Analysis space reduction with state merging for ensuring safety properties of self-adaptive systems

Kazuya Aizawa
Waseda University
Tokyo, Japan
k.a.s-s-i-t-w@ruri.waseda.jp

Kenji Tei
Waseda University /
National Institute of Informatics
Tokyo, Japan
ktei@aoni.waseda.jp

Shinichi Honiden
Waseda University /
National Institute of Informatics
Tokyo, Japan
honiden@nii.ac.jp

***Abstract*—Analyzing guaranteeable safety properties in a running environment aids the decision making of self-adaptive systems. Our previous work generates and updates an analysis space with respect to environmental changes for identifying guaranteeable safety properties efficiently. However, our work cannot use the existing technique for reducing the analysis space, which means that its analysis space has a state explosion problem. In this paper, we propose a new reduction method that merges states while preserving information required for the safety properties analysis. We prove that our technique satisfies the condition for identifying guaranteeable safety properties. In addition, we evaluate the reduction in gives by using a production cell example and confirm that, in the best case, our proposal reduces the analysis space as much as that of a reachability analysis technique that cannot be applied to safety properties analysis.**

***Index Terms*—Self-adaptive system, Discrete controller synthesis ,Safety property, Space reduction**

I. Introduction

Self-adaptation is an essential technique for software systems deployed in uncertain and changeable environments. The systems may not satisfy their requirements if they cannot adapt their behavior to environmental changes. As such, they are expected to have alternative behaviors that they can switch between in response to environmental changes. Moreover, the safety properties of the systems must conform to specifications on the behaviors in environments that change. Safety properties here mean conditions and assurances that "bad things do not happen" [1]. For example, one safety property of a production cell [2] would be "do not put any unprocessed material on the out-tray". If the safety property is violated, the production may become faulty or even unsafe. In contrast, it is hard for engineers to guarantee safety properties for every environment. While some properties can be guaranteed under certain assumptions about the environment, they are no longer guaranteed when the environment changes such that the assumptions become invalid. Our purpose is *to identify guaranteeable safety properties* when the environment changes.

A number of studies [3]–[7] have addressed this problem. Analyses can be classified into two categories: design-time analysis and runtime analysis. Design-time analysis [3], [4] predicts possible environmental changes and identifies the requirements that are affected by the changes. These techniques have a low calculation cost at runtime, but they cannot deal with environments which are not foreseen at design time. In contrast, runtime analysis [5]–[7] collects environmental information and analyzes it to identify guaranteeable requirements at runtime. These techniques enable systems to deal with environmental changes more flexibly at the expense of calculation cost. Reducing the calculation cost is one of the important issues in runtime analysis.

Our previous work [8] proposed an efficient technique that identifies guaranteeable safety properties at runtime. This technique is based on two-player game analysis [9]. It generates, at design time, an analysis space from an environment model in the form of a labeled transition system (LTS) and safety properties in the form of fluent linear temporal logic (FLTL). When the environment changes at runtime, it updates the space and analyzes the updated part to identify the guaranteeable safety properties. Our technique successfully reduces the calculation time at runtime by localizing the analysis to only the updated part. However, our technique still suffers from a problem wherein the state space increases explosively as the number of safety properties. Thus, it can't resolve huge problems because of shortage of memory.

State-space explosion is a big issue and largely investigated in many domains which use state transition models [10]–[18]. To handle this problem, model checking and discrete controller synthesis typically reduce the safety property analysis to a reachability analysis [10]. This technique checks the reachability of any state that violates the safety properties. States in which violations of a safety property occur are removed from the analysis space. However, this technique cannot be used to identify guaranteeable safety properties. In identifying guaranteeable safety properties, it is required to check the reachability of state that violates other safety properties even after a violation of a safety property has occurred. Thus, the problem is that states after one in which a violation of a safety property has occurred cannot be removed.

In this paper, we propose a space reduction technique which preserves the conditions of the safety properties analysis. A key idea is not removing states but merging them. In

The research was partially supported by National Institute of Information and Communications Technology (NICT) and JSPS KAKENHI Grant Number 18H03225, 17H00732.

identifying guaranteeable safety properties, once the safety properties are violated, the information about these properties can be ignored. Our technique merges states after a violation of a safety property has occurred into ones that have same information other than the violated safety property. In so doing, our technique reduces the number of states and identifies guaranteeable safety properties at the same time.

We evaluated our technique through case studies of production cells [2]. The results showed that it reduces the size of the analysis space and extends the range of application. In addition, we compared our technique with a reachability analysis technique [10] in terms of the size of the analysis space. Although the reachability analysis cannot be used to identify guaranteeable safety properties, it can be an indicator of the reduction effect. We confirmed that, in the best case, our technique reduced states by as much as the reachability analysis technique did.

The remainder of the paper is organized as follows. Section II introduced the related works. A motivating example is shown in section III. Background knowledge is explained in section IV. Our proposed method is presented in section V and evaluated in section VI. Conclusions follow in section VII.

II. Related work

Self-adaptive systems analyze environmental changes in order to decide their specifications. These analyses are roughly classified into design time and runtime. Design-time analysis identifies requirements that are satisfied in the changed environment. At runtime, the self-adaptive systems decide the specifications according to the design-time analysis. D'Ippolito et al. [3] devised multi-tier environment models that provide graceful degradation. This method requires a prediction of the order in which assumptions are broken in the environment; extreme degradation may happen if the prediction is wrong. The method of Cailliau et al. [4] identifies obstacles to the system's goals and prepares countermeasures to them at design time. This technique has a risk that the relationships between the obstacles and the countermeasures may change while running the systems.

Runtime analysis uses the environmental information to identify which requirements can be satisfied. Calinescu et al. [5] prepared several specifications in the form of a discrete-time Markov chain with environment parameters. They get the parameter values and identify which specification satisfies more requirements with probabilistic model checking. Qian et al. [6] proposed a hybrid adaptation strategy combining design-time and runtime analysis. They store the context, adaptation configuration, and its effect as a case when they execute an adaptation at runtime. They reuse the case if the system faces a similar context. Camara et al. [7] modeled the environment as a stochastic game and composed an adaptation strategy that maximizes the rewards of the game at runtime.

Our previous work [8] identifies guaranteeable safety properties in changing environments. This method is based on a two-player game [9] and generates an analysis space from an environment model and a formalized set of safety properties; it however suffers from state explosion problem. Partial order reduction [19], which is a popular technique in model checking, ignores the order of state transitions that have no effect on the checking. Shuanglong et al. [13] apply partial order reduction to LTL, including next the operator, by using heuristics. Christoph et al. [11] focus on model checking in an adaptive case management and abstract parts which are not related to model checking. Ciolek et al. [14] proposed a reduction technique for discrete controller synthesis. They confine the analysis space by using a domain-independent heuristic. Giannakopoulou et al. [10] remove a state after a property violation when they generate an analysis space from a system model in LTS and safety properties in LTL. However, such states must not be removed from the analysis space when identifying guaranteeable safety properties, because our work analyzes other properties even after a violation happens.

State merging with abstraction is a classical approach. Many works abstract something which has little effect to their problem domains. Grumberg el al. [16] abstract the uncertainty of state transition and reduce the complexity of the game in order to identify the non-losing strategies in a 3-valued model checking game. Hussien et al. [17] abstract the uncertainty of state transition in order to synthesize the controller in the discretized model of a continuous system. Lomuscio et al. [15] abstract the agents' behavior which are irrelative to the verification of multi-agent systems against to alternating-time temporal logic specification. Burns et al. [18] abstract processes that consist search space for parallel model checking. The abstraction is used for duplicate detection which enables parallel model checking more efficient.

In our problem, we abstract the number of times each of the safety properties has been violated. We ignore the violations after the first time violation of each of the properties when merging states of an analysis space.

III. Motivating example

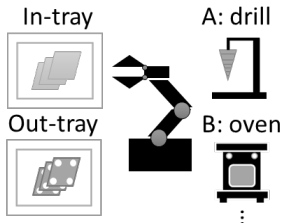

Fig. 1. Robot arm system in the production cell

We consider the scenario of a production cell [2], which processes materials in a factory. Fig.1 shows the layout in the factory. The cell has an in-tray, out-tray, several processing machines, and a robot arm. When a processing request is received, the robot arm picks a material from the in-tray, moves it to a processing machine, puts the material on the machine, which then processes it, picks up the processed material from the machine, moves it to the out-tray, and puts

the material. The robot arm system has to decide on the basis of the request which machine is to be used. For example, the robot arm may choose a drill when the request is "make a hole in a material" or it may choose require an oven to meet another request " bake a material".

The safety properties of the system are about the behaviors of the robot arm, such as "do not move a material to an undesignated place" or "put only processed materials on the out-tray". There are several properties that require assumptions to be made about the environment, such as "machines never break materials while processing." Invalidating such assumptions means that some properties are no longer guaranteed. For example, the robot arm would be compelled to put broken material on the out-tray if the oven overheated and burned the material black. Our purpose is to identify safety properties that are still guaranteeable in such an environment.

IV. Background

Here, we show how to generate an analysis space from an environment model and a set of safety properties. The environment model describes interactions between a software system and its environment in LTS, which has two types of action: actions that are controllable by the system and those that are uncontrollable by it. We will assume that all the LTS models are deterministic. An analysis space is a labeled transition kripke structure (LTKS), which has an LTS model with a proposition and a valuation function. In addition, we define paths of the LTKS and the valuation function for them.

Definition 1. *(Labelled Transition Kripke Structure and the valuation function for its path) A labeled transition Kripke structure (LTKS) is $E = (S, A, \Delta, s_0, P, v)$, where S is a finite set of states, $A = A_C \uplus A_M$ is a communicating alphabet that, we assume, is partitioned into controllable and uncontrollable actions, $\Delta \subseteq (S \times A \times S)$ is a transition relation, s_0 is an initial state, P is a set of propositions and $v : S \to 2^P$ is a valuation function for states. $\pi = s_0, \ell_0, s_1, \ell_1, ...$ is a path of E, where s_0 is an initial state, and for every $i \geq 0$, we have $(s_i, \ell_i, s_{i+1}) \in \Delta$. We denote the set of infinite paths of E by Π. $v_\pi : \Pi \to 2^P$ is a valuation function for paths such that $v_\pi(\pi) = \{p \in P | p \in v(s), s \in \pi\}$.*

The safety properties are formalized in fluent linear temporal logic [10] as propositions, which in turn consist of actions in LTS, normal operators, such as ¬(negation), ∧ (and), ∨ (or), → (implies), and temporal operators, such as X (next), U(until), W(weak until), □(always), ◇(eventually). They are expressed in the form "□p".

The safety properties can be monitored using tester models [10] in LTKS.

Definition 2. *(Tester model for Safety property) Given a safety property p, a LTKS model $T = (S_T, A_T, \Delta_T, s_{0T}, P_T, v_T)$ is a tester model, where $P_T = \{\neg p\}$ and $v_T(s) = \{\neg p\}$, if s is a violation state of p; otherwise $v_T(s) = \emptyset$. s is a dead-end state if $v(s) = \{\neg p\}$.*

Fig. 2 shows an example set of an environment model and two tester models. The model on the left is one of a simple environment for the production cell. In it, a robot arm system waits for a processing request and the start command. After receiving the start command, it processes a material and then waits for the next request. The controllable actions are "wait" and "processA/B", and the uncontrollable actions are "start" and "requestA/B". The safety properties of the example are as follows: "requestA/B must not be received again before finishing processA/B". The middle model is the tester for property A, and the right one is the tester for property B. The orange state in the middle model represents a violation of property A, while the yellow state of the right model represents a violation of property B.

- $propertyA = \Box(requestA \to X(!requestA\ W\ processA))$
- $propertyB = \Box(requestB \to X(!requestB\ W\ processB))$

The analysis space is generated from the environment model and tester models. The space is generated by modified parallel composition [10], which is similar to normal parallel composition but only follows the behavior of the environment model. "*" in the below definition is a single action that represents all actions in the environment model except for those in one of the tester model.

Definition 3. *(Modified Parallel Composition) Given an environment model $E = (S_E, A_E, \Delta_E, s_{0E}, P_E, v_E)$ and tester automaton $T = (S_T, A_T, \Delta_T, s_{0T}, P_T, v_T)$ such that $A_T \backslash \{*\} \subseteq A_E$, the modified parallel composition $E \parallel_* T$ is the LTKS $E \parallel_* T = (S, A, P, \Delta, v, s_0)$, where $S \subseteq S_E \times S_T$, $A = A_E$, $s_0 = (s_{0E}, s_{0T})$, $v((s_E, s_T)) = v_E(s_E) \cup v_T(s_T)$, and Δ is the smallest relation that satisfies the rules below.*

$$\frac{s_E \xrightarrow{\ell} s'_E, s_T \xrightarrow{\ell} s'_T}{(s_E, s_T) \xrightarrow{\ell} (s'_E, s'_T)} \ell \in A_E \cap A_T, \quad \frac{s_E \xrightarrow{\ell} s'_E, s_T \xrightarrow{*} s_T}{(s_E, s_T) \xrightarrow{\ell} (s'_E, s_T)} \ell \in A_E \backslash A_T$$

Modified parallel composition with multiple tester models is defined inductively, e.g., $E \parallel_* T_1 \parallel_* T_2 ... = (...((E \parallel_* T_1) \parallel_* T_2)...)$.

Fig. 3 is an example of an analysis space generated from the environment model and the tester models in Fig. 2. The orange states violate property A, the yellow states violate property B, and states having both colors violate both A and B. Let v_π be a path valuation function of the space and considering the path $\pi_1 = 0, wait, 1, requestB, 2, start, 3, processA, 4, wait, 5, requestB, 6, ...$ such that $\{14, 15, ...17, 21, ..., 29\} \not\subseteq \pi_1$. The result of the path valuation function is $v_\pi(\pi_1) = \{\neg A\}$, because π_1 has orange states like "6". Therefore, π_1 violates property A. In contrast, π_1 does not violate property B because it doesn't have any yellow states of states including both colored states. The analysis space tells us which properties a given path violates.

The analysis space can be regarded as a two-player game [9]. This idea is derived from a controller synthesis technique [20]. The game players are the specification and the environment. The specification chooses transitions with controllable actions, and the environment chooses transitions with uncontrollable actions. A path of the game π is winning for the

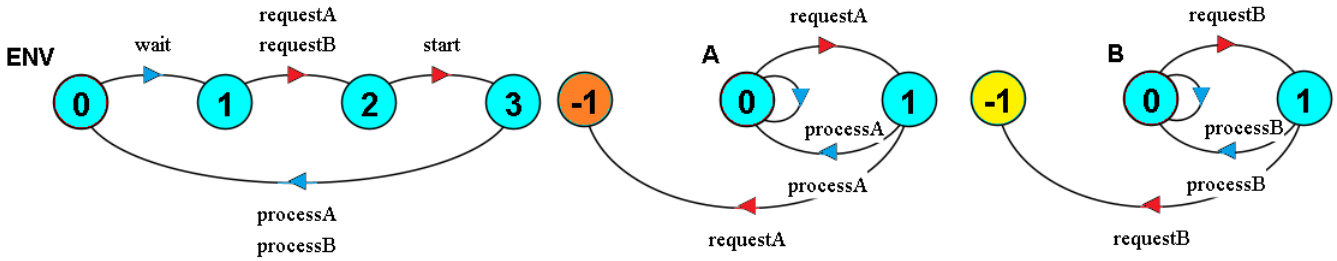

Fig. 2. Example of environment model and safety tester model

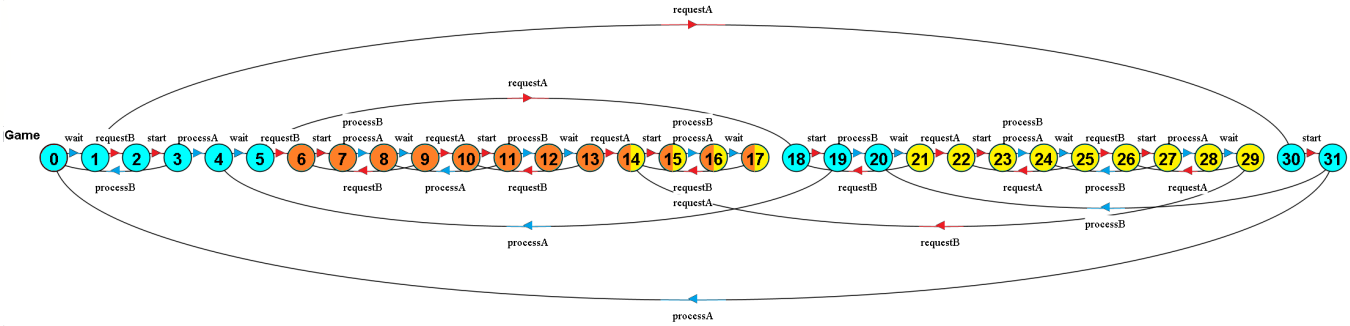

Fig. 3. Example of analysis space

environment when the result of a path valuation function is $v_\pi(\pi) \neq \emptyset$. Any other paths are winning for the specification.

Definition 4. *(Two-player Safety Game) A two-player safety game (hereafter safety game) is* $SG = (S_{sg}, \Gamma^-, \Gamma^+, s_{sg0}, X)$*, where* S_{sg} *is a finite set of states,* $\Gamma^- \subseteq S_{sg} \times S_{sg}$ *is a transition relation and similarly for* Γ^+*,* $s_{sg0} \in S_{sg}$ *is the initial state, and* $X \subseteq 2^{S_{sg}}$ *such that* $s_{sg0} \notin x$ *is a winning condition. We denote* $\Gamma^-(s_{sg}) = \{s'_{sg} | (s_{sg}, s'_{sg}) \in \Gamma^-\}$ *and similarly for* Γ^+*. A play on SG is a sequence* $\pi = s_{sg0}, \gamma_0, s_{sg1}, \gamma_1 ...$ *where for every* $i \geq 0$*,* $\gamma_i \in \Gamma^- \cup \Gamma^+$ *is* (s_{sgi}, s_{sgi+1})*. If plays contain a state in* $x \in X$*, the plays are winning for the environment in SG. Any other plays are winning for the specification.*

we can translate the LTKS $E = (S, A = A_C \cup A_M, \Delta, s_0, P, v)$ to the corresponding safety game $SG = (S_{sg}, \Gamma^-, \Gamma^+, s_{sg0}, X)$ as below.

- S_{sg} corresponds to S
- Γ^- corresponds to $\{(s, s') | s, s' \in S, (s, a_M, s') \in \Delta, a_M \in A_M\}$
- Γ^+ corresponds to $\{(s, s') | s, s' \in S, (s, a_C, s') \in \Delta, a_C \in A_C\}$
- s_{sg0} corresponds to s_0
- X corresponds to $\{S_{\neg P} \subseteq 2^S | s \in S_{\neg P}, s \in S, v(s) = \neg P, \neg P \subseteq P\}$

Results of path valuation function of the analysis space correspond to winning or loosing of the game. More precisely, given LTKS and the corresponding SG, for all path $\pi \in \Pi$, $v(\pi) \neq \emptyset$ means that the corresponding play of SG is winning for the environment. Any other plays are winning for the specification. We identify winning regions of the analysis space for each property. Moreover, the regions can be used to identify guaranteeable safety properties. To make the regions, all the paths in the environment model must be contained in the analysis space. Therefore, we cannot use the existing reduction technique [10] that removes states after properties violations.

V. State Merge Reduction

Here, we place two conditions on the analysis space reduction for identifying the guaranteeable safety properties and propose a new parallel composition for which the conditions hold.

A. Condition for reduction

Paths and the results of their valuation function must be preserved after the reduction in order to identify guaranteeable safety properties in an analysis space. More formally, given an original analysis space $E_{Orig} = (S_o, A, \Delta_o, s_{o0}, P, v_o)$ and its reduction space $E_{Redu} = (S_r, A, \Delta_r, s_{r0}, P, v_r)$, there are two features that must be preserved between the two spaces for identifying guranteeable safety properties.

- Condition 1: For any path $\pi_o = s_{o0}, \ell_{o0}, s_{o1}, \ell_{o1} ... \in \Pi_o$ of E_{Orig}, there exists a path $\pi_r = s_{r0}, \ell_{r0}, s_{r1}, \ell_{r1} ... \in \Pi_r$ of E_{Redu} such that $\ell_{oi} = \ell_{ri}$ for all $i \geq 0$. We denote such a relation of paths as $\pi_o = \pi_r$.
- Condition 2: If $\pi_o = \pi_r$, then $v_o(\pi_o) = v_r(\pi_r)$.

The results of the path valuation function correspond to winning or losing the game, as mentioned above. Therefore, winning or losing the game also corresponds to the original analysis space if the reduction space in which the above conditions hold is regarded as a safety game. Guaranteeable

safety properties are identified on the basis of winning or losing the game. Thus, knowing the reduction space in which the above conditions hold enables us to analyze the safety properties of as the original space.

B. State merge parallel composition

The key idea behind our reduction technique is merging states that occur affter one that has a violation. A path which reaches a state in which a safety property is violated always violates the property regardless of the sequence remaining after reaching that state. Thus, we can merge the states that are after the first property violations. To do so, we have to change the valuation function, because these "violation" states are merged with normal ones. Therefore, we define another LTKS that has a valuation function for transitions rather than states. We also define the valuation function for its paths.

Definition 5. *(Transition valuation LTKS and the valuation function for its paths) A transition valuation LTKS is* $E_\Delta = (S, A, \Delta, s_0, P, v_\Delta)$*, where* S, A, Δ, s_0 *and* P *are same as normal LTKS and* $v_\Delta : \Delta \to 2^P$ *is a valuation function for transitions. a path* $\pi_\Delta = s_0, \ell_0, s_1, \ell_1, ...$ *and the set of the paths* Π_Δ *are same as normal LTKS. We say that* $\delta \in \pi$ *if for all* $i \geq 0$*,* $\delta = (s_i, \ell_i, s_{i+1})$ *and there exists a sequence* s_i, ℓ_i, s_{i+1} *in* π*.* $v'_\pi : \Pi \to 2^P$ *is a valuation function for paths such that* $v_\pi(\pi) = \{p \in P | p \in v(\delta), \delta \in \pi\}$*.*

Hereafter, we call the model in definition 1 the normal LTKS and the model in definition5 the transition valuation LTKS . A normal LTKS $E = (S, A, \Delta, s_0, P, v)$ can be converted into a transition valuation LTKS $E_\Delta = (S, A, \Delta, s_0, P, v_\Delta)$ where $v_\Delta(\delta) = \{p \in P | p \in v(s'), \delta = (s, a, s'), s, s' \in S, a \in A\}$.

The transition valuation LTKS can be regarded as the a safety game in analogy with the normal LTKS.

Definition 6. *(Transition valuation Safety Game) A transition valuation safety game is* $SG_\Gamma = (S_{sg}, \Gamma^-, \Gamma^+, s_{sg0}, X_\Gamma)$*, where* $S_{sg}, \Gamma^-, \Gamma^+$ *and* s_{sg0} *are same as normal safety game and* $X_\Gamma \subseteq 2^{\Gamma^- \cup \Gamma^+}$ *is a winning condition. A play on* SG_Γ *is a sequence* $\pi = s_{sg0}, \gamma_0, s_{sg1}, \gamma_1 ...$ *which is also same as normal SG. If plays contain a transition relation in* $x_\Gamma \in X_\Gamma$*, the plays are winning for the environment in* SG_Γ*. Any other plays are winning for the specification.*

We can translate the transition valuation LTKS $E_\Delta = (S, A, \Delta, s_0, P, v_\Delta)$ to the corresponding transition valuation safety game $SG_\Gamma = (S_{sg}, \Gamma^-, \Gamma^+, s_{sg0}, X_\Gamma)$ as below.

- Translating to $S_{sg}, \Gamma^-, \Gamma^+, s_{sg0}$ are same procedure as normal LTKS translation
- X_Γ corresponds to $\{\Gamma_{\neg P} \subseteq 2^{S \times S} | (s, s') \in \Gamma_{\neg P}, s, s' \in S, a \in A, (s, a, s') \in \Delta, v((s, a, s')) = \neg P, \neg P \subseteq P\}$

Results of path valuation function of transition valuation LTKS corresponds to winning or loosing of the game. Therefore, we can conform winning or losing of the play in the transition valuation SG to winning or losing of the play in the normal SG if we can conform the results of path valuation function of transition valuation LTKS to the normal LTKS's one.

We propose a new composition rule that connects a transition to existing states instead of violation states.

Definition 7. *(State Merge Modified Parallel Composition) Given an environment model* $E_\Delta = (S_E, A_E, \Delta_E, s_{0E}, P_E, v_{\Delta E})$ *and tester model* $T_\Delta = (S_T, A_T, \Delta_T, s_{0T}, P_T, v_{\Delta T})$ *such that* $A_T \backslash \{*\} \subseteq A_E$*, the modified parallel composition* $E \parallel'_* T$ *is the transition valuation LTKS* $E \parallel'_* T = (S, A, P, \Delta, v_\Delta, S_0)$*, where* $S \subseteq S_E \times S_T, A = A_E, s_0 = (s_{0E}, s_{0T}), \Delta$ *is the smallest relation that satisfies the rules below and* $v_\Delta(\delta) = \{p \in P | p \in v_{\Delta E}(\delta_E) \cup v_{\Delta T}(\delta_T), \delta_E \in \Delta_E, \delta_T \in \Delta_T\}$ *for* $\delta \in \Delta$ *which satisfies below rules with* δ_E *and* δ_T*.*

$$\frac{s_E \xrightarrow{\ell} s'_E, s_T \xrightarrow{\ell} s'_T}{(s_E,' s_T) \xrightarrow{\ell} (s'_E, s'_T)} \ell \in A_E \cap A_T,$$

$$\frac{s_E \xrightarrow{\ell} s'_E, s_T \xrightarrow{\ell} s'_{T_{\neg p}}}{(s_E, s_T) \xrightarrow{\ell} (s'_E, s_{T_{p*}})} \ell \in A_E \cap A_T,$$

$$\frac{s_E \xrightarrow{\ell} s'_E, s_T \xrightarrow{*} s_T}{(s_E, s_T) \xrightarrow{\ell} (s'_E, s_T)} \ell \in A_E \backslash A_T$$

where $\neg p \notin v((s_T, \ell, s'_T))$*,* $\neg p \in v((s_T, \ell, s'_{T_{\neg p}}))$ *and* $s_{T_{p*}}$ *is an arbitrary state which is not dead-end state*

The composition with multi tester models is defined inductively same as the composition in definition 3.

We confirm that the two conditions mentioned above hold under our composition. In particular, condition 1 holds because the transition relation rules in definition 3 and definition 7 reflects all the transitions in the environment model. This means both analysis spaces have the same paths as the environment model. We can confirm condition 2 by proving the following theorem.

Theorem 1. Given an environment model $E = (S_E, A_E, s_{E_0}, \Delta_E, P_E, v_E)$ and N testers $T_i = (S_{T_i}, A_{T_i}, \Delta_{T_i}, s_{T_{i0}}, P_{T_i}, v_{T_i})$ for $1 \leq i \leq N$, the result of the path valuation function $v_\pi(\pi)$ of $E_{\parallel_*} = E \parallel_* T_1 \parallel_* T_2 ... \parallel_* T_N$ is equal to the result of $v'_\pi(\pi')$ of $E_{\parallel'_*} = E \parallel'_* T_1 \parallel'_* T2 ... \parallel'_* T_N$, which means $v_\pi(\pi) = v'_\pi(\pi')$ if $\pi = \pi'$.

Proof. For all π of $E_{\parallel_*} = (S_{\parallel_*}, A, \Delta_{\parallel_*}, s_{\parallel_* 0} P, v_{\parallel_*})$ and its valuation function v_π, if $\neg p \in v_\pi(\pi)$, π contains $\delta_{\parallel_*} \in \Delta_{\parallel_*}$ such that $\delta_{\parallel_*} = (s_{\parallel_*}, a, s'_{\parallel_*})$ where $s_{\parallel_*}, s'_{\parallel_*} \in S_{\parallel_*}, a \in A$ and $\neg p \notin v_{\parallel_*}(s) \wedge \neg p \in v_{\parallel_*}(s')$. If π contains $\delta_{\parallel_*}$, π' of $E_{\parallel'_*} = (S_{\parallel'_*}, A, \Delta_{\parallel'_*}, s_{\parallel'_* 0} P, v_{\parallel'_*})$ such that $\pi = \pi'$, π' contains $\delta_{\parallel'_*}$ such that $\delta_{\parallel'_*} = (s_{\parallel'_*}, a, s_{\parallel'_*})$ where $s_{\parallel'_*}, s'_{\parallel'_*} \in S_{\parallel'_*}, a \in A$ and $\neg p \in v_{\parallel'_*}(\delta_{\parallel'_*})$. Therefore, $\neg p \in v'_\pi(\pi')$ where v'_π is the path valuation function of $E_{\parallel'_*}$. Similarly, for all π of $E_{\parallel_*}$, if $\neg p \notin v_\pi(\pi)$, π' of $E_{\parallel'_*}$ such that $\pi = \pi'$ does not contain any $\delta_{\parallel'_*} \in \Delta_{\parallel'_*}$ such that $\neg p \in v_{\parallel'_*}(\delta_{\parallel'_*})$. Therefore, $\neg p \notin v'_\pi(\pi')$. From the above, if $\pi = \pi'$, $v_\pi(\pi) = v'_\pi(\pi')$. □

To save space in this paper, the details of our algorithm are given in our repository [1]. Our algorithm generates and merges states according to definition 7. If violation states exist which cannot be merged into any state, the algorithm generates a new state from the states of the environment and tester models.

[1] https://github.com/k-aizawa/AnalysisSpaceReduction

The algorithm resets the violated testers to the initial state when generating such states. The aim here is to minimize the maximum size of the analysis space. A side effect of using initial states instead of violated states is discussed in the next section.

Fig. 4 shows the reduction analysis space generated from models in Fig. 2. There are two violating transitions whose actions are marked. The yellow transition violates property A, and the orange one violates B. Suppose we are given a path $\pi'_1 = 0, wit, 1, requestB, 7, start, 9, processA, 3, wait, 6,$ $requestB, 7, ...$ which is equal to π_1 in previous section. π'_1 violates property A because it has the yellow transitions. On the other hand, π'_1 does not violates property B because it has no orange transition. The results of the path valuation functions in Fig. 3 and Fig.4 are the same. In this example, we can reduce the number of states in the analysis space from 32 to 12.

VI. Evaluation

We evaluated the effect of the proposed reduction. We measured the number of states and memory size of the analysis space.

We compared our proposal with the original analysis space and the space determined from a reachability analyses [10]. Note that, spaces determined from reachability analyses cannot be used for identifying guaranteeable safety properties because they can't hold the condition described in the previous section. Nevertheless, we can examine how closely the analysis space of our technique resembles the one of the reachability analysis.

We created two case studies. One was the simple model of the production cell, shown in Fig. 2, to which we added processing actions (like "processC") and up to 15 corresponding properties and evaluated the size of the generated analysis space. The other was a more realistic model of a production cell [2], in which we modeled the behavior of the robot arm, i.e., how it moves and handles materials. The environment model had 52 states, and we defined ten safety properties for it. The environment model and safety properties are contained in our repository [1]. We added the safety properties in stages and evaluated each stage.

The evaluation used a desktop computer Intel(R) Core(TM) i7-4790K CPU @4.00GHz, 16.0GB RAM and Windows10 Home 64bit OS. We implemented each technique in Java and used Java Object Layout(jol) of openJDK in order to measure the memory size of the analysis spaces. The maximum size of the heap memory was 1GB.

A. Experimental results

The results for the simple case are shown in Fig. 5. The horizontal axis represents the number of safety properties used in generating the analysis space. Bar graphs, whose vertical axis is on the left side, show the number of states. Line graphs, whose vertical axis is on the right side, show the memory size. Note that both vertical axes are logarithmic. For the original space, we stopped measuring at property 11 because the computation ran out of memory. Fig. 5 shows that the memory size increases with the number of states. In the simple case, the added properties are similar to each other and the number of states increases with regularity. The original analysis space increases its states threefold when adding a property. In contrast, both reduction spaces suppress the increase in the state size increase to double. The number of states and memory size are almost the same between the two reduction spaces. This means that there is little overhead to changing the analysis space from a normal LTKS to a transition valuation LTKS.

The results for the realistic case are shown Fig. 6. The realistic case adds various safety properties, and the results shows us some side effects of the reduction technique. State merge reduction, which is our proposal, has more states than original when generating the space from 2 or 3 properties. Moreover, the state size of the reachability analysis sometimes decreases despite adding safety properties; we will discuss these results in the next subsection. Here, we confirm that our proposal can generate an analysis space even when the original space cannot the generated.

Our proposal generates a space in a reasonable time in each case. It took 10.1 s at the longest, whereas using the original space took 27.3 s and using the existing reduction technique took 4.0 s.

B. Discussion

The simple case shows us that our reduction has the same effect as the existing reduction technique even though it holds conditions for identifying guaranteeable safety properties. Our proposal tries to merge the states that are to be removed from the space of the reachability analysis. If all the violation states can be merged to the normal states, the size of our reduction space is the almost same as that of existing reduction.

On the other hand, the realistic case shows the limitations of our proposal. Our proposal is not as effective as in the simple case because it often fails to merge the violation states. Our reduction algorithm generates a new state with resetting the violated tester when it fails to merge the violation state. Resetting the violated testers narrows the maximum state space of the analysis. In contrast, reset testers sometimes monitor actions and increase the states of the analysis space with no meaning. Therefore, our algorithm sometimes generates more states than the original space has.

The realistic case also shows us a side effect of the existing reduction technique. The existing technique sometimes reduces the states of the analysis space in spite of adding safety properties. Strict safety properties lead to violation states in a short while. Thus, many states are removed from the spaces of the reachability analysis. Note that we cannot identify guaranteeable safety properties if we use the existing reduction technique. It only tells us whether the input safety property set can be guaranteed or not. Therefore, analysis spaces must be generated from all subsets of the safety properties. We ran out of memory trying to generate such analysis spaces in the

[1] https://github.com/k-aizawa/AnalysisSpaceReduction

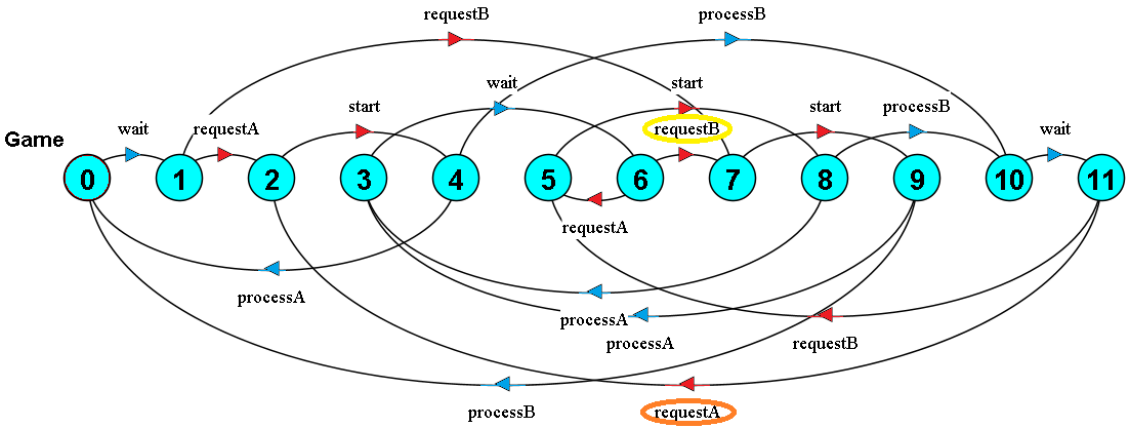

Fig. 4. Example of analysis space with our reduction technique

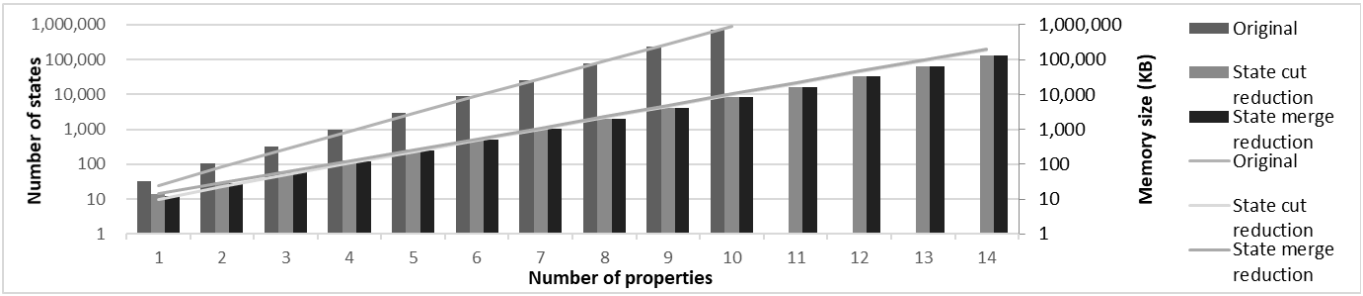

Fig. 5. Memory size of each analysis space (simple example)

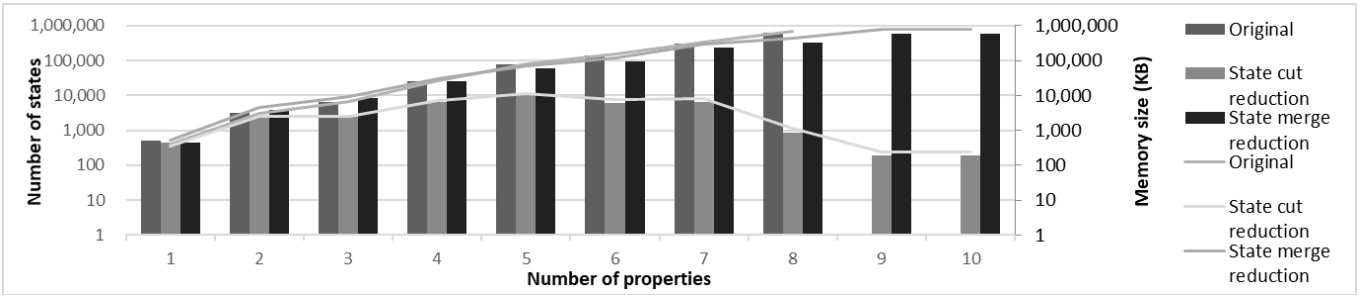

Fig. 6. Memory size of each analysis space (simple example)

simple case with 15 properties and in the realistic case with 10 properties.

VII. Conclusion

Self-adaptive systems require that guaranteeable safety properties be analyzed in changing environments. To do so, all the paths of the environment model have to be reflected in the analysis space. The existing reduction technique used in model checking cannot be used for this because it removes such paths. We proposed another reduction technique that merges the states instead of removing them. We proved that it can hold the conditions for identifying safety properties and confirmed its reduction effect.

In the future, we will optimize the state merging method in order to increase its space reduction effect. A tradeoff exists between the reduction effect and time efficiency. We cannot ignore time efficiency when applying the reduction technique to a self-adaptive system. We will also examine the effect of combining the existing technique with our proposal. So far, we have not considered dependencies between safety properties when generating an analysis space. We can use the existing technique to find those states that can be omitted from the analysis.

References

[1] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distributed Computing*, vol. 2, no. 3, pp. 117–126, 1987.

[2] C. Lewerentz and T. Lindner, Eds., *Formal Development of Reactive Systems - Case Study Production Cell*. London, UK, UK: Springer-Verlag, 1995.

[3] N. D'Ippolito, V. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel, "Hope for the best, prepare for the worst: Multi-tier control for adaptive systems," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, 2014, pp. 688–699.

[4] A. Cailliau and A. van Lamsweerde, "Runtime monitoring and resolution of probabilistic obstacles to system goals," in *2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2017, pp. 1–11.

[5] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, Sep. 2012.

[6] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao, "Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, Aug 2014, pp. 113–122.

[7] J. Cámara, D. Garlan, B. Schmerl, and A. Pandey, "Optimal planning for architecture-based self-adaptation via model checking of stochastic games," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15, 2015, pp. 428–435.

[8] K. Aizawa, K. Tei, and S. Honiden, "Identifying safety properties guaranteed in changed environment at runtime," in *2018 IEEE International Conference on Agents (ICA)*, July 2018, pp. 75–80.

[9] E. Grädel, W. Thomas, and T. Wilke, Eds., *Automata Logics, and Infinite Games: A Guide to Current Research*. New York, NY, USA: Springer-Verlag New York, Inc., 2002.

[10] D. Giannakopoulou and J. Magee, "Fluent model checking for event-based systems," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 5, pp. 257–266, Sep. 2003.

[11] C. Czepa, H. Tran, U. Zdun, T. Tran, E. Weiss, and C. Ruhsam, "Reduction techniques for efficient behavioral model checking in adaptive case management," in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17, 2017, pp. 719–726.

[12] O. Kupferman and M. Y. Vardi, "Module checking," in *Computer Aided Verification*, R. Alur and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 75–86.

[13] S. Kan, Z. Huang, Z. Chen, W. Li, and Y. Huang, "Partial order reduction for checking ltl formulae with the next-time operator," *Journal of Logic and Computation*, vol. 27, no. 4, pp. 1095–1131, 2017.

[14] D. Ciolek, V. Braberman, N. D'Ippolito, and S. Uchitel, "Directed controller synthesis of discrete event systems: Taming composition with heuristics," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec 2016, pp. 4764–4769.

[15] A. Lomuscio and J. Michaliszyn, "An abstraction technique for the verification of multi-agent systems against atl specifications," in *Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning*, ser. KR'14. AAAI Press, 2014, pp. 428–437. [Online]. Available: http://dl.acm.org/citation.cfm?id=3031929.3031981

[16] O. Grumberg, M. Lange, M. Leucker, and S. Shoham, "When not losing is better than winning: Abstraction and refinement for the full μ-calculus," *Information and Computation*, vol. 205, pp. 1130–1148, 08 2007.

[17] O. Hussien and P. Tabuada, "Lazy controller synthesis using three-valued abstractions for safety and reachability specifications," in *2018 IEEE Conference on Decision and Control (CDC)*, Dec 2018, pp. 3567–3572.

[18] E. Burns and R. Zhou, "Parallel model checking using abstraction," in *Proceedings of the 19th International Conference on Model Checking Software*, ser. SPIN'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 172–190. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31759-0_13

[19] E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds., *Handbook of Model Checking*, 2018.

[20] N. R. D'Ippolito, V. Braberman, N. Piterman, and S. Uchitel, "Synthesis of live behaviour models," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10, 2010, pp. 77–86.