{(by definition of until property)}\\
&=\begin{cases}1 & (s_0\vDash\phi_2)\\ 0 & (s_0\nvDash\phi_1\wedge s_0\nvDash\phi_2)\\ \displaystyle\sum_{t=1}^{\infty}\sum_{\substack{s_1\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\cdots\sum_{\substack{s_{t-1}\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\sum_{\substack{s_t\in\\ \{s\in S|s\vDash\phi_2\}}}\prod_{i=0}^{t-1}[\![M]\!](s_i,s_{i+1}) & (otherwise)\end{cases}\\
&\quad\text{(by definition of }\chi)\\
&=\begin{cases}1 & (s_0\vDash\phi_2)\\ 0 & (s_0\nvDash\phi_1\wedge s_0\nvDash\phi_2)\\ \displaystyle\sum_{t=1}^{\infty}\sum_{\substack{s_1\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\cdots\sum_{\substack{s_{t-1}\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\sum_{\substack{s_t\in\\ \{s\in S|s\vDash\phi_2\}}}\prod_{i=0}^{t-1}\frac{\sum_{\mathtt{n}\in next(M,s_i)}[\![lookup(M,\mathtt{n})]\!](s_i,s_{i+1})}{|next(M,s_i)|} & (otherwise)\end{cases}\\
&\quad\text{(by definition of }M)\\
&=\begin{cases}1 & (s_0\vDash\phi_2)\\ 0 & (s_0\nvDash\phi_1\wedge s_0\nvDash\phi_2)\\ \displaystyle\sum_{t=1}^{\infty}\sum_{\substack{s_1\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\cdots\sum_{\substack{s_{t-1}\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\sum_{\substack{s_t\in\\ \{s\in S|s\vDash\phi_2\}}}\\ \displaystyle\prod_{i=0}^{t-1}\left(\frac{\sum_{\mathtt{na}\in next(M,s_i)\cap AL(M)}[\![lookup(M,\mathtt{na})]\!](s_i,s_{i+1})+\sum_{\mathtt{ns}\in next(M,s_i)\cap SL(M)}[\![lookup(M,\mathtt{ns})]\!](s_i,s_{i+1})}{|next(M,s_i)|}\right) & (otherwise)\end{cases}\\
&\quad(\because SL(M)\cap AL(M)=\emptyset)\\
&=\begin{cases}1 & (s_0\vDash\phi_2)\\ 0 & (s_0\nvDash\phi_1\wedge s_0\nvDash\phi_2)\\ \displaystyle\sum_{t=1}^{\infty}\sum_{\substack{s_1\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\cdots\sum_{\substack{s_{t-1}\in\\ \{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\}}}\sum_{\substack{s_t\in\\ \{s\in S|s\vDash\phi_2\}}}\\ \left(\displaystyle\prod_{i=0}^{t-2}\left(\frac{\sum_{\substack{\mathtt{na}\in\\ next(M,s_i)\cap AL(M)}}[\![lookup(M,\mathtt{na})]\!](s_i,s_{i+1})}{|next(M,s_i)|}+\frac{\sum_{\substack{\mathtt{ns}\in\\ next(M,s_i)\cap SL(M)}}[\![lookup(M,\mathtt{ns})]\!](s_i,s_{i+1})}{|next(M,s_i)|}\right)\right)\\ \times\left(0+\dfrac{\sum_{\substack{\mathtt{ns}\in\\ next(M,s_{t-1})\cap SL(M)}}[\![lookup(M,\mathtt{ns})]\!](s_{t-1},s_t)}{|next(M,s_{t-1})|}\right) & (otherwise)\end{cases}\\
&\quad(\because\forall\mathtt{na}\in AL(M),\forall q\in\{s\in S|s\vDash\phi_1\wedge s\nvDash\phi_2\},\ \forall q'\in\{s\in S|s\vDash\phi_2\},\ [\![lookup(M,\mathtt{na})]\!](q,q')=0\ .)
\end{aligned}
$$

Box II.

From the above definitions, LHS is immediately transformed as follows:

$$(LHS)=\lim_{t\to\infty}\upsilon\cdot(I+A+\cdots+A^{t-1})\cdot B\cdot w\ .$$

(2) Trim matrices based on states that cannot be reached by action-labels in $AL(M)$.
We define more matrices as follows:

- Let $P=\{p\in S\mid p|_{\mathbf{I}}=s_0|_{\mathbf{I}}\}$ and υ' be the $|P|$-sized row vector where s_0 has a value of 1 and the other states in P have a value of 0.
- Let $p,p'\in P$ and $A'=(a'_{p,p'})\in\mathbb{R}^{|P|\times|P|}$ be the transition probability matrix by action-labels in $AL(M)$ as follows:

$$a'_{p,p'}\ \triangleq\sum_{\substack{\mathtt{na}\in\\ next(M,p)\cap AL(M)}}\frac{[\![lookup(M,\mathtt{na})]\!](p,p')}{|next(M,p)|}\ .$$

- Let $p\in P$, $s\in S$, and $B'=(b'_{p,s})\in\mathbb{R}^{|P|\times|S|}$ be the transition probability matrix by action-labels in $SL(M)$ as follows:

$$b'_{p,s}\ \triangleq\sum_{\substack{\mathtt{ns}\in\\ next(M,p)\cap SL(M)}}\frac{[\![lookup(M,\mathtt{ns})]\!](p,s)}{|next(M,p)|}\ .$$

The 0 vectors of the υ, A, and B matrices are trimmed and the matrices are transformed as follows:

$$
\begin{aligned}
&\lim_{t\to\infty}\upsilon\cdot(I+A+\cdots+A^{t-1})\cdot B\cdot w\\
&=\lim_{t\to\infty}\upsilon'\cdot(I+A'+\cdots+A'^{t-1})\cdot B'\cdot w\\
&\quad(\because\forall\mathtt{na}\in AL(M),\ \forall s\in S,\ \forall s'\in\{q\in S\mid q\neq s|_{\mathbf{I}}\},\\
&\qquad[\![lookup(M,\mathtt{na})]\!](s,s')=0)\ .
\end{aligned}
$$

$$
= \begin{cases}
1 & (s_0 \vDash \phi_2) \\
0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2) \\
\sum_{j=1}^{\infty} \sum_{t_1=1}^{\infty} \sum_{t_2=t_1+1}^{\infty} \cdots \sum_{t_{j-1}=t_{j-2}+1}^{\infty} \sum_{t_j=t_{j-1}+1}^{\infty} \\
\quad \sum_{s_1 \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \cdots \sum_{s_{t_1} \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \\
\quad \sum_{s_{t_1+1} \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \cdots \sum_{s_{t_2} \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \\
\quad \vdots \\
\quad \sum_{s_{t_{j-1}+1} \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \cdots \sum_{s_{t_j-1} \in \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}} \sum_{s_{t_j} \in \{s \in S \mid s \vDash \phi_2\}} \\
\left(\left(\prod_{i=0}^{t_1-2} \frac{\sum_{\mathtt{na} \in next(M,s_i) \cap AL(M)} [\![lookup(M, \mathtt{na})]\!](s_i, s_{i+1})}{|next(M, s_i)|} \right) \times \frac{\sum_{\mathtt{ns} \in next(M, s_{t_1-1}) \cap SL(M)} [\![lookup(M, \mathtt{ns})]\!](s_{t_1-1}, s_{t_1})}{|next(M, s_{t_1-1})|} \right) \\
\quad \times \cdots \times \\
\left(\left(\prod_{i=t_{j-1}}^{t_j-2} \frac{\sum_{\mathtt{na} \in next(M,s_i) \cap AL(M)} [\![lookup(M, \mathtt{na})]\!](s_i, s_{i+1})}{|next(M, s_i)|} \right) \times \frac{\sum_{\mathtt{ns} \in next(M, s_{t_j-1}) \cap SL(M)} [\![lookup(M, \mathtt{ns})]\!](s_{t_j-1}, s_{t_j})}{|next(M, s_{t_j-1})|} \right) & (otherwise)
\end{cases}
$$

(by double counting: the set of paths sorted by the length t and the set of paths sorted by the number j of transitions of $\mathtt{ns} \in$ SL(M) are the same. $\forall t \in \mathbb{N},\ \forall \pi \in Path(M, s_0, t),\ j = \left| \left\{ i \in \mathbb{N} \mid \pi_i = s_i \wedge \sum_{\substack{\mathtt{ns} \in \\ next(M, s_{i-1}) \\ \cap SL(M)}} [\![lookup(M, \mathtt{ns})]\!](s_{i-1}, s_i) > 0 \right\} \right|$.

$\forall k \in 1 \le k \le j,\ \forall t_k \in \mathbb{N}$, s_{t_k} is the state transitioned by $\mathtt{ns} \in$ SL(M).)

Box II. (*continued*).

(3) Simplify infinite matrix additions into an inverse matrix.

$$
\begin{aligned}
&\lim_{t \to \infty} v' \cdot (I + A' + \cdots + A'^{t-1}) \cdot B' \cdot w \\
&= v' \cdot (I - A')^{-1} \cdot B' \cdot w \\
&\quad \text{(by Gershgorin circle theorem [59] and } \lim_{t \to \infty} A^t = 0) \ .
\end{aligned}
$$

(4) Collapse B' and w by restricting states based on **I**.
We define more matrices as follows:

- Let w' be the $|S'|$-sized column vector where all states in $\{s' \in S' \mid \forall t' \in T, s' = t'|_{\mathbf{I}}\}$ have a value of 1 and the other states in S' have a value of 0.
- Let $p \in P$, $s' \in S'$, $S_{s'} = \{s \in S \mid s' = s|_{\mathbf{I}}\}$, and $B'' = (b''_{p,s'}) \in \mathbb{R}^{|P| \times |S'|}$ be the transition probability matrix by action-labels in $SL(M)$ as follows:

$$
b''_{p,s'} \triangleq \sum_{\substack{\mathtt{ns} \in \\ next(M,p) \cap SL(M)}} \sum_{s \in S_{s'}} \frac{[\![lookup(M, \mathtt{ns})]\!](p, s)}{|next(M, p)|} \ .
$$

We transform B' into B'' which is adding values of restricted states in $S_{s'}$ first and product the matrices as follows:

$$
\begin{aligned}
&v' \cdot (I - A')^{-1} \cdot B' \cdot w \\
&= v' \cdot (I - A')^{-1} \cdot B'' \cdot w' \\
&\left(\because \bigcup_{\substack{q' \in \\ \{s' \in S' \mid s' \nvDash \phi_1 \wedge s' \nvDash \phi_2\}}} \{s \in S | q' = s|_{\mathbf{I}}\} = T_1 \ \wedge \bigcup_{\substack{q' \in \\ \{s' \in S' \mid s' \nvDash \phi_2\}}} \{s \in S | q' = s|_{\mathbf{I}}\} = T_2 \right) .
\end{aligned}
$$

(5) Simplify the matrix product of $(I - A')^{-1}$ and B''.
We define more matrices as follows:

- Let $p \in P$, $s' \in S'$ and $C' = (c'_{p,s'}) \in \mathbb{R}^{|P| \times |S'|}$ be the transition probability matrix excluding action-labels in $AL(M)$ from B as follows:

$$
c'_{p,s'} \triangleq \sum_{\substack{\mathtt{ns} \in \\ next(M,p) \cap SL(M)}} \sum_{s \in S_{s'}} \frac{[\![lookup(M, \mathtt{ns})]\!](p, s)}{|next(M, p) \cap SL(M)|} \ .
$$

- Let $i \in 1 \le i \le |P|$, $p_i \in P$, $s' \in S'$, $k_i = \frac{1}{|next(M,p_i)|}$, $k'_i = \frac{1}{|next(M,p_i) \cap SL(M)|}$, $D = (d)_{p_i,s'} \in \mathbb{R}^{|P| \times |S'|}$ be the matrix as follows:

$$
d_{p_i,s'} \triangleq \sum_{\mathtt{ns} \in next(M,p_i) \cap SL(M)} \sum_{s \in S_{s'}} [\![lookup(M, \mathtt{ns})]\!](p_i, s) \ .
$$

- Let O be the stochastic matrix as follows:

$$
O \triangleq \begin{pmatrix} A' & B'' \\ 0 & I \end{pmatrix} .
$$

We prove $v' \cdot (I - A')^{-1} \cdot B'' \cdot w' = v' \cdot C' \cdot w'$ as follows.

Proof.

$$
\begin{aligned}
&((I - A')^{-1} \cdot B'')_{p_i,*} \\
&= \left((I - A')^{-1} \cdot \begin{pmatrix} k_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & k_{|P|} \end{pmatrix} \cdot D \right)_{p_i,*} \\
&= \left((I - A')^{-1} \cdot \begin{pmatrix} k_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & k_{|P|} \end{pmatrix} \cdot \begin{pmatrix} k'_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & k'_{|P|} \end{pmatrix}^{-1} \cdot \begin{pmatrix} k'_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & k'_{|P|} \end{pmatrix} \cdot D \right)_{p_i,*}
\end{aligned}
$$

$$
= \begin{cases} 1 & (s_0 \vDash \phi_2) \\ 0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2) \\ \displaystyle\sum_{j=1}^{\infty} \left(\sum_{t_1=1}^{\infty} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}}} \cdots \sum_{\substack{s_{t_1} \in \\ \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}}} \right. \\ \quad \left(\prod_{i=0}^{t_1-2} \sum_{\substack{\mathrm{na} \in \\ next(M,s_i) \cap AL(M)}} \frac{[\![lookup(M,\mathrm{na})]\!](s_i, s_{i+1})}{|next(M,s_i)|} \right) \times \sum_{\substack{\mathrm{ns} \in \\ next(M,s_{t_1-1}) \cap SL(M)}} \frac{[\![lookup(M,\mathrm{ns})]\!](s_{t_1-1}, s_{t_1})}{|next(M,s_{t_1-1})|} \\ \quad \left. \times \cdots \times \right. \\ \quad \left(\displaystyle\sum_{t_j=t_{j-1}+1}^{\infty} \sum_{\substack{s_{t_{j-1}+1} \in \\ \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}}} \cdots \sum_{\substack{s_{t_j-1} \in \\ \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}}} \sum_{\substack{s_{t_j} \in \\ \{s \in S \mid s \vDash \phi_2\}}} \right. \\ \quad \left. \left(\displaystyle\prod_{i=t_{j-1}}^{t_j-2} \sum_{\substack{\mathrm{na} \in \\ next(M,s_i) \cap AL(M)}} \frac{[\![lookup(M,\mathrm{na})]\!](s_i, s_{i+1})}{|next(M,s_i)|} \right) \times \sum_{\substack{\mathrm{ns} \in \\ next(M,s_{t_j-1}) \cap SL(M)}} \frac{[\![lookup(M,\mathrm{ns})]\!](s_{t_j-1}, s_{t_j})}{|next(M,s_{t_j-1})|} \right) & (otherwise) \end{cases}
$$

(by the distributive property)

$$
= \begin{cases} 1 & (s_0 \vDash \phi_2) \\ 0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2) \\ \displaystyle\sum_{t=1}^{\infty} \left(\sum_{\substack{s_1 \in \\ \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}}} \sum_{\substack{\mathrm{ns} \in \\ next(M,s_0) \cap SL(M)}} \frac{[\![lookup(M,\mathrm{ns})]\!](s_0, s_1)}{|next(M,s_0) \cap SL(M)|} \right) \\ \quad \times \cdots \times \\ \quad \left(\displaystyle\sum_{\substack{s_t \in \\ \{s \in S \mid s \vDash \phi_2\}}} \sum_{\substack{\mathrm{ns} \in \\ next(M,s_{t-1}) \cap SL(M)}} \frac{[\![lookup(M,\mathrm{ns})]\!](s_{t-1}, s_t)}{|next(M,s_{t-1}) \cap SL(M)|} \right) & (otherwise) \end{cases}
$$

(by Lemma 4)

$$
= \begin{cases} 1 & (s_0 \vDash \phi_2) \\ 0 & (s_0 \nvDash \phi_1 \wedge s_0 \nvDash \phi_2) \\ \displaystyle\sum_{t=1}^{\infty} \sum_{\substack{s_1 \in \\ \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}}} \cdots \sum_{\substack{s_{t-1} \in \\ \{s \in S \mid s \vDash \phi_1 \wedge s \nvDash \phi_2\}}} \sum_{\substack{s_t \in \\ \{s \in S \mid s \vDash \phi_2\}}} \left(\prod_{i=0}^{t-1} \sum_{\substack{\mathrm{ns} \in \\ next(M,s_i) \cap SL(M)}} \frac{[\![lookup(M,\mathrm{ns})]\!](s_i, s_{i+1})}{|next(M,s_i) \cap SL(M)|} \right) & (otherwise) \end{cases}
$$

(by the distributive property) □

Box II. (*continued*).

$$
= \left((I - A')^{-1} \cdot \begin{pmatrix} k_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & k_{|P|} \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{k_1'} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{k_{|P|}'} \end{pmatrix} \cdot \begin{pmatrix} k_1' & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & k_{|P|}' \end{pmatrix} \cdot D \right)_{p_i,*}
$$

$$
= \left((I - A')^{-1} \cdot \begin{pmatrix} \frac{k_1}{k_1'} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{k_{|P|}}{k_{|P|}'} \end{pmatrix} \cdot \begin{pmatrix} k_1' & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & k_{|P|}' \end{pmatrix} \cdot D \right)_{p_i,*}
$$

$$
= \left((I - A')^{-1} \cdot \begin{pmatrix} \frac{k_1}{k_1'} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{k_{|P|}}{k_{|P|}'} \end{pmatrix} \cdot C' \right)_{p_i,*}
$$

$$
= ((I - A')^{-1})_{p_i,p_i} \cdot \frac{k_i}{k_i'} \cdot (C')_{p_i,*}
$$

$$
= (C')_{p_i,*}
$$

$$
\left(\because \forall p \in P, \sum_{s' \in S'} c'_{p,s'} = \sum_{s' \in S'} ((I - A)^{-1} \cdot B')_{p,s'} = 1 \text{ by definition of } C' \text{ and infinite product of the stochastic matrix } O, \right.
$$

$$
\left. \lim_{t \to \infty} O^t = \begin{pmatrix} 0 & (I - A')^{-1} B'' \\ 0 & I \end{pmatrix} . \right) \quad \square
$$

(6) Expand C' and w' by unrestricting states from $\mathbf{I}$.

- Let $s, s' \in S$ and $C = (c_{s,s'}) \in \mathbb{R}^{|S| \times |S|}$ be the transition probability matrix excluding action-labels in $AL(M)$ from B as follows:

$$
c_{s,s'} \triangleq \sum_{\substack{\mathrm{ns} \in \\ next(M,s) \cap SL(M)}} \frac{[\![lookup(M,\mathrm{ns})]\!](s, s')}{|next(M,s) \cap SL(M)|} .
$$

$$
\begin{aligned} & v' \cdot C' \cdot w' \\ &= v \cdot C \cdot w \\ & \left(\because \bigcup_{\substack{q' \in \\ \{s' \in S' \mid s' \vDash \phi_1 \wedge s' \nvDash \phi_2\}}} \{s \in S \mid q' = s|_{\mathbf{I}}\} = T_1 \ \wedge \bigcup_{\substack{q' \in \\ \{s' \in S' \mid s' \vDash \phi_2\}}} \{s \in S \mid q' = s|_{\mathbf{I}}\} = T_2 \right) \end{aligned}
$$

(7) Transform matrices into RHS as it is.

$$
v \cdot C \cdot w = (RHS) . \quad \square
$$

$$\sum_{s_1\in\{s\in S|s'_1=s|_{\mathbf{I}}\}}\cdots\sum_{s_t\in\{s\in S|s'_t=s|_{\mathbf{I}}\}}\prod_{i=0}^{t-1}\sum_{\substack{\mathtt{ns}\in\\ next(M,s_i)\cap SL(M)}}\frac{[\![lookup(M,\mathtt{ns})]\!](s_i,s_{i+1})}{|next(M,s_i)\cap SL(M)|}$$

$$=\sum_{s_1\in\{s\in S|s'_1=s|_{\mathbf{I}}\}}\cdots\sum_{s_{t-1}\in\{s\in S|s'_{t-1}=s|_{\mathbf{I}}\}}\prod_{i=0}^{t-2}\sum_{\substack{\mathtt{ns}\in\\ next(M,s_i)\cap SL(M)}}\frac{[\![lookup(M,\mathtt{ns})]\!](s_i,s_{i+1})}{|next(M,s_i)\cap SL(M)|}\times\left(\sum_{s_t\in\{s\in S|s'_t=s|_{\mathbf{I}}\}}\sum_{\substack{\mathtt{ns}\in\\ next(M,s_{t-1})\cap SL(M)}}\frac{[\![lookup(M,\mathtt{ns})]\!](s_{t-1},s_t)}{|next(M,s_{t-1})\cap SL(M)|}\right)$$

(by distributive property)

$$=\sum_{s_1\in\{s\in S|s'_1=s|_{\mathbf{I}}\}}\cdots\sum_{s_{t-1}\in\{s\in S|s'_{t-1}=s|_{\mathbf{I}}\}}\prod_{i=0}^{t-2}\sum_{\substack{\mathtt{ns}\in\\ next(M,s_i)\cap SL(M)}}\frac{[\![lookup(M,\mathtt{ns})]\!](s_i,s_{i+1})}{|next(M,s_i)\cap SL(M)|}\times\left(\sum_{\substack{\mathtt{ns}'\in\\ next(SLI(M,\mathbf{I}),s'_{t-1})\cap SL(SLI(M,\mathbf{I}))}}\frac{[\![lookup(SLI(M,\mathbf{I}),\mathtt{ns}')]\!](s'_{t-1},s'_t)}{|next(SLI(M,\mathbf{I}),s'_{t-1})\cap SL(SLI(M,\mathbf{I}))|}\right)$$

$$\left(\because\sum_{\substack{s'\in\\ \{s\in S|s|_{\mathbf{I}}=q\}}}\frac{[\![lookup(M,\mathtt{ns})]\!](s,s')}{|next(M,s)\cap SL(M)|}=\frac{[\![lookup(SLI(M,\mathbf{I}),\mathtt{ns}')]\!](s|_{\mathbf{I}},q)}{|next(SLI(M,\mathbf{I}),s|_{\mathbf{I}})\cap SL(SLI(M,\mathbf{I}))|}\text{ by definition of SLI}\right)$$

$$=\quad\cdots\quad\text{(by applying above steps recursively)}$$

$$=\prod_{i=0}^{t-1}\sum_{\substack{\mathtt{ns}'\in\\ next(SLI(M,\mathbf{I}),s'_i)\cap SL(SLI(M,\mathbf{I}))}}\frac{[\![lookup(SLI(M,\mathbf{I}),\mathtt{ns}')]\!](s'_i,s'_{i+1})}{|next(SLI(M,\mathbf{I}),s'_i)\cap SL(SLI(M,\mathbf{I}))|}\ .\ \square$$

Box III.

Lemma 5. *Let* $s_0\in\{s\in S\mid s\vDash\phi_1\wedge s\nvDash\phi_2\}$, $s'_0=s_0|_{\mathbf{I}}$, $t\in\mathbb{N}^+$, $s'_1,\ldots,s'_{t-1}\in\{s'\in S'\mid s'\vDash\phi_1\wedge s'\nvDash\phi_2\}$, *and* $s'_t\in\{s'\in S'|s'\vDash\phi_2\}$.

$$\sum_{s_1\in\{s\in S|s'_1=s|_{\mathbf{I}}\}}\cdots\sum_{s_t\in\{s\in S|s'_t=s|_{\mathbf{I}}\}}\prod_{i=0}^{t-1}\sum_{\substack{\mathtt{ns}\in\\ next(M,s_i)\cap SL(M)}}\frac{[\![lookup(M,\mathtt{ns})]\!](s_i,s_{i+1})}{|next(M,s_i)\cap SL(M)|}$$

$$=\prod_{i=0}^{t-1}\sum_{\substack{\mathtt{ns}'\in\\ next(SLI(M,\mathbf{I}),s'_i)\cap SL(SLI(M,\mathbf{I}))}}\frac{[\![lookup(SLI(M,\mathbf{I}),\mathtt{ns}')]\!](s'_i,s'_{i+1})}{|next(SLI(M,\mathbf{I}),s'_i)\cap SL(SLI(M,\mathbf{I}))|}\ .$$

Proof. Following unnumbered equation is given in Box III.

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.infsof.2022.106904.

References

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

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

[3] M. Jin, D. Shin, D.-H. Bae, 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, ACM, 2018, pp. 1698–1701.

[4] J. Kephart, J. Kephart, D. Chess, C. Boutilier, R. Das, J.O. Kephart, W.E. Walsh, An Architectural Blueprint for Autonomic Computing, IBM White Paper, 2003, pp. 2–10.

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

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

[7] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.

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

[9] G. Bellinger, D. Castro, A. Mills, Data, information, knowledge, and wisdom, 2004.

[10] N. Esfahani, S. Malek, Uncertainty in self-adaptive software systems, in: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 214–238.

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

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

[13] S.-W. Cheng, D. Garlan, B. Schmerl, Evaluating the effectiveness of the rainbow self-adaptive system, in: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, IEEE, 2009, pp. 132–141.

[14] S. Lawson, Worm on the sensor: What happens when IoT data is bad? IDG News Serv. URL https://www.computerworld.com/article/3151402/worm-on-the-sensor-what-happens-when-iot-data-is-bad.html.

[15] J.H. Ziegeldorf, O.G. Morchon, K. Wehrle, Privacy in the internet of things: threats and challenges, Secur. Commun. Netw. 7 (12) (2014) 2728–2742.

[16] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer (1) (2003) 41–50.

[17] A. Pnueli, L.D. Zuck, Probabilistic verification, Inform. and Comput. 103 (1) (1993) 1–29.

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

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

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

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

[22] P. Ballarini, B. Barbot, M. Duflot, S. Haddad, N. Pekergin, HASL: A new approach for performance evaluation and model checking from concepts to experimentation, Perform. Eval. 90 (2015) 53–77.

[23] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: International Conference on Computer Aided Verification, Springer, 2011, pp. 585–591.

[24] B. Boyer, K. Corre, A. Legay, S. Sedwards, PLASMA-lab: A flexible, distributable statistical model checking library, in: International Conference on Quantitative Evaluation of Systems, Springer, 2013, pp. 160–164.

[25] P. Bulychev, A. David, K.G. Larsen, M. Mikučionis, D.B. Poulsen, A. Legay, Z. Wang, UPPAAL-SMC: Statistical model checking for priced timed automata, 2012, arXiv preprint arXiv:1207.1272.
[26] M. Weiser, Program slicing, in: Proceedings of the 5th International Conference on Software Engineering, IEEE Press, 1981, pp. 439–449.
[27] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, Y. Markov, The first collision for full SHA-1, in: Annual International Cryptology Conference, Springer, 2017, pp. 570–596.
[28] Y. Xing, Z. Yang, C. Chen, J. Xue, Y. Dai, On the QoS of offline download in retrieving peer-side file resource, in: 2011 International Conference on Parallel Processing, IEEE, 2011, pp. 783–792.
[29] T. Klingberg, R. Manfredi, The Gnutella Protocol Specification v0. 6, Technical Specification of the Protocol, 2002.
[30] C. Courcoubetis, M. Yannakakis, Verifying temporal properties of finite-state probabilistic programs, in: [Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1988, pp. 338–345.
[31] R. Alur, T.A. Henzinger, Reactive modules, Form. Methods Syst. Des. 15 (1) (1999) 7–48.
[32] PRISM, 2020, http://www.prismmodelchecker.org/doc/semantics.pdf [Online; accessed 18-November-2020].
[33] A. Baouya, D. Bennouar, O.A. Mohamed, S. Ouchani, On the probabilistic verification of time constrained sysml state machines, in: International Conference on Intelligent Software Methodologies, Tools, and Techniques, Springer, 2015, pp. 425–441.
[34] Transmission, 2021, https://github.com/transmission/transmission accessed: 2021-02-25.
[35] H. Hansson, B. Jonsson, A logic for reasoning about time and reliability, Form. Asp. Comput. 6 (5) (1994) 512–535.
[36] D. Seo, D. Shin, Y.-M. Baek, J. Song, W. Yun, J. Kim, E. Jee, D.-H. Bae, 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, 2016, pp. 12–18.
[37] A. Legay, S. Sedwards, L.-M. Traonouez, Plasma lab: a modular statistical model checking platform, in: International Symposium on Leveraging Applications of Formal Methods, Springer, 2016, pp. 77–93.
[38] L. Li, H. Xiaoguang, C. Ke, H. Ketai, The applications of wifi-based wireless sensor network in internet of things and smart grid, in: 2011 6th IEEE Conference on Industrial Electronics and Applications, IEEE, 2011, pp. 789–793.
[39] S. Park, Z.M. Belay, D.-H. Bae, 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, 2019, pp. 2–9.
[40] A.A. Abusnaina, C.H. Yong, Cooperative multi-agent system for solving packet world problem in grid, in: 2009 IEEE 9th Malaysia International Conference on Communications (MICC), IEEE, 2009, pp. 754–758.
[41] M. Kwiatkowska, G. Norman, D. Parker, The PRISM benchmark suite, 2012.
[42] D. Knuth, A. Yao, Algorithms and Complexity: New Directions and Recent Results, Academic Press, 1976, Ch. The complexity of nonuniform random number generation.
[43] A. Gambi, M. Pezze, M. Young, SLA protection models for virtualized data centers, in: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, IEEE, 2009, pp. 10–19.
[44] I. Georgiadis, J. Magee, J. Kramer, Self-organising software architectures for distributed systems, in: Proceedings of the First Workshop on Self-Healing Systems, 2002, pp. 33–38.
[45] K.B. Gallagher, J.R. Lyle, Using program slicing in software maintenance, IEEE Trans. Softw. Eng. 17 (8) (1991) 751–761.
[46] G. Canfora, A. Cimitile, A. De Lucia, Conditioned program slicing, Inf. Softw. Technol. 40 (11–12) (1998) 595–607.
[47] A. Cimitile, A.D. Lucia, M. Munro, A specification driven slicing process for identifying reusable functions, J. Softw. Maint. Res. Pract. 8 (3) (1996) 145–178.
[48] N.E. Gold, D. Binkley, M. Harman, S. Islam, J. Krinke, S. Yoo, Generalized observational slicing for tree-represented modelling languages, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 547–558.
[49] J. Song, J.O. Tørring, S. Hyun, E. Jee, D.-H. Bae, Slicing executable system-of-systems models for efficient statistical verification, in: Proceedings of the 7th International Workshop on Software Engineering for Systems-of-Systems and 13th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems, IEEE Press, 2019, pp. 18–25.
[50] E.M. Clarke Jr., O. Grumberg, D. Kroening, D. Peled, H. Veith, Model checking, MIT Press, 2018.
[51] C.-K. Hur, A.V. Nori, S.K. Rajamani, S. Samuel, Slicing probabilistic programs, ACM SIGPLAN Not. 49 (6) (2014) 133–144.
[52] N. Goodman, V. Mansinghka, D.M. Roy, K. Bonawitz, J.B. Tenenbaum, Church: a language for generative models, 2012, arXiv preprint arXiv:1206.3255.
[53] T. Minka, J. Winn, J. Guiver, A. Kannan, Infer .NET 2.3, 2009, Software Available from http://research.microsoft.com/infernet.
[54] E.M. Clarke, O. Grumberg, D.E. Long, Model checking and abstraction, ACM Trans. Program. Lang. Syst. (TOPLAS) 16 (5) (1994) 1512–1542.
[55] M. Ammar, S. Ouchani, O.A. Mohamed, Symmetry reduction of time-triggered ethernet protocol, Procedia Comput. Sci. 19 (2013) 273–280.
[56] G.B. Hamad, O.A. Mohamed, System level SEUs propagation analysis via data flow-based reduction and quantitative model checking, in: 2017 First International Conference on Embedded & Distributed Systems (EDiS), IEEE, 2017, pp. 1–6.
[57] P.R. D'argenio, B. Jeannet, H.E. Jensen, K.G. Larsen, Reachability analysis of probabilistic systems by successive refinements, in: Joint International Workshop Von Process Algebra and Probabilistic Methods, Performance Modeling and Verification, Springer, 2001, pp. 39–56.
[58] D. Parker, G. Norman, M. Kwiatkowska, Game-based abstraction for Markov decision processes, in: Third International Conference on the Quantitative Evaluation of Systems-(QEST'06), IEEE, 2006, pp. 157–166.
[59] E.W. Weisstein, Gershgorin circle theorem, 2003, https://mathworld.wolfram.com/.