Compositional Simulation-Based Analysis of AI-Based Autonomous Systems for Markovian Specifications

Beyazit Yalcinkaya[1(✉)], Hazem Torfah[1], Daniel J. Fremont[2], and Sanjit A. Seshia[1]

[1] University of California, Berkeley, CA, USA
{beyazit,torfah,sseshia}@berkeley.edu
[2] University of California, Santa Cruz, CA, USA
dfremont@ucsc.edu

Abstract. We present a framework for the compositional simulation-based analysis of AI-based autonomous systems for Markovian safety specifications. Our compositional approach allows us to cut down the cost of executing a large number of long-running simulations, by decomposing a simulation-based analysis task into several shorter and more efficient ones. Results obtained from the individual analyses are then stitched together to generate a result for the overall simulation-based task. Our approach is based on a decomposition of scenarios formalized as concurrent hierarchical probabilistic extended state machines that describe sequential and parallel compositions of scenarios. We present two instantiations of our framework for falsification and statistical verification. Using case studies from the autonomous driving domain, we demonstrate the scalability of our compositional approach in comparison to a monolithic analysis approach.

Keywords: Simulation-based analysis · AI-based autonomous systems · Compositional scenarios

1 Introduction

Artificial intelligence (AI) and machine learning (ML) are starting to be used more widely in autonomous systems, in tasks that span perception, prediction, planning, and control. However, there is a growing concern about how to assure the safety of systems that use AI/ML-based components. Formal methods can play a key role in assuring the safety of AI systems [23]. However, due to the high complexity of these components, verification and testing methods must often handle them as black-box components rather than using classic model-based approaches. Simulation-based formal analysis has become commonplace

This work is partially supported by NSF grants 1545126 (VeHICaL, including an NSF-TiH grant) and 1837132, by DARPA contracts FA8750-18-C-0101 (AA), FA8750-20-C-0156 (SDCPS), and FA8750-23-C-0080 (ANSR), by Berkeley Deep Drive, by C3DTI, and by Toyota under the iCyPhy center.

P. Katsaros and L. Nenzi (Eds.): RV 2023, LNCS 14245, pp. 191–212, 2023.
https://doi.org/10.1007/978-3-031-44267-4_10

for assessing the correctness of AI-based autonomous systems. The correctness of the system is evaluated against a specification, defining the safety conditions, by searching through its behaviors in a number of simulations.

Obtaining meaningful and high-confidence verification results requires the execution of a *large number of long-running simulations*, which remains an intensive and costly process. This is a consequence of the high dimensionality of the simulation feature spaces induced by the complex environments in which the systems are executed. In many cases, however, simulation models, also called scenarios, are composed of several smaller scenarios in which a system is tested. For a better and more efficient simulation-based analysis process, we need to take advantage of this composition, transforming a large monolithic analysis process into several easier analysis tasks.

In this paper, we present a compositional approach to simulation-based analysis of autonomous systems. Our approach is based on a decomposition of scenarios where sub-scenarios can be either composed sequentially, representing the different stages of the simulation, or in parallel, representing different possibilities for a stage of the simulation. In this way, a simulation-based analysis problem is decomposed into several smaller problems, each on the sub-scenario level. Results obtained from each sub-analysis problem are stitched together to form a result for the bigger problem. Our framework assumes that the specifications are *Markovian* (memoryless). In practice, most specifications encountered in AI-based autonomous systems are Markovian, e.g., the absence of collisions, obeying traffic lights, keeping a safe distance between agents.

We present a formalization of compositional scenarios based on concurrent hierarchical probabilistic extended state machines, where each of the states of a machine represents one of the sub-scenarios. A scenario is defined in terms of a Markov decision process, representing the agents' behavior and distributions over the feature space of the environment model that the scenario represents. Based on this formalization, we present a framework that can be instantiated to a compositional algorithm for simulation-based analysis tasks of Markovian safety specifications provided there is an aggregation function that allows us to correctly stitch together the individual result to a global one. In general, it is not straightforward to do compositional statistical analysis when the interface specifications are not given. Our formalization and framework define a systematic way to solve this problem via constructing the interface specifications on the fly by computing the post-conditions of the sub-scenarios in each iteration.

We show how we can use our framework to define compositional algorithms for the tasks of falsification and statistical model checking. We evaluate our approach by applying the instantiated algorithms for these problems on benchmarks from the domain of autonomous driving. Our results show that using the compositional algorithms results in a speed up in solving the tasks in comparison to a monolithic approach. In particular, for falsification, the compositional algorithm improves by more than 50% over the monolithic approach in terms of the number of simulation steps needed to find a falsifying example. For statistical verification, in one case study, our approach outperforms the monolithic one using, on average, only half the number of simulation steps. In another case

study, the compositional approach converges after a small number of steps, while the monolithic one exceeds a timeout threshold of 100 simulation runs.

To summarize, our main contributions are: (i) a formalization of compositional scenarios based on concurrent hierarchical probabilistic extended state machines and Markov-decision processes, (ii) a generic framework for the compositional analysis of AI-based autonomous systems, and (iii) experimental evaluation on two instantiations of the framework for falsification and statistical verification, showing its efficacy and scalability in comparison to monolithic simulation-based analysis methods.

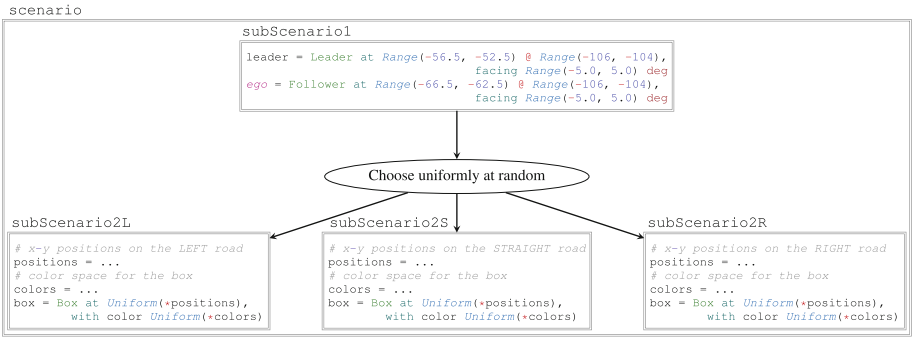


Fig. 1. A snippet of a compositional scenario where individual sub-scenarios are defined using a SCENIC program.

2 Motivating Example

Consider an autonomous driving task with two cars, `Leader` and `Follower`, where the former drives around the city and, at an intersection, turns left, right, or goes straight uniformly at random, and the latter follows `Leader` while keeping a safe distance. We analyze this system using the compositional scenario given in Fig. 1. The individual scenarios are defined using SCENIC [11], a probabilistic scenario-description language, which we will use in our experiments as a way to define scenarios.[1] The program defines a monolithic scenario `scenario` composed of four sub-scenarios, `subScenario1`, `subScenario2L`, `subScenario2S`, and `subScenario2R`. The scenario begins with the sub-scenario `subScenario1`, then composes it sequentially with a uniform choice among `subScenario2L`, `subScenario2S`, and `subScenario2R` to simulate the system in a turn left, go straight, or turn right sub-scenario at an intersection.

`subScenario1` creates the cars, `Leader` and `Follower`, where `Follower` is defined as the *ego* vehicle. Then, it defines a distribution over possible initial position and orientation values for both cars. `subScenario2L` creates a box (an obstacle) after the left turn and defines position and color distributions for it. Similarly, `subScenario2S` and `subScenario2R` do the same for their own cases.

[1] For the full syntax of SCENIC, see [11].

Assume that the controllers of the two cars are equipped with AI-based components for lane keeping and car following. Our goal is to analyze the system composed of the two cars against a system-level specification that requires the two cars to remain within a safe distance from each other that is not larger than 15 m and not smaller than 5. The analysis can be done in two ways, using *falsification* [8], capturing cases where the specification is violated, and *statistical verification* [16], providing statistical guarantees on to what extent the system satisfies the specification. The falsification process searches for counterexamples by simulating the system and sampling initial conditions and actions for the environment. On the other hand, the statistical verification process aims to gather an adequate number of executions of the system in its environment to provide a statistical guarantee for the correctness of the system.

The traditional approach to simulation-based analysis samples initial conditions for the environment and rolls out its entire trajectory. For the scenario in Fig. 1, this translates to interpreting it *monolithically*, i.e., running `scenario`. Simulating a system using `scenario` implies that, for each simulation, Scenic samples a sub-scenario among `subScenario2L`, `subScenario2S`, and `subScenario2R`, composes it with the initial sub-scenario `subScenario1`, and samples conditions for the environment from this composition. Then, we run the simulation with the sampled conditions and continue simulating the system in this way until a termination condition is satisfied. However, this monolithic approach does not exploit the inherent compositional structure of the program. Therefore, it suffers from long simulation runs, requires a large number of executions for statistical guarantees, and does not always provide information about the intermediate behaviors of the system during its execution.

In this paper, we propose to leverage the compositional structure. Our method uses the Scenic program in Fig. 1 at the sub-scenario level. Specifically, both for falsification and statistical verification, we first simulate the system using `subScenario1` until the cars arrive at the intersection and save the post-condition resulting from this execution. We simulate the system in this manner until the saved post-conditions converge to a stable distribution. We refer to this distribution as the *output distribution* of `subScenario1`. We then use this output distribution to analyze `subScenario2L`, `subScenario2S`, and `subScenario2R` by sampling initial conditions for these sub-scenarios. Observe that this approach is inherently more efficient than the monolithic approach as it divides the overall analysis task into smaller ones and solves them in isolation while avoiding redundant computation: e.g., we analyze `subScenario1` only once and reuse its output distribution for the other three sub-scenarios.

3 Preliminaries

3.1 Executions and Specifications

Let S be a system defined over a set of system variables V over the domain of real numbers $\mathbb{R}$. Let $[\![V]\!]$ denote the set of valuations of V. An execution of a

system is a sequence σ of valuations of the variables V, i.e., $\sigma \in [\![V]\!]^*$, where for some alphabet Σ, the set Σ^* defines the set of all finite words over Σ.

For a set of system variables V, we define a specification as a set $\varphi \subseteq [\![V]\!]^*$. A specification thus defines a valid set of executions. In our work, we are particularly interested in Markovian safety specifications, also known as memoryless safety specifications, whose satisfaction on an execution can be determined based on the current valuation of system variables independent of the valuations of the variables in previous steps of the execution. Formally, a Markovian safety specification φ can be defined in terms of a set $\varphi' \subseteq [\![V]\!]$ such that for every execution $\sigma = \sigma_1 \dots \sigma_m \in [\![V]\!]^*$ for any $m \in \mathbb{N}^+$, $\sigma \in \varphi$ if and only if for all $i \leq m$, it holds that $\sigma_i \in \varphi'$. In the rest of the paper, a specification will always refer to a Markovian safety specification.

3.2 Markov Decision Processes

For a countable set X, let $\mathit{Distr}(X) \subset (X \to [0,1])$ define the set of all distributions over X, i.e., for $d \in \mathit{Distr}(X)$ it holds that $\Sigma_{x \in X} d(x) = 1$. For $d \in \mathit{Distr}(X)$, let the *support* of d be defined by $\mathit{Supp}(d) = \{x \mid d(x) > 0\}$. A *Markov Decision Process (MDP)* is a tuple $M = (Q, \mathit{Act}, \mathbf{P}, \iota, X, L)$ where Q is a set of states, Act is a finite set of actions, $\mathbf{P} : Q \times \mathit{Act} \to \mathit{Distr}(Q)$ is the transition probability function such that $\sum_{q' \in Q} \mathbf{P}(q,a)(q') \in \{0,1\}$ for every $q \in Q$ and $a \in \mathit{Act}$, $\iota \in \mathit{Distr}(Q)$ is the initial distribution such that $\sum_{q \in Q} \iota(q) = 1$, X is a finite set of variables and $L : S \to [\![X]\!]$ is a labeling function that assigns each state a valuation of the variables in X. For a state $q \in Q$, we define $\mathsf{AvAct}(q) = \{a \mid \mathbf{P}(q,a) \neq \bot\}$, where $\bot$ is the empty distribution. W.l.o.g., $|\mathsf{AvAct}(q)| \geq 1$. If $|\mathsf{AvAct}(q)| = 1$ for all $q \in Q$, we refer to M as a *Markov chain* (MC). A *finite path* in the MDP M is a sequence $\pi = q_0 a_0 q_1 \dots q_n \in Q \times \left(\mathit{Act} \times Q\right)^*$ such that for every $0 \leq i < n$ it holds that $\mathbf{P}(q_i, a_i)(q_{i+1}) > 0$ and $\iota(q_0) > 0$. We denote the set of finite paths of M by Π_M. We use $\pi_\downarrow$ to denote the last state in π. A *policy* for the MDP M is a function $\sigma : \Pi_M \to \mathit{Distr}(\mathit{Act})$ with $\mathit{Supp}(\sigma(\pi)) \subseteq \mathsf{AvAct}(\pi_\downarrow)$ for every $\pi \in \Pi_M$. A policy σ of M induces a Markov chain $[\![M]\!]_\sigma$. We denote the set of policies of M by $\mathit{Policies}(M)$.

4 Compositional Scenarios

A scenario represents a model of an environment in which we want to deploy and analyze a system. This model is defined as a distribution over spatial and temporal configurations of the environment, including those of all objects and agents. The underlying semantics of a scenario can therefore be defined by an MDP, where the nondeterministic actions capture the nondeterministic behavior of the agents and simulator, and the probabilistic behavior reflects the distributions over the feature space and probabilistic actions of agents. For example, each of the SCENIC sub-scenarios in Fig. 1, defines an MDP. A compositional scenario is a collection of scenarios that are composed sequentially and in parallel.

A compositional scenario is therefore best modeled as a concurrent hierarchical probabilistic extended state machine (CHPESM). Concurrent, because scenarios can be executed concurrently; hierarchical, because a compositional scenario can be composed of other compositional scenarios; probabilistic because switching to the next scenario can be done probabilisticly and, also, at any time the choice of attributes and actions is done based on an underlying probability distribution over valuations of the environment's feature space. The semantics of a compositional scenario is then defined by an infinite-state MDP. In the following sections, we give an overview of the CHPESM model and show how the MDP of a compositional scenario can be obtained by computing the flattening of its CHPESM.

4.1 Concurrent Hierarchical Probabilistic Extended State Machines

We define a probabilistic extended state machine (PESM) as a tuple $M = (Q, \iota, V, F, P)$ where Q is a set of states, $\iota \in \mathit{Distr}(Q)$ is an initial distribution over states, V is a finite set of real-valued variables, F is a set of exit states, and $P : Q \times G(V) \to \mathit{Distr}(Q)$ is a probabilistic transition relation, where $G(V)$ defines the set of boolean constraints of the form $x \sim c$, so-called guards, for $x \in V$ and $\sim \in \{<, \leq, =, >, \geq\}$. Given a state and a guard, P returns a distribution over states. A transition $(q, g, d) \in P$ is enabled for a valuation $v \in [\![V]\!]$ if and only if $g(v)$ is true.

Hierarchical probabilistic extended state machines (HPESMs) are probabilistic extended state machines whose states are themselves PESMs or HPESMs [20,25]. Formally, HPESMs are defined inductively as follows. Let $\mathcal{M}$ be a set of HPESMs over variables V. A HPESM $H = (Q, \iota, V, F, P, \mathcal{M}, \mu)$, where Q, ι, V, F and P are a set of states, an initial distribution over states, a set of variables, a set of exit states, and a probabilistic transition relation as in any probabilistic extended state machine. Further, $\mu : Q \to \mathcal{M}$ is a mapping that associates each state $q \in Q$ with an HPESM from $\mathcal{M}$. For example, in Fig. 1, `scenario` is an HPESM with one state mapped to a PESM with states `subScenario2L`, `subScenario2S`, and `subScenario2R`, and `subScenario1`.

An HPESM H provides a compact notation for a corresponding PESM, denoted by *flat(H)* [25]. The machine *flat(H)* is defined inductively as follows. If H is a PESM then $\mathit{flat}(H) = H$. If H is not a PESM, then $\mathit{flat}(H) = (Q', \iota', V, F', P')$, defined as follows. $Q' = \bigcup_{q \in Q} \mathit{states}(\mathit{flat}(\mu(q)))$, with $\mathit{states}(.)$ being a function that returns the set of states of any PESM. For all $q \in Q$ and for all $q' \in \mathit{states}(\mathit{flat}(\mu(q)))$, $\iota'(q') = \iota(q).\mathit{init}(\mathit{flat}(\mu(q)))(q')$ with *init* returning the initial distribution of a state machine. $P' = \{\tau = (q_1, g, d) \mid \exists q \in Q.\, \tau \in \mathit{transitions}(\mathit{flat}(q))$ *or* $\exists (q, g', d') \in P.\, q_1 \in \mathit{exit}(\mathit{flat}(\mu(q))) \wedge \forall q' \in Q. \forall q_2 \in \mathit{states}(\mathit{flat}(\mu(q'))).\, d(q_2) = d'(q') \cdot \mathit{init}(\mathit{flat}(\mu(q')))\,(q_2)\}$, with *transitions* and *exit* functions returning the transition relation and exit states.

A concurrent hierarchical probabilistic extended finite state machine (CHPESM) [25] is defined inductively as follows. Every PESM is a CHPESM. For the inductive step, we distinguish two cases: (1) a CHPESM C is either a hierarchical composition of other CHPESMs, as previously defined for HPESMs,

or (2) it is a parallel composition of CHPESMs, i.e., $C = C_1 \parallel \cdots \parallel C_n$ for CHPESMs $C_1, \ldots, C_n$.

Every CHPESM C is associated with a corresponding PESM defined again inductively as follows. If C is a basic PESM, then $\mathit{flat}(C) = C$. If C is a hierarchical composition then $\mathit{flat}(C)$ is defined as in the case of HPESMs. If $C = C_1 \parallel \cdots \parallel C_n$ for CHPESMs $C_1, \ldots C_n$ over a set of variables V, then $\mathit{flat}(C) = (Q, \iota, V, F, P)$ where $Q = \mathit{states}(\mathit{flat}(C_1)) \times \ldots \mathit{states}(\mathit{flat}(C_n))$, $\iota = \mathit{init}(\mathit{flat}(C_n)) \cdot \cdots \cdot \mathit{init}(\mathit{flat}(C_n)))$, $F = \mathit{exit}(\mathit{flat}(C_n)) \times \cdots \times \mathit{exit}(\mathit{flat}(C_n))$, and P a probabilistic transition relation where $((q_1, \ldots, q_n), \bigwedge_{i \leq n} g_i, d) \in P$ iff there is a transition $(q_i, g_i, d_i') \in \mathit{transitions}(\mathit{flat}(C_i))$ for some $i \in \{1, \ldots, n\}$ and where $d((q_1', \ldots, q_n')) = \Pi_{j \text{ s.t. } q_j \neq q_j'} d_j(q_j')$. We note that we choose a product based on self-composition, i.e., transitions represent the execution actions from different CHPESMs that can execute in parallel, in contrast to the usual synchronous product used in [25]. This is necessary as individual processes in our setting may evolve at their own pace.

4.2 Abstract Syntax and Semantics of Compositional Scenarios

Let $\mathcal{S}$ be a set of compositional scenarios. A compositional scenario is abstractly defined as a CHPESM $\mathcal{P} = (S, \iota, V, F, T, \mathcal{S}, \mu)$. Each state $s \in S$ represents a compositional scenario from $\mathcal{S}$. Each of the basic (i.e., non-decomposable) scenarios $s \in \mathcal{S}$ is defined over the feature space V.

The semantics of a compositional scenario $\mathcal{P}$ is defined by a (infinite-state) Markov decision process $[\![\mathcal{P}]\!] = (Q, \mathit{Act}, \mathbf{P}, \iota, X, L)$, obtained by first computing the flattening of the CHPESM and then refining the states of the resulting PESM to the MDPs they represent. A concrete scenario of $\mathcal{P}$ is a pair (v, π) for some $v \in [\![V]\!]$ and $\pi \in \mathit{Policies}([\![\mathcal{P}]\!])$, inducing a Markov chain $[\![\mathcal{P}]\!]_{v,\pi} = (Q, \mathbf{P}', \iota', X, L)$ where $\iota'(q) = 1$ if $q = (v, -)$ and $\iota(q) > 0$. For any other $q' \in Q \setminus \{q\}, \iota'(q') = 0$. An execution of $\mathcal{P}$ is a path of a Markov chain $[\![\mathcal{P}]\!]_{v,\pi}$ obtained for an initial sampled valuation $v \in [\![V]\!]$ and a policy $\pi \in \mathit{Policies}([\![\mathcal{P}]\!])$.

5 Compositional Simulation-Based Analysis

In this section, we present a generic compositional simulation-based analysis framework that can be instantiated to concrete algorithms for solving specific simulation-based analysis tasks. The framework assumes a compositional scenario structure, given by a CHPESM. Working on the PESM defining the flattening of the CHPESM, the algorithm decomposes a general simulation-based analysis task into smaller tasks, one for each state of the PESM, thus, executing analysis tasks on the sub-scenario level. Results obtained from the individual simulation-based analyses are stitched together providing a result for the overall analysis task. We first introduce the generic framework and then show two instantiations of the algorithm for falsification and statistical verification.

5.1 Generic Framework

We introduce a generic compositional simulation-based analysis framework in Algorithm 1. An instantiation of the framework is an algorithm that operates on an PESM $M = (S, \iota, V, F, P)$, defining a flattening of a compositional scenario. Such an instantiation is obtained by implementing three key procedures: *evaluate*, *terminate*, and *finalize*. The algorithm then iterates over all sub-scenarios defined by the states of M, starting with an initial state that is in the support of ι and following the transition relation P. For each sub-scenario, an algorithm executes a simulation-based analysis task implemented by the procedure *evaluate*. This process is repeated until a termination condition is satisfied, checked by the procedure *terminate*. Termination is decided based on the change in the outcomes of the evaluation processes at each sub-scenario. If the termination condition is satisfied, then the procedure *finalize* is called to compute a final result. This result is computed by stitching together the results computed by the evaluation process for the sub-scenario. If the termination condition is not satisfied, the algorithm continues by evaluating other sub-scenarios. In the following, we elaborate on the workflow of the framework by providing more details on the functionalities of procedures *evaluate*, *terminate*, and *finalize*.

Evaluation. An instantiation of our compositional approach executes simulation-based analysis tasks at the sub-scenario level starting with states from the initial distribution ι (line 2). The set W represents a working set including all sub-scenarios for which a task should be executed next. For each sub-scenario $s \in W$ (line 4), the procedure *evaluate* is invoked (line 5), which implements a specific simulation-based analysis process for a given task of interest (e.g., falsification, statistical verification, etc.). The *evaluate* procedure is provided with an input distribution d^s_{in} on the input sample space of s, and output distribution d^s_{out} on the output space of s, which is to be updated by the evaluation procedure. We represent d^s_{in} and d^s_{out} as multisets, so the distributions are extended by adding new elements to these multisets. Notice that other representations for these distributions are also possible depending on the specific needs of the task under consideration. Finally, while termination of *evaluate* may depend on the simulation-based analysis task at hand, it can also be stopped after a simulation budget c has been exhausted, i.e., a number of simulation steps c is reached as a sum of steps taken over all simulation runs performed by *evaluate*. The budget can also be unlimited, i.e., $c = -$. In this case, the termination of *evaluate* solely depends on the termination condition of the analysis task given by the underlying implementation of *evaluate*.

Algorithm 1. Compositional Simulation-based Analysis

Input:
Probabilistic state machine $M = (S, \iota, V, F, P)$ /* Flattening of a CHPESM*/
Local simulation budget $c \in \mathbb{N} \cup \{-\}$
1: *initialize*$(\{d^s_{in}\}_{s\in S}, \{d^s_{out}\}_{s\in S}, r, \{D^s_{out}\}_{s\in S}, R)$
2: $W := Supp(\iota)$, $W' := \emptyset$
3: **while** *True* **do**
4: **for** $s \in W$ **do**
5: $d^s_{out}, r(s) := evaluate(s, d^s_{out}, d^s_{in}, c)$
6: $R(s) := append(R(s), r(s))$
7: $D^s_{out} := append(D^s_{out}, d^s_{out})$
8: **if** $terminate(\{D^s_{out}\}_{s\in S}, R)$ **then**
9: **return** $finalize(M, r)$
10: **for** $(s, g, d) \in P$ **do**
11: **for** $s' \in Supp(d)$ **do**
12: $W' := W' \cup \{s'\}$
13: $d^{s'}_{in} := d^{s'}_{in} + d^s_{out}$
14: **if** $W' \neq \emptyset$ **then**
15: $W := W'$
16: $W' := \emptyset$
17: **else**
18: $W = Supp(\iota)$

The outcome of *evaluate* is a distribution d^s_{out} on the outputs reached by the simulation process of *evaluate*, and a result of the analysis process that updates a mapping r for the specific sub-scenario s. For example, if *evaluate* implements a falsification task for some specification φ, then $r(s)$ is assigned to a valuation $v \in [\![V]\!]$, and a policy $\pi \in Policies(s)$, that falsify φ. If *evaluate* implements a statistical verification method, then $r(s)$ stores an estimate of the probability of satisfying a given specification in s. The history of results obtained for each sub-scenario, as well as the history of output distributions are stored in a list of mappings R, and a list of distributions D^s_{out} for each sub-scenario $s \in S$ (lines 6 and 7). For this we assume that the input and output distributions d^s_{in}, and d^s_{out}, the mapping r, as well as the lists D^s_{out} and R are initialized at the beginning of the algorithm (line 1). These lists are necessary for checking termination and are forwarded to the procedure *terminate*.

Termination. Termination of Algorithm 1 is checked after every evaluation process, and is done executing an implementation of the procedure *terminate* (line 8). Termination is decided based on the history of results stored in R, and obtained using *evaluate* at each sub-scenario, as well as the history of output distributions D^s_{out} for each sub-scenario s (collected in lines 7 and 6). For example, in the case of falsification, *terminate* is implemented as the procedure that returns *True* if a falsifying valuation is found at any sub-scenario s, or, if such valuation is not found, then it might return *True* after observing a stabilization in the output distribution computed for each sub-scenario. The latter stabilization condition can also be used as a termination condition for statistical verification (More details in the next sections). If *terminate* returns *False*, then Algorithm 1 continues by applying the *evaluate* procedure on other scenarios in the working set W that have not been processed so far. As long as termination is not satisfied, after each evaluation process, a new working set of sub-scenarios

is computed that will be processed in the next iteration of the algorithm (lines 10 - 13). Here, successor sub-scenarios s' of a currently evaluated sub-scenario s, following the transition relation P, are added to the new working set W'. Furthermore, the input distribution $d^{s'}_{in}$ of a state s' is updated with the output distribution d^{s}_{out} of its predecessor s (line 13). Once all states in W have been processed, and the algorithm has not terminated, the set W is replaced with the set W' (line 15) and the evaluation process is repeated for the new working set. In case no new sub-scenarios are added, i.e., we reached and processed all exist scenarios F of M, and the termination condition has not been satisfied yet, the evaluation is restarted by re-initializing the set W with the set of initial states (line 18). When it finally comes to a termination of the algorithm, as a last step, the procedure *finalize* computes a final result for the overall simulation-based analysis task, using the results stored in r. We discuss some instantiations of the *finalize* procedure next.

Finalization. If the termination condition defined by the procedure *terminate* is satisfied, a last procedure *finalize* is applied on the PESM M and the computed mapping r. Executing an implementation of *finalize* stitches the results stored in r, which were obtained from evaluating each sub-scenario, to a general result for the general simulation-based analysis task. For example, in the case of a falsification task for a specification φ, *finalize* will return a valuation satisfying the input distributions of one of initial scenarios $s_0 \in \mathit{Supp}(\iota)$, and for which there is policy that leads to falsifying φ at s_0, or a later scenario, reachable via P from s_0. In the case of statistical verification, a satisfaction probability is computed by computing the minimum/maximum reachability probability computed over all probabilities and distributions computed for the sub-scenarios and with respect to the transition relation of M.

In the rest of the paper we refer to Algorithm 1 as the procedure *comp*. An instantiation of Algorithm 1, for an evaluation, termination, and finalization procedures λ_{ev}, λ_{ter}, and λ_{fin}, respectively, is denoted by $\mathit{comp}(\lambda_{ev}, \lambda_{ter}, \lambda_{fin})$.

5.2 Compositional Simulation-Based Falsification

In falsification we are interested in finding an evaluation of the feature space that falsifies a given property. Given a flattening of a compositional scenario $M = (S, \iota, V, F, P)$ and a system-level specification $\varphi \subseteq [\![V]\!]^*$, find $v \in [\![V]\!]$ and $\pi \in \mathit{Policies}([\![M]\!])$ such that $[\![M]\!]_{v,\pi} \not\models \varphi$. In this section, we show how we can define a compositional falsification approach by instantiating the procedures *evaluate*, *terminate*, and *finalize*.

- *evaluate*: we instantiate *evaluate* with a procedure λ^{φ}_{ev} that for a given scenario s, an input distribution d^{s}_{in}, output distribution d^{s}_{out}, and a simulation budget c, simulates s sampling initial inputs from the distribution d^{s}_{in}, and evaluates a simulation run based on a function $\lambda : 2^{[\![V]\!]^*} \times 2^{[\![V]\!]^*} \to \mathbb{B}$, that for a specification $\varphi \subseteq [\![V]\!]^*$ and a set of simulation runs, returns *True* if and only if a simulation run τ is a falsifying example for φ. i.e., $\tau \not\models \varphi$. Simulations

are restarted as long as no falsifying example is found or until the simulation budget is exhausted. If a falsifying examples is detected, the falsification process returns the initial valuation of inputs sampled at that simulation and a simulation trace.
- *terminate*: termination is implemented by λ_{ter} returning *True* once a falsifying example is found at any sub-scenario. If no such example is found, the process terminates using the stabilization condition defined over the list of output distributions.
- *finalize*: The finalization procedure is implemented by λ_{fin} that chooses an initial valuation of the feature space and a path in M such that it leads to a non-empty r in one of the scenarios reachable via the path if they exist, otherwise it returns *False*.

Theorem 1. *For a compositional scenario M, a specification φ, and a falsification method λ, it is the case that $comp(\lambda_{ev}^{\varphi}, \lambda_{ter}, \lambda_{fin})(M, c) = \lambda(\varphi, M)$.*

Proof (Sketch). The correctness of this instantiation follows from the fact that once a sub-scenario has been falsified, then there is a valuation from its input distribution and a policy, inducing traces falsifying the specification. Since the input distribution is the union of output distributions of predecessor sub-scenarios, then we can find a valuation from the input distribution of that scenario and a policy that lead to the violation. We can extend this argument to reach a valuation from one of the initial sub-scenarios and build a policy that induces a trace, violating the specification. □

5.3 Compositional Simulation-Based Statistical Verification

Statistical verification is a method that allows estimating the correctness of a system for a given property by simulating the system for a number of runs and using methods from the area of statistical theory to provide guarantees on the correctness up to a statistical error [16]. We show that for a given statistical verification method, we can instantiate Algorithm 1 to an algorithm that performs the statistical verification compositionally, preserving the guarantees obtained by the statistical verification method.

Let $M = (S, \iota, V, F, P)$ be the flattening of a compositional scenario, $\varphi \subseteq [\![V]\!]^*$ a specification we are interested in verifying, and λ a statistical verification procedure that for M and φ, estimates the probability of M satisfying φ up to a statistical error. A compositional algorithm for solving the statistical verification problem can be achieved by using the following implementations of *evaluate*, *terminate*, and *finalize*.

- *evaluate*: we implement *evaluate* as a procedure λ_{ev}^{φ} that applies λ at each sub-scenario $s \in S$ for the specification φ. Simulation runs created by λ_{ev}^{φ} start from the input distribution d_{in}^{s}, using a simulation budget c.
- *terminate*: a termination procedure λ_{ter} can be implemented in several ways [16]. In general, the termination condition for a statistical verification process λ can be applied also over the sequence of results and post-conditions

computed by the individual statistical verification processes. One prominent example that we will use in our experiments, is a stabilization condition based on the convergence of the standard error of the mean of the simulation post-conditions [13].

– *finalize*: The finalization procedure λ_{fin} computes a probability based on the probabilities computed for each sub-scenario. It aggregates the probabilities starting from the exit states to the initial states. Specifically, the probability for a sub-scenario s, is computed as $f_{(s,g,d)\in P} \sum_{s'\in S} r(s').d(s')$ where f is an aggregation over probabilities (e.g., max, min, etc.).

Theorem 2. *For a compositional scenario M, a specification φ, and an statistical verification method λ, it is the case that* $comp(\lambda^{\varphi}_{ev}, \lambda_{ter}, \lambda_{fin})(M, c) = \lambda(\varphi, M)$.

Proof (Sketch). The correctness follows from the fact that the statistical verification processes have been *performed independently* and based on independent output distributions computed at each sub-scenario. The overall result is thus statistically correct up to the same statistical error for each sub-scenario. □

6 Experimental Evaluation[2]

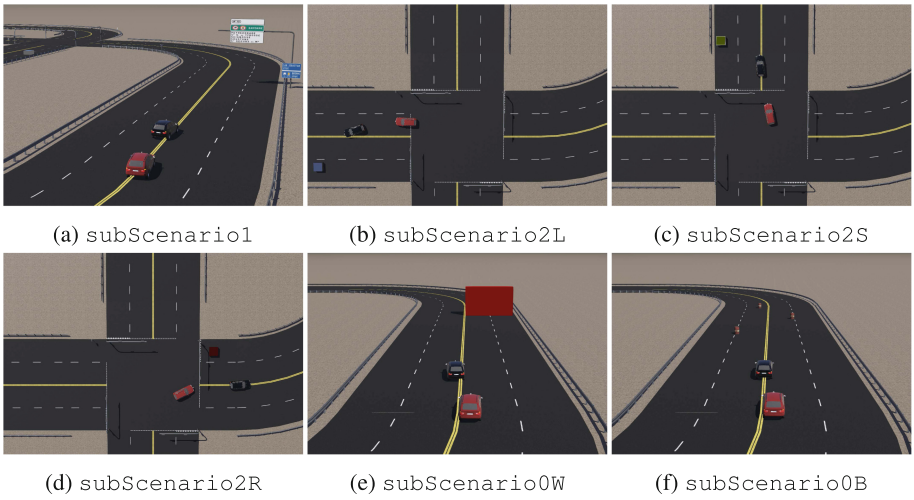
In this section, we present an experimental evaluation of the proposed method on two case studies from the autonomous driving domain. We first provide a high-level description of the autonomous driving tasks. Then, we present the details of the simulator setup, the controller implementations, and the system-level specification. We also explain the evaluation metrics and our baseline. Finally, we present compositional scenarios used for evaluation, details of their feature spaces, and the evaluation results.

6.1 An Autonomous Driving Task

The environment consists of several straight and curved road segments along with two intersections and obstacles (see Fig. 2). We have two cars: `Leader` (the black car) and `Follower` (the red car). They are tasked with following each other while maintaining a safe distance. `Leader` follows the yellow line in the middle of the road. At an intersection, it chooses to go left, straight, or right uniformly at random. After an intersection, `Leader` continues to follow the yellow line all while trying to avoid obstacles. `Follower` must follow the lead car while keeping a safe distance.

We use the Webots, an open-source 3D robot simulator widely used in industry, education, and research [17,24]. Both cars are modeled as a BMW X5 equipped with a camera facing the road. Using its camera input, `Leader`'s computer vision component uses a standard image processing technique to estimate the car's angle to the yellow line. This estimate is then used as an input to a

[2] Available at https://github.com/BerkeleyLearnVerify/compositional-analysis.



(a) `subScenario1` (b) `subScenario2L` (c) `subScenario2S`

(d) `subScenario2R` (e) `subScenario0W` (f) `subScenario0B`

Fig. 2. Snapshots of the sub-scenarios of both case studies.

PID controller which controls the steering. `Follower` estimates its angle to the `Leader` by processing its camera inputs in a similar way, and it also performs image segmentation for estimating the distance to `Leader`. Additionally, `Leader` uses a Sick LMS 291 LiDAR sensor for collision avoidance. Both cars set a target speed of 40 km/h, but `Follower` changes its speed by braking or speeding up to maintain a safe distance from `Leader`. The system-level specification formalizes the safe and specified distance between the cars with the Metric Temporal Logic (MTL) [14] property $\square\,(\texttt{distance} \geq 5 \wedge \texttt{distance} \leq 15)$, where `distance` denotes the distance between the cars. Specifically, the property defines the notion of safety as keeping a distance between the cars that is not less than 5 m and not more than 15 m throughout the entire trajectory.

6.2 Evaluation Details

We evaluate our method by performing both compositional falsification and compositional statistical verification on two different compositional scenarios for the presented task. In compositional falsification, *terminate* (see line 8 in Algorithm 1) is designed to terminate the counter-example search either as soon as a falsifying example is found or until the output distributions of each sub-scenario *stabilizes* (the notion of stabilization will be defined precisely). In compositional statistical verification, the *terminate* method halts the process once the output distribution of each sub-scenario stabilizes. The notion of stabilization for each sub-scenario's output distribution is defined by the *convergence of the standard error of the mean (SEM) of the simulation post-conditions*. Specifically, at the end of each sub-scenario, we get a point in a 6D space consisting of the x-y coordinates and the orientations of both `Leader` and `Follower`. The post-conditions of sub-scenarios form a distribution, which we try to approximate

through simulations. We stop generating more samples for a sub-scenario once the change in the SEM of the post-conditions drops down a threshold value Δ, i.e., the SEM converges to a stable value. The SEM $\sigma_{\bar{\mu}}$ is defined as $\sigma_{\bar{\mu}} = \frac{\bar{\sigma}}{\sqrt{n}}$, where $\bar{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{\mu})^2}$ is the unbiased estimator of standard deviation, $\{x_1, x_2 \ldots, x_n\}$ is the set of n samples, and $\bar{\mu}$ is the sample mean. The stability of the SEM can be used as an indicator of a robust and reliable estimate of the population mean [13]. One can also calculate confidence intervals for the true population mean using SEM, e.g., a 95% confidence interval can be calculated as $\bar{\mu} \pm 1.96 \times \sigma_{\bar{\mu}}$.

A nuance between the theory and the practical implementation of the proposed method is that for compositional falsification, to find counter-examples earlier, we implement our method in *batched* mode. Specifically, given a batch size, we run each sub-scenario in batches and interleave the falsification process of each sub-scenario until either all sub-scenarios satisfy their convergence condition or a counter-example is found. This way we avoid redundantly waiting for the convergence of the output distributions of sub-scenarios that cannot be falsified to find counter-examples faster.

In our evaluation, we use two different sampling strategies, uniform sampling and Halton sampling [12], for both compositional falsification and compositional statistical verification. We leverage VerifAI [8,9], a toolkit for the formal design and analysis of AI/ML systems, to sample scenes from Scenic programs according to uniform and Halton sampling. These sampling strategies are used by the *evaluate* method (see line 5 of Algorithm 1) for sampling initial conditions to simulate trajectories. Observe that both uniform and Halton sampling are *passive* sampling strategies. The usage of passive sampling strategies provides a simpler implementation for the batched execution of compositional falsification since we initialize a new sampler for each batch. One can use active sampling strategies like cross-entropy, simulated annealing, Bayesian optimization, etc. by saving the sampler state so that the sampler state would be preserved between the interleavings of different falsification processes. Another way to use active samplers is to run the falsification process of each sub-scenario either concurrently or in parallel instead of running them in batched mode. However, these implementations are outside the scope of this work as they require diligent engineering efforts.

Baseline. The baseline for our evaluation is the *monolithic* simulation-based analysis approach that is currently supported by Scenic and VerifAI. Specifically, this approach treats the Scenic program as a black-box sampler and does not leverage the compositional structure of the program. It samples initial conditions for the entire scenario and rolls out the entire trajectory until the end. The termination condition for the baseline is defined in a similar way to our method, i.e., for statistical falsification, the search ends either when a counter-example is found or when the output distribution of the entire Scenic program stabilizes, and for statistical verification, we run simulations, again, until the output distribution stabilizes.

Metrics for Evaluation. To demonstrate the efficacy of our method compared to the monolithic baseline, we focus on the total number of simulator steps and the estimated specification satisfaction probability. For falsification, we analyze the number of simulator steps taken until the falsification process ends. Due to the inherent randomness of both statistical methods, we run the falsification process with 10 different random seeds and analyze the mean and the standard deviation of the number of simulator steps. For verification, we again focus on the number of simulator steps taken until convergence, and, we compare the estimated specification satisfaction probability of both methods.

6.3 Case Study 1

We use the compositional SCENIC program from Fig. 1 to test this system against the system-level specification. The program defines four sub-scenarios: `subScenario1`, `subScenario2L`, `subScenario2S`, and `subScenario2R`. `subScenario1` is a sub-scenario starting at the straight road segment and ending at the intersection (see Fig. 2a). This sub-scenario defines possible initial positions and orientations for both cars. Once `subScenario1` is over, the next sub-scenario is sampled uniformly at random from the other three. `subScenario2L` defines the sub-scenario where `Leader` turns left (see Fig. 2b), `subScenario2S` is the sub-scenario for going straight at the intersection (see Fig. 2c), and `subScenario2R` is the case for turning right (see Fig. 2d). All sub-scenarios at the intersection have an adversarial obstacle to trick the image processing and segmentation performed by `Follower`. Specifically, they all sample positions and colors for a box that can potentially cause `Follower` to violate the safety specification by mixing `Leader` with the obstacle. The space of possible positions for the obstacle, in each sub-scenario, is defined to be either on the right-most or the left-most lanes so that the yellow line and the inner lanes are not blocked for `Leader`. However, if the `Leader` takes the turn too wide, it could still collide with the obstacle, which would cause a specification violation. The color space for the obstacle consists of nine different colors, only one of which can fool `Follower`. If the obstacle is black and visible from `Follower`, it could corrupt the angle and the distance estimates and therefore potentially cause a specification violation. Notice that in our implementation, we manually decomposed sub-scenario definitions of the monolithic Scenic program since Scenic's current Webots interface does not allow the usage of sub-scenarios.

Falsification. To understand the performance gains of our method compared to the monolithic baseline, we run both our method and the monolithic baseline with 10 different random seeds. We run our method in batched mode with a batch size of 5. Figure 3a presents the results of the experiments for both uniform and Halton sampling strategies. Both methods find counter-examples before converging to a stable distribution. However, the compositional method finds counter-examples by taking fewer simulator steps on average. Moreover, the standard deviation of the total number of simulator steps is much smaller compared to the monolithic baseline. An important detail to note here is that when we compare the sampling strategies, we see that Halton finds counter-examples

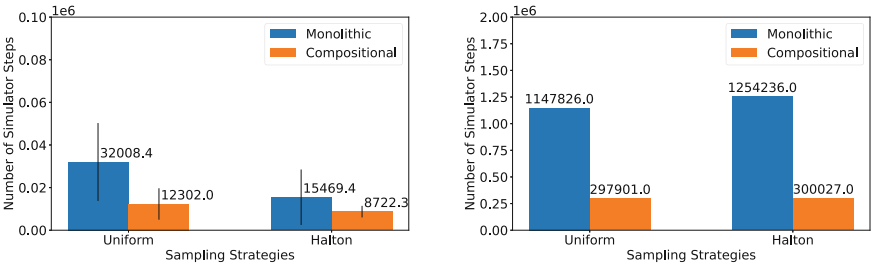
earlier compared to the uniform sampling strategy. This result is aligned with the intuition that Halton sampling provides a more uniform coverage compared to uniform sampling which uses pseudorandom number generators.

Statistical Verification. For comparing the performances of both methods for statistical verification, we run both until convergence, i.e., until their output distributions stabilize w.r.t. to the stabilization metric given in Sect. 6.2. For this experiment, we set the threshold for convergence to $\Delta = 0.001$ and run both methods until the change in their SEM values drops down to this threshold. Figure 3b presents a comparison between the total number of simulator steps taken by each method before converging to a stable output distribution. With uniform sampling, the compositional method provides a 3.85× speed up, and with Halton sampling, our method is 4.18× faster. Notice that with Halton sampling, both methods take slightly more simulator steps compared to the uniform sampling strategy. This is due to the fact that the coverage provided by Halton sampling makes the convergence of the output distributions harder.

We also compare the estimated probabilities output by the methods. Table 1 presents these results for both sampling strategies. The probability for the compositional method is calculated by combining the results from sub-scenarios. Since we sample among `subScenario2L`, `subScenario2S`, and `subScenario2R` uniformly at random, their probabilities are averaged, and since `subScenario1` precedes the other three, its estimated probability is multiplied by the calculated average. Table 1 shows that both methods converge to similar specification satisfaction probabilities with minor differences. However, the compositional approach uses fewer simulations to converge to this value. Moreover, the average simulation length (i.e., the average number of simulator steps) taken by our method is significantly less than the monolithic approach. For the compositional method, the average simulation length is calculated by summing the number of simulation steps of all sub-scenarios and dividing it by the total number of simulation runs, i.e., 383. Notice that compared to the monolithic approach, our compositional method provides more information about the safety of the system in different sub-scenarios. For example, we see that the system does not violate the system-level specification in `subScenario1`, along with the individual specification satisfaction probabilities for each sub-scenario, whereas the monolithic approach does not provide any insight into the system behavior across different sub-scenarios. We conclude by noting that on a Quad-Core Intel i7 processor clocked at 2.3 GHz and a 32 GB main memory, our compositional statistical verification method took a little over 2 h to converge for both uniform and Halton sampling, whereas the monolithic baseline took 6 h for uniform sampling and a little over 8 h for Halton sampling to converge.

6.4 Case Study 2

We build on top of the first case study by adding two uniformly sampled subscenarios before the first sub-scenario: `subScenario0W` and `subScenario0B`. Both of these sub-scenarios start at the straight road segment connecting to



(a) Falsification (b) Statistical Verification

Fig. 3. Experiment results for falsification and statistical verification of Case Study 1.

Table 1. Estimated specification satisfaction probabilities.

	Uniform Sampling			Halton Sampling		
	Estimated Probability	Number of Sim	Mean Sim Length	Estimated Probability	Number of Sim	Mean Sim Length
`subScenario1`	1.00	76	1100.07	1.00	72	1100.08
`subScenario2L`	0.91	187	657.04	0.90	188	657.23
`subScenario2S`	0.90	70	685.97	0.93	68	688.71
`subScenario2R`	0.88	50	868.22	0.90	58	869.48
Compositional	**0.90**	**383**	**777.81**	**0.91**	**386**	**777.27**
Monolithic	**0.88**	**639**	**1796.28**	**0.89**	**698**	**1796.90**

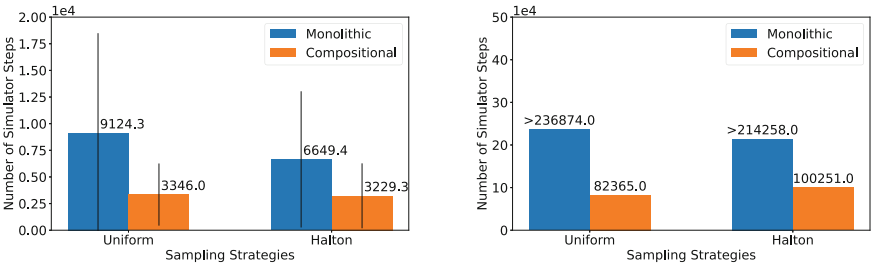
the road segment of `subScenario1`. `subScenario0W` samples a position for a wall that blocks half of the road (Fig. 2e), and `subScenario0B` (Fig. 2f) samples positions for three oil barrels. Both of these sub-scenarios can potentially cause a specification violation if either `Leader` or `Follower` cannot perform the necessary maneuver on time to avoid a collision with these obstacles. Specifically, `Leader`'s controller uses its LiDAR sensor to sense surrounding obstacles and attempts to avoid them while still following the yellow line whereas `Follower` does not implement any obstacle avoidance procedure, so to avoid obstacles, its needs to follow the maneuvers of `Leader` precisely.

Falsification. Similar to the previous case study, we run both methods with 10 different random seeds, and our method is, again, run in the batched mode with a batch size of 5. Figure 4a presents the results for falsification. Similar to the previous case study, both methods find counter-examples before their output distributions stabilize. Figure 4a shows that our method finds counter-examples significantly faster than the baseline, and its standard deviation is more stable. Between uniform and Halton sampling strategies, we observe a similar pattern to that observed in 6.3, i.e., due to its uniform coverage property, Halton sampling finds counter-examples slightly faster. Notice that the simulator steps in these experiments are smaller than the previous one since the new initial sub-scenarios can also potentially cause a specification violation, whereas in the previous case

study, we have not observed any specification violation in the first sub-scenario; therefore, we find counter-examples earlier in this case study.

Statistical Verification. To compare the compositional statistical verification with the monolithic baseline, we run both until their output distributions stabilize. However, for this experiment, we set the convergence threshold to a larger value, i.e., $\Delta = 0.005$, and we also set an upper bound of 100 simulations for the number of simulations performed. Specifically, the statistical verification process terminates either when the output distributions converge or when the process reaches 100 simulations. The motivation for this decision is to understand how the proposed method compares when fewer simulations are performed, which causes output distributions to be less accurate for each sub-scenario. With the given threshold value, i.e., $\Delta = 0.005$, the compositional approach stabilizes before reaching 100 simulation runs whereas the monolithic baseline reaches the upper bound of 100 simulations before its output distribution converges. Figure 4b presents the results for this experiment. We observe that the proposed approach performs better than the baseline, and we also see a slight increase in the total number of simulator steps due to Halton sampling. Since the baseline cannot converge to a stable output distribution before 100 simulations, its results are statistically less reliable than the results for the compositional method, and its total number of simulator steps is upper-bounded by 100 simulations, not the reliability of its output distribution.

Table 2 presents the comparison between the specification satisfaction probabilities output by both methods. The estimated probability for the compositional method is calculated by combining the results for each sub-scenario. Table 2 shows that even with a less accurate output distribution approximation, probabilities estimated by both methods are close to each other. Moreover, even though the number of simulations performed by the compositional method is more than the monolithic baseline (which is due to the fact that the baseline reaches the upper bound for the number of simulations), the average simulation length (i.e., the average number of simulator steps) is much smaller than the baseline since the compositional approach performs shorter simulations at the sub-scenario level. The average simulation length for the compositional method is calculated by combining the results from all sub-scenarios. Note that our method also provides more insight into the system behavior. For example, in Table 2, we observe that `subScenario0W` has the smallest specification satisfaction probability, which implies that the system does not perform well in the presence of large obstacles blocking half of the road. We conclude by noting that on the same hardware as the previous case study, our compositional statistical verification method took a little over 1 h to converge for both uniform and Halton sampling, whereas the monolithic baseline took 1.5 h to reach the simulation limit for both sampling strategies.



(a) Falsification (b) Statistical Verification

Fig. 4. Experiment results for falsification and statistical verification of Case Study 2.

Table 2. Estimated Specification Satisfaction Probabilities for Case Study 2.

	Uniform Sampling			Halton Sampling		
	Estimated Probability	Number of Sim	Mean Sim Length	Estimated Probability	Number of Sim	Mean Sim Length
`subScenario0W`	0.63	41	924.49	0.55	53	861.08
`subScenario0B`	0.94	31	1235.32	0.90	30	1159.70
`subScenario1`	1.00	8	1122.75	1.00	25	1122.92
`subScenario2L`	0.92	63	656.05	0.94	62	660.84
`subScenario2S`	0.92	26	687.88	0.88	26	683.46
`subScenario2R`	0.94	16	885.44	0.93	15	895.73
Compositional	**0.73**	**185**	**857.10**	**0.66**	**211**	**856.30**
Monolithic	**0.69**	**Timeout**	**Timeout**	**0.61**	**Timeout**	**Timeout**

7 Related Work

Compositional Analysis Methods. Formal compositional analysis techniques have a long history in the design and verification of systems [3–6,8,10,15,18]. Many of these methods are based on assume-guarantee reasoning. These include methods concerned with compositional reasoning for properties expressed in temporal logics [3,5], approaches with a focus on compositional verification for models such as interface and I/O automata [10,15], and those for the contract-based design of systems [2,18]. With the rise of ML-based components, approaches for the compositional verification of systems with black-box components have been investigated. For example, an approach for the compositional falsification of systems with DNN components was introduced in [8]. An initial investigation of compositional verification for these types of systems was introduced in [19]. In contrast to these approaches, our introduced framework provides a compositional approach from a simulation-based, not model-based, analysis perspective, with the goal of increasing the scalability of simulation-based methods.

Statistical Analysis Methods. The increasing complexity of cyber-physical systems, making combinatorial methods infeasible, has increased the interest in investigating statistical analysis methods [1,21,27–30]. Coined by the term statistical model checking [16], many scalable simulation-based methods have been introduced in the literature that can give formal statistical guarantees on the correctness of a system relying on different statistical methods. For example, Zuliani et al. show how a statistical model approach based on Bayesian statistics can be used to solve the probabilistic model checking problem for temporal properties and for system models given by Stateflow-style hybrid systems with probabilistic transitions [30]. Younes and Simmons [29] use hypothesis testing and discrete-event simulation to perform probabilistic verification of continuous-time stochastic processes. David et al. [7] present an approach based on statistical model checking to check the correctness of timed systems. In addition to handling systems with large state spaces, a significant benefit of using statistical model checking is that it can handle systems whose implementation models are unknown. For example, Sen et al. [22] present a statistical approach for the verification of stochastic systems based on Monte Carlo simulation and statistical hypothesis testing, with no knowledge of a formal model for the system. An improved algorithm is provided by Younes [26]. Our framework can be instantiated with all the methods mentioned above, allowing for a compositional approach to applying these statistical model-checking methods.

8 Conclusion

We presented a framework for the compositional simulation-based analysis of AI-based autonomous systems. Given a simulation-based analysis task, our approach decomposes the task into several smaller simulation-based analysis tasks avoiding the execution of expensive long-running simulations. Results for the overall tasks are computed by stitching together the results obtained from the smaller analysis tasks. We show how our framework can be used to generate compositional algorithms for falsification and statistical verification. Our experimental results show the scalability and efficacy of our approach in comparison to monolithic simulation-based analysis methods.

References

1. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. **28**(1), 1–39 (2018)
2. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2007. LNCS, vol. 5382, pp. 200–225. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92188-2_9
3. Bhaduri, P., Ramesh, S.: Interface synthesis and protocol conversion. Form. Asp. Comput. **20**(2), 205–224 (2008)

4. Chilton, C., Jonsson, B., Kwiatkowska, M.Z.: Compositional assume-guarantee reasoning for input/output component theories. Sci. Comput. Program. **91**, 115–137 (2014)
5. Clarke, E.M., Long, D.E., McMillan, K.L.: Compositional model checking. In: Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS 1989), Pacific Grove, California, USA, 5–8 June 1989, pp. 353–362. IEEE Computer Society (1989)
6. Cobleigh, J.M., Giannakopoulou, D., PĂsĂreanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36577-X_24
7. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 349–355. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_27
8. Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. J. Autom. Reason. **63**(4), 1031–1053 (2019)
9. Dreossi, T., et al.: VerifAI: a toolkit for the formal design and analysis of artificial intelligence-based systems. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 432–442. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_25
10. Emmi, M., Giannakopoulou, D., Păsăreanu, C.S.: Assume-guarantee verification for interface automata. In: Cuellar, J., Maibaum, T., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 116–131. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68237-0_10
11. Fremont, D.J., et al.: Scenic: a language for scenario specification and data generation. Mach. Learn. **112**, 3805–3849 (2023)
12. Halton, J.H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numer. Math. **2**, 84–90 (1960)
13. Hastie, T., Tibshirani, R., Friedman, J.H., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, vol. 2. Springer, New York (2009). https://doi.org/10.1007/978-0-387-21606-5
14. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. **2**(4), 255–299 (1990)
15. Larsen, K.G., Nyman, U., Wąsowski, A.: Interface input/output automata. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 82–97. Springer, Heidelberg (2006). https://doi.org/10.1007/11813040_7
16. Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R.: Statistical model checking. In: Steffen, B., Woeginger, G. (eds.) Computing and Software Science. LNCS, vol. 10000, pp. 478–504. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_23
17. Michel, O.: Webots: professional mobile robot simulation. J. Adv. Robot. Syst. **1**(1), 39–42 (2004)
18. Nuzzo, P., Li, J., Sangiovanni-Vincentelli, A.L., Xi, Y., Li, D.: Stochastic assume-guarantee contracts for cyber-physical system design. ACM Trans. Embed. Comput. Syst., **18**(1), 2:1–2:26 (2019)
19. Pasareanu, C.S., Gopinath, D., Yu, H.: Compositional verification for autonomous systems with deep learning components. CoRR, abs/1810.08303 (2018)

20. Saikrishna, V., Ray, S.: MML inference of hierarchical probabilistic finite state machine. In: 2019 Cybersecurity and Cyberforensics Conference (CCC), pp. 78–84 (2019)
21. Sen, K., Viswanathan, M., Agha, G.: VESTA: a statistical model-checker and analyzer for probabilistic systems. In: Second International Conference on the Quantitative Evaluation of Systems (QEST 2005), pp. 251–252 (2005)
22. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 202–215. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27813-9_16
23. Seshia, S.A., Sadigh, D., Sastry, S.S.: Toward verified artificial intelligence. Commun. ACM **65**(7), 46–55 (2022)
24. Webots. http://www.cyberbotics.com Open-source Mobile Robot Simulation Software
25. Yannakakis, M.: Hierarchical state machines. In: van Leeuwen, J., Watanabe, O., Hagiya, M., Mosses, P.D., Ito, T. (eds.) TCS 2000. LNCS, vol. 1872, pp. 315–330. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44929-9_24
26. Younes, H.L.S.: Probabilistic verification for "black-box" systems. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 253–265. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_25
27. Younes, H.L.S.: Ymer: a statistical model checker. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 429–433. Springer, Heidelberg (2005). https://doi.org/10.1007/11513988_43
28. Younes, H.L., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. Int. J. Softw. Tools Technol. Transf. **8**(3), 216–228 (2006)
29. Younes, H.L.S., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Inf. Comput. **204**(9), 1368–1409 (2006)
30. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Stateflow/simulink verification. Formal Meth. Syst. Des. **43**(2), 338–367 (2013)