2022 International Symposium on Nonlinear Theory and Its Applications, NOLTA2022, Virtual, December 12-15, 2022

Black Box Checking of Mobile Robot Path Planning Satisfying Safety Hyperproperties

Naomi Kuze[†], Keiichiro Seno[†], and Toshimitsu Ushio[†]

†Graduate School of Engineering Science, Osaka University
1–3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

Abstract—A k-safety hyperproperty is a hyperproperty that is characterized by "bad prefixes." For example, it can express security policies for safety-critical and safety-related systems. Black box checking (BBC) is a promising formal verification method of systems whose internal structure is unknown. However, a BBC method for verifying hyperproperties has not been considered yet. We extend a BBC for k-safety hyperproperties, and apply it to verification of a security policy for path planning of a mobile robot, and demonstrate it with an illustrative example.

1. Introduction

A *hyperproperty* [1], a set of trace properties, allows us to represent relations over multiple traces. A *safety property* is a property that proscribes "bad thing" [1]. That is, in a safety property, "bad thing" does not happen during execution. According to [2], the "bad prefix" is (i) *finitely observable*, and (ii) *irremediable*. A *k-hyperproperty*, an extension of safety properties can express security policies for safety-critical or safety-related systems. Finkbeiner et al. [3] showed that a regular k-safety hyperproperty $\boldsymbol{S}$ can be represented by a finite automaton which accepts the violation (the bad-prefix) of $\boldsymbol{S}$. This representation makes it possible to perform an automata-theoretic approach for regular k-safety hyperproperties.

Black box checking (BBC) is a method that checks whether a black box system (BBS) whose internal structure is unknown satisfies a given specification. Peled et al. [4] established a BBC approach that utilizes automata learning algorithm [5]. In recent years, the BBC approach has been refined and applied to safety assurance for cyber-physical systems (CPSs) [6,7]. However, these BBC methods focus only on trace properties, but not on hyperproperties.

In this paper, we (1) propose a BBC method of mobile robot path planning satisfying a k-safety hyperproperty, utilizing an active automata learning, based on the Angluin's L^* algorithm, and (2) demonstrate the proposed method with an illustrative example.

The rest of the paper is organized as follows. We give preliminaries in Section 2. We present problem formulation in Section 3, and propose a BBC method for a mobile robot in Section 4. Section 5 shows the demonstration of the proposed method with an illustrative example. We conclude this paper in Section 6.

ORCID iDs Naomi Kuze: 0000-0001-8224-3476, Keiichiro Seno: 0000-0002-5837-8132, Toshimitsu Ushio: 0000-0002-4009-270X

2. Preliminaries

In this paper, let AP be a finite set of atomic propositions and $\Sigma := 2^{\mathsf{AP}}$ be a finite alphabet. Also, let Σ^ω (*resp.* Σ^*) denote the set of all infinite (*resp.* finite) traces (or words) over Σ. For a set A, let $\mathcal{P}(A)$ be the powerset of A, and let $\mathcal{P}^{fin}(A)$ be the set of all finite subsets of A. $|A|$ denotes the cardinality of a set A.

2.1. Hyperproperties

A *hyperproperty* [1] over an alphabet Σ is a set of sets of infinite traces over Σ. A hyperproperty $\boldsymbol{H}$ is a subset of $\mathcal{P}(\Sigma^\omega)$. A set of infinite traces T satisfies a hyperproperty $\boldsymbol{H}$, denoted by $T \models \boldsymbol{H}$, iff $T \in \boldsymbol{H}$.

The definition of safety can be extended to hyperproperties by generalizing a bad prefix from a finite trace to a finite set of finite traces [1]. For a finite set $T \in \mathcal{P}^{fin}(\Sigma^*)$ of finite traces and a set $T' \in \mathcal{P}(\Sigma^\omega)$ of infinite traces, T is a prefix of T', denoted by $T \leq T'$, iff

$$\forall t \in T,\ (\exists t' \in T',\ t \leq t').$$

Definition 1 (k-safety hyperproperty [1]) *A hyperproperty $\boldsymbol{S}$ is a k-safety hyperproperty iff*

$$\forall T \in \mathcal{P}(\Sigma^\omega). (T \notin \boldsymbol{S} \rightarrow \exists M \in \mathcal{P}^{fin}(\Sigma^*). (M \leq T \wedge |M| \leq k \wedge (\forall T' \in \mathcal{P}(\Sigma^\omega). M \leq T' \rightarrow T' \notin \boldsymbol{S}))).$$

We call M a *k-bad prefix* for $\boldsymbol{S}$. Intuitively, it is possible to check whether there is a violation of $\boldsymbol{S}$ by simply finding at most k traces that violate $\boldsymbol{S}$.

It is shown that a regular k-safety hyperproperty $\boldsymbol{S}$ can be represented by a finite automaton which accepts the violation (the bad-prefix) of $\boldsymbol{S}$ (Theorem 6 in [3]).

2.2. Automata

A finite automaton is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$, where Q is a finite set of states, $q_0 \in Q$ is an initial state, Σ is an input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and $F \subseteq Q$ is a set of accepting states. A run of $\mathcal{A}$ over a finite word $\sigma = \sigma_0\sigma_1 \ldots \sigma_n \in \Sigma^*$ is a finite sequence $r = q_0 q_1 \ldots q_{n+1} \in Q^*$, where $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all

$i = 0, 1, \dots, n$. A finite word σ is accepted by $\mathcal{A}$ if there exists a run r over σ that ends in an accepting state, i.e., $q_{n+1} \in F$. The set of all words accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$ and is called the *language* accepted by $\mathcal{A}$. The language that is accepted by a finite automaton over finite words is called a *regular language*. A finite automaton is called *deterministic* if $(q, \sigma, q') \in \delta$ and $(q, \sigma, q'') \in \delta$, then $q' = q''$, for all $q, q', q'' \in Q$ and $\sigma \in \Sigma$. If not, it is called *nondeterministic*.

A Büchi automaton is a tuple $\mathcal{B} = (Q, q_0, \Sigma, \delta, F)$ that has the same component as a finite automaton $\mathcal{A}$. The semantics of the Büchi automaton are defined over infinite input words in Σ^ω. A run of $\mathcal{B}$ over an infinite word $\sigma = \sigma_0\sigma_1 \cdots \in \Sigma^\omega$ is an infinite sequence $r = q_0q_1 \cdots \in Q^\omega$, where $(q_i, \sigma_i, q_{i+1}) \in \delta$ for all $i = 0, 1, \dots$. A run is accepting if at least one accepting state appears in it infinitely many times. An infinite word σ is accepted by $\mathcal{B}$ if and only if there exists at least one accepting run over σ. The set of all words accepted by $\mathcal{B}$ is denoted by $\mathcal{L}(\mathcal{B})$.

According to [3], for a k-safety hyperproperty $\boldsymbol{S}$, we can build a deterministic finite automaton (DFA) $\mathcal{A} = (Q, q_0, \Sigma^k, \delta_A, F_A)$ that accepts a finite word of k-tuples $\sigma \in (\Sigma^k)^*$ such that $unzip(\sigma)^*$ is one of the k-bad-prefixes for $\boldsymbol{S}$. In [3], such a DFA $\mathcal{A}$ is called a *k-bad-prefix automaton* for $\boldsymbol{S}$.

3. Problem formulation

We consider a mobile robot moving in a workspace consisting of N rooms. The work space is represented by a graph $G = (V, E)$, where where $V = \{v_0, v_1, \dots, v_{N-1}\}$ is a set of vertexes and $E \subseteq V \times V$ is a set of edges. Each vertex v_i denotes the room i and $(v_i, v_j) \in E$ means that there is a door between the room i and j by which the robot can enter the room j from the room i. Without loss of generality, we assume that the robot starts moving from the room 0. Let AP be a set of atomic propositions and we introduce a labeling function $L : E \to 2^{AP}$. Then, the behavior of the robot is modeled by a finite automaton $\mathcal{B} = (V, \Sigma, v_0, \delta)$, where V is the set of states and $\Sigma = 2^{\mathsf{AP}}$ is the alphabet, $v_0 \in V$ is the initial state, and $\delta : V \times \Sigma \to V$ is the transition function defined by

$$\delta(v, \sigma) = \begin{cases} v' & \text{if } (v, v') \in E \ \wedge \ \sigma = L((v, v')), \\ \text{undefined} & \text{otherwise.} \end{cases} \tag{1}$$

We assume that $\mathcal{B}$ is unknown and consider a problem such that we check if the behavior of the robot always satisfies an information-flow specification described by a k-safety hyperproperty $\mathbf{S}$. In the next section, we present a black box checking based method to check if the mobile robot satisfies the k-safety hyperproperty.

*For a finite word $\sigma = \sigma_0\sigma_1 \dots \sigma_n \in (\Sigma^k)^*$, $unzip(\sigma) = \{t_i \in \Sigma^* | 1 \le i \le k, 0 \le \forall j \le n.\ \ t_i[j] = \sigma_j[i]\}$, where $\sigma_i[j]$ represents the j-th element of $\sigma_i \in \Sigma^k$, i.e., $\sigma_i = (\sigma_i[1], \sigma_i[2], \dots, \sigma_i[k]\ )$. The inverse function of $unzip$ is denoted by zip.

4. Black box checking of mobile robot

In this section, we propose a BBC method for the mobile robot moving problem based on the L^* algorithm [5].

When we check whether a given automaton $\mathcal{B}$ satisfies a k-safety hyperproperty $\boldsymbol{S}$, we introduce the technique called *self-composition* for the emptiness check. The k-bad-prefix automaton $\mathcal{A}$ for $\boldsymbol{S}$ is defined over Σ^k while $\mathcal{B}$ is defined over Σ, and we cannot evaluate directly the emptiness of $\mathcal{L}(\mathcal{B}) \cap \mathcal{L}(\mathcal{A})$.

For a k-tuple $\theta = (\theta_1, \dots, \theta_k)$, we denote $\theta[i] = \theta_i$ for all $1 \le i \le k$.

Definition 2 (k-fold self-composition) *For a Büchi automaton $\mathcal{B} = (S, s_0, \Sigma, \delta_B, F_B)$, the k-fold self-composition of $\mathcal{B}$ is the Büchi automaton $\mathcal{B}^k = (S^k, s'_0, \Sigma^k, \delta'_B, F^k_B)$, where $s'_0 := (s_0, \dots, s_0) \in S^k$, and δ'_B is a transition relation. δ'_B is defined by $(s, \sigma, s') \in \delta'_B$ iff $(s[i], \sigma[i], s'[i]) \in \delta_B$ for all $s, s' \in S^k$, $\sigma \in \Sigma^k$, and $i = 1, \dots, k$.*

Then, $\mathcal{L}(\mathcal{B}^k)$ is given by

$$\mathcal{L}(\mathcal{B}^k) = \{zip(\{t_1, \dots, t_k\}) \mid t_1, \dots, t_k \in \mathcal{L}(\mathcal{B})\}.$$

Automata-theoretic model checking methods rely on the construction of a product automaton. In our case, we need to construct the product automaton between a k-fold self-composition $\mathcal{B}^k$ and a k-bad prefix-automaton $\mathcal{A}$.

Definition 3 (product automaton) *The product automaton $P = \mathcal{B}^k \times \mathcal{A}$ between a k-fold self-composition $\mathcal{B}^k = (S^k, s'_0, \Sigma^k, \delta'_B, F^k_B)$, and a k-bad prefix-automaton $\mathcal{A} = (Q, q_0, \Sigma^k, \delta_A, F_A)$ is defined as $P = (S^k \times Q, p_0, \Sigma^k, \delta_P, F^k_B \times F_A)$, where $p_0 := (s'_0, q_0)$ is a initial state, and $\delta_P \subseteq (S^k \times Q) \times \Sigma^k \times (S^k \times Q)$ is a transition relation defined by $\delta_P = \{((s, q), \sigma, (s', q')) \mid (s, \sigma, s') \in \delta'_B \wedge (q, \sigma, q') \in \delta_A\}$.*

We have $\mathcal{L}(\mathcal{B}^k \times \mathcal{A}) = \mathcal{L}(\mathcal{B}^k) \cap \mathcal{L}(\mathcal{A})$.

Given a system $\mathcal{B}$ and a k-safety hyperproperty $\mathbf{S}$, we can check whether $\mathcal{B}$ satisfies $\mathbf{S}$ by the following procedure:

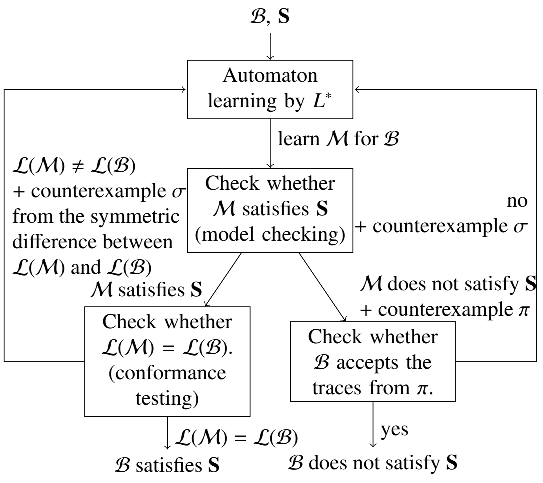
1. Construct k-bad-prefix automaton $\mathcal{A}$ for $\mathbf{S}$. $\mathcal{A}$ accepts words $\sigma \in (\Sigma^k)^*$ such that $unzip(\sigma)$ is one of k-bad-prefixes for $\boldsymbol{S}$.

2. Construct the k-fold self-composition $\mathcal{B}^k$ of $\mathcal{B}$.

3. Construct the product automaton $\mathcal{B}^k \times \mathcal{A}$ whose language is composed of the k-tuples of traces of $\mathcal{B}$ that does not satisfy $\mathbf{S}$.

4. Check whether $\mathcal{L}(\mathcal{B}^k) \cap \mathcal{L}(\mathcal{A}) = \emptyset$. If it is empty, $\mathcal{B}$ satisfies $\mathbf{S}$. This is because the emptiness means that all pairs of traces that are accepted by $\mathcal{B}$ cannot be a bad-prefix for $\boldsymbol{S}$. If not, there is a counterexample $\sigma \in (\Sigma^k)^*$ in the intersection $\mathcal{L}(\mathcal{B}^k) \cap \mathcal{L}(\mathcal{A})$, which means that $\mathcal{B}$ does not satisfy $\mathbf{S}$.

The flow of BBC for k-safety hyperproperties is shown in Fig. 1. We showed the details of our algorithm in [8].



$\mathcal{B}$, **S**
Automaton learning by L^*
learn $\mathcal{M}$ for $\mathcal{B}$
Check whether $\mathcal{M}$ satisfies **S** (model checking)
$\mathcal{L}(\mathcal{M}) \neq \mathcal{L}(\mathcal{B})$ + counterexample σ from the symmetric difference between $\mathcal{L}(\mathcal{M})$ and $\mathcal{L}(\mathcal{B})$
no + counterexample σ
$\mathcal{M}$ satisfies **S**
$\mathcal{M}$ does not satisfy **S** + counterexample π
Check whether $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{B})$. (conformance testing)
Check whether $\mathcal{B}$ accepts the traces from π.
$\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{B})$
yes
$\mathcal{B}$ satisfies **S**
$\mathcal{B}$ does not satisfy **S**

Figure 1: Black box checking for k-safety hyperproperties. $\mathcal{M}$ corresponds to a conjectured automaton returned by the L^* algorithm. A conjectured automaton is referred to as a hypothesis, and it converges to a correct hypothesis (the system model $\mathcal{B}$).

5. Illustrative example

In this section, we consider a work space depicted in Fig. 2, where there are 9 rooms labeled by R_i ($i = 0, 1, \ldots, 8$) and the arrow indicates the direction of the movement of the robot. The robot starts at R_0 marked in gray. When the robot is in R_0, we give the command which room to move to, the right (R_1) or the left (R_5). Then, the robot follows the command, and moves to either R_1 or R_5. Let Y be a set of a "yellow room" and we assume that an intruder can observe the movement of the robot into a yellow room but cannot identify the room. In Fig. 3, $Y = \{R_2, R_6\}$.

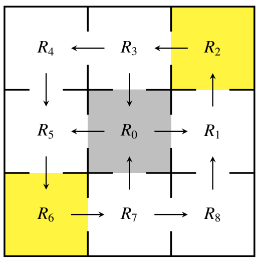


Figure 2: Floor map of the example ($Y = \{R_2, R_6\}$).

A set of atomic propositions is $\mathsf{AP} = I_H \cup I_L \cup O_L$, where I_H, I_L, and O_L are sets of high-level input variables, low-level input variables, and low-level output variables, respectively. We consider the following atomic propositions: v (move up or down), h (move left or right), y (move to a yellow room), and r (when the robot is in R_0, move to the right room R_1; if r is false, the robot moves to the left room R_5). We assume that $I_H = \{r\}$, $I_L = \{v, h\}$, and $O_L = \{y\}$. In other words, the intruder gets the information that the robot is moving left or right, up or down, and movement into a yellow room. On the other hand, he/she cannot get information about which room the robot moved from R_0 to the left or right. The behavior of the robot is modeled as a deterministic finite automaton shown in Fig. 3. The initial state is s_0. State s_i corresponds to the room R_i, for each $i = 0, 1, \ldots, 8$. The transitions are labeled by the inputs $\sigma \in \Sigma = 2^{\mathsf{AP}}$.

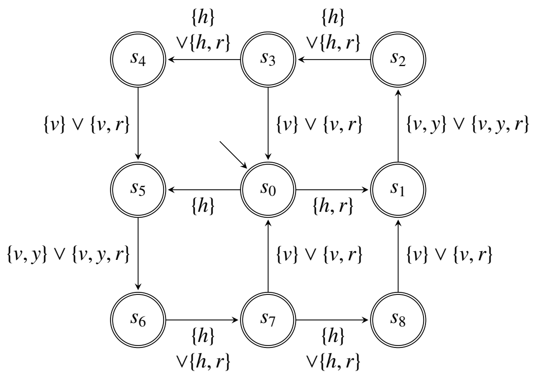
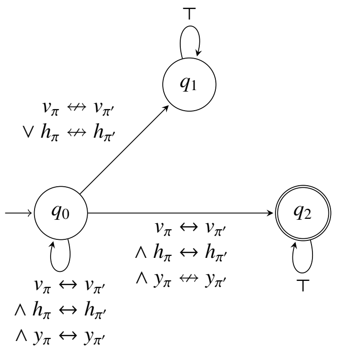


Figure 3: Finite automaton of the example.

We now verify that the behavior of the robot satisfies the noninterference specification — the commands (r or $\neg r$) in the room R_0 do not interfere with the low-level output y. In other words, the intruder who observes the low-level variables can not infer information about the high-level input r. This specification is a 2-safety hyperproperty whose bad-prefix automaton $\mathcal{A}$ is shown in Fig. 4.



$\top$
q_1
$v_\pi \not\leftrightarrow v_{\pi'} \vee h_\pi \not\leftrightarrow h_{\pi'}$
q_0
q_2
$v_\pi \leftrightarrow v_{\pi'} \wedge h_\pi \leftrightarrow h_{\pi'} \wedge y_\pi \not\leftrightarrow y_{\pi'}$
$v_\pi \leftrightarrow v_{\pi'} \wedge h_\pi \leftrightarrow h_{\pi'} \wedge y_\pi \leftrightarrow y_{\pi'}$
$\top$

Figure 4: The 2-bad-prefix automaton $\mathcal{A}$ for **S**

We consider several patterns Y of the yellow rooms and performed a BBC to check that the behavior of the robot satisfies **S**. Table 1 shows the sets Y considered in our experiments and the results of BBC for each pattern.

First, we consider the case where $Y = \{R_2, R_6\}$ (see Fig. 2). Since we do not have a model of the system at the beginning, we utilize the L^* algorithm to learn the system model. The final model $\mathcal{M}_5$ has six states, and five of them are accepting states (see Fig. 6). We have that $\mathcal{M}_5$ satisfies **S** (by model checking), and that $\mathcal{M}_5$ is equivalent to the

Table 1: Experimental results.

Y	# hypothesis	# states	specification
$\{R_0\}$	5	6	true
$\{R_2\}$	3	4	false
$\{R_2, R_6\}$	5	6	true
$\{R_0, R_2, R_6\}$	3	6	true
$\{R_2, R_6, R_8\}$	7	9	false

system (by conformance testing). Therefore, we conclude that the behavior satisfies the noninterference specification. Thus, in the case where the BBS satisfies the specification, the BBC algorithm continues to learn the system model until it becomes equivalent to the BBS.

Second, we discuss the case where $Y = \{R_2\}$. The final model $\mathcal{M}_3$ has four states, and three of them are accepting states (see Fig. 7). We have that $\mathcal{M}_3$ does not satisfies $\mathbf{S}$ because $\mathcal{L}(\mathcal{M}_3^2 \times \mathcal{A}) \neq \emptyset$ (by model checking). A trace $\pi = (\{h\}, \{h, r\})(\{v\}, \{v, y\})$ is returned as its counterexample. Next, we confirm that both of two traces $t_1 = \{h\}\{v\}$ and $t_2 = \{h, r\}\{v, y\}$ such that $t_1, t_2 \in \mathit{unzip}(\pi)$ are accepted by the robot planning system. Therefore, we conclude that the system does not satisfy the specification. The pair of the two traces t_1, t_2 is a counterexample (see Fig. 5). Note that the final conjectured model $\mathcal{M}_3$ is not equivalent to the BBS. Thus, in the case where the BBS does not satisfy the specification, it is possible to conclude that the BBS does not satisfy the specification without learning its model until a model equivalent to the BBS is obtained.

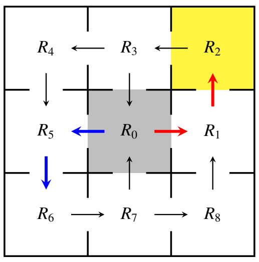


Figure 5: The pair of the two trace $t_1 = \{h\}\{v\}$ (blue) and $t_2 = \{h, r\}\{v, y\}$ (red) in the case where $Y = \{R_2\}$.

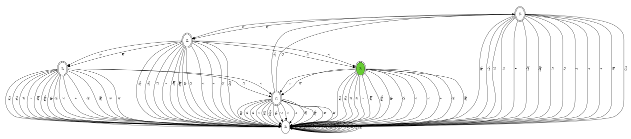


Figure 6: Learned automaton ($Y = \{R_2, R_6\}$).

6. Conclusion

In this paper, we focus on an extension of BBC to k-safety hyperproperties, and applied it to mobile robot path planning. We demonstrate the proposed mechanism with an illustrative example.

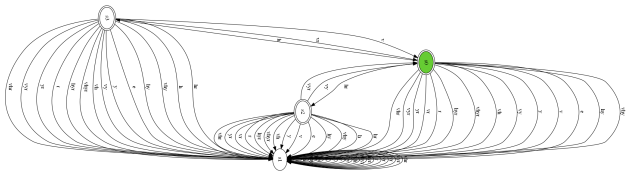


Figure 7: Learned automaton ($Y = \{R_2\}$).

In future work, we will extend the proposed BBC method to hyperproperties described by more general classes of hyperproperties.

Acknowledgments This work was supported by JST CREST Grant Number JPMJCR2012.

References

[1] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, no. 6, pp. 1157–1210, 2010.

[2] B. Alpern and F. B. Schneider, "Defining liveness," *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 1985.

[3] B. Finkbeiner, L. Haas, and H. Torfah, "Canonical representations of k-safety hyperproperties," in *Proc. 2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019, pp. 17–31.

[4] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black box checking," *IFIP Advances in Information and Communication Technology*, vol. 28, pp. 225–240, 1999.

[5] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[6] M. Waga, "Falsification of cyber-physical systems with robustness-guided black-box checking," in *Proc. the 23rd International Conference on Hybrid Systems: Computation and Control*, no. 11. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 1–13.

[7] J. Shijubo, M. Waga, and K. Suenaga, "Efficient black-box checking via model checking with strengthened specifications," *Lecture Notes in Computer Science*, vol. 12974, pp. 100–120, Oct. 2021.

[8] N. Kuze, K. Seno, and T. Ushio, "Learning-based black box checking for k-safety hyperproperties," *submitted to IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences.*