Scenario-based Compositional Verification of Autonomous Systems with Neural Perception

Christopher Watson[1][0000−0003−3716−516X](✉), Rajeev Alur[1][0000−0003−1733−7083], Divya Gopinath[2][0000−0002−1242−7701] Ravi Mangal[3][0000−0001−6267−6995], and Corina S. Păsăreanu[2,4][0000−0002−5579−6961]

[1] University of Pennsylvania, USA
ccwatson@seas.upenn.edu
[2] KBR Inc., NASA Ames, USA
[3] Colorado State University, USA
[4] Carnegie Melon University, USA

Abstract. Recent advances in deep learning have enabled the development of autonomous systems that use deep neural networks for perception. Formal verification of these systems is challenging due to the size and complexity of the perception DNNs as well as hard-to-quantify, changing environment conditions. To address these challenges, we propose a probabilistic verification framework for autonomous systems based on the following key concepts: (1) *Scenario-based Modeling:* We decompose the task (e.g., car navigation) into a composition of *scenarios*, each representing a different environment condition. (2) *Probabilistic Abstractions:* For each scenario, we build a compact *abstraction of perception* based on the DNN's performance on an offline dataset that represents the scenario's environment condition. (3) *Symbolic Reasoning and Acceleration:* The abstractions enable efficient compositional verification of the autonomous system via symbolic reasoning and a novel *acceleration proof rule* that bounds the error probability of the system under arbitrary variations of environment conditions. We illustrate our approach on two case studies: an experimental autonomous system that guides airplanes on taxiways using high-dimensional perception DNNs and a simulation model of an F1Tenth autonomous car using LiDAR observations.

1 Introduction

Recent advances in deep learning have enabled the development of autonomous systems that use deep neural networks (DNNs) for perception [10–12,19,21,32] Formal verification of these systems is uniquely challenging due to the complexity of formal reasoning about the DNNs. The perception DNNs are massive (with millions or billions of parameters) and are also known to be very sensitive to input perturbations. For instance, changing the light conditions in an image can lead to very different DNN outputs and this can adversely affect the safety of the overall system. Furthermore, real-world autonomous systems typically operate in an environment whose configuration is a priori unknown (e.g., the car is navigating in a location with an unknown layout of tracks). In this work, we propose a probabilistic verification framework to address these challenges.

System Description. Consider the autonomous system from Figure 1; it consists of four interacting components: *Sensor*, *Perception DNN*, *Controller*, and *Dynamics*. The *Sensor* (e.g., a camera or LiDAR system) observes the current underlying system state (potentially subject to hard-to-model environmental conditions) and produces sensor readings (e.g., images or LiDAR readings). The *Perception DNN* takes inputs from the *Sensor* and produces *state estimates* that are used by the *Controller* to output control commands for the system. In response to these commands, the system state is updated as described by the *Dynamics* and the cycle repeats. This forms a closed-loop system where the *Perception DNN* repeatedly receives *Sensor* inputs as the system operates in its environment. Note that such systems tend to be stochastic, even when *Dynamics* and *Controller* are deterministic, due to the uncertainty in the *Sensor* readings.

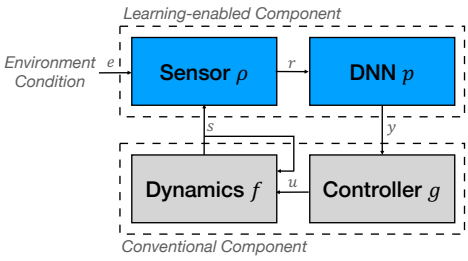


Fig. 1: Components of an autonomous system with DNN-based perception.

Safety Properties. We analyze such systems with respect to critical safety properties that can be framed as reachability queries. We focus on two questions: (1) given a set of initial conditions, what is the probability of reaching an error state? and (2) given a user-specified bound on the probability of reaching an error state, what is the corresponding set of most permissive initial conditions? The specific notion of error depends on the system being analyzed. In our first case study, we analyze an autonomous airplane taxiing system that relies on Boeing's *TaxiNet* perception DNN and define error as excessive deviation from the runway's centerline. In our simulated F1Tenth case study, we define error to be collision with the edge of the race track or failure to navigate a track segment within a predetermined time limit.

Modeling the Learning-enabled Components. The conventional components (*Controller* and *Dynamics*) can be modeled using well-known techniques [29], for instance, they can be modeled as transition systems or Discrete-time Markov Chains (DTMCs). However, the learning-enabled components (*Perception DNN* and *Sensor*) are difficult to model due to enormous sizes of modern DNNs and the complex nature of the sensors as well as the changing environmental conditions in which these systems operate. The key challenge is to analyze the learning enabled components in a way that can be composed with traditional systems verification techniques to obtain a guarantee for the entire closed loop system. Approaches to this challenge form three groups: (1) those that apply neural network verification to the *Perception DNN* and precisely model the *Sensor* mathematically [8, 14, 15, 30] or with a learned model [1, 5, 22, 33] (2) those that build *contracts* [2, 17, 18, 24, 25, 29, 31] around the learning enabled components, and (3) those that build *probabilistic abstractions* [3, 4, 6, 7, 28] around the learning enabled component.

In general, neural network verification only scales to DNNs of modest size, and realistic sensors are hard to model, especially when subject to environmental disturbances. Contract-based approaches provide obligations for the learning-enabled

components that ensure system safety, but cannot quantify system safety if the contracts are not met. To provide a scalable, quantitative analysis of sytem-level safety, our current work extends the technique of *probabilistic abstraction*, which models the learning-enabled components' behavior as an explicit map from each system state to a distribution over *Perception DNN* outputs. The probabilities in this map are computed from the *confusion matrix* that measures the performance of the *Perception DNN* on a representative dataset obtained from simulations or real-world setting. With this abstraction, the resulting system can be modeled as a DTMC and is amenable to verification using off-the-shelf tools such as PRISM [23] and Storm [16].

While the compact abstraction of the learning-enabled components allows the analysis to scale to arbitrarily large DNNs and does not require modeling the *Sensor*, if the offline dataset pools data from multiple distinct environment conditions the abstraction may be too coarse, resulting in an imprecise analysis.

Our Proposal. In this work, we structure the analysis of the autonomous system as a composition of *scenarios*, each of which represents the behavior of the system under a different *environment condition* for a fixed duration. In the context of our autonomous taxiing case study, we observe that the *TaxiNet* perception system has dramatically different performance in *bright* versus *dark* lighting conditions (see Figure 2). To perform an analysis of system level safety that accounts for changes in lighting conditions, we define the scenarios `bright` and `dark`. Within the `bright` scenario, we can accurately model the behavior of the closed loop system by incorporating a probabilistic of abstraction constructed using a dataset collected in *bright* light. A separate dataset lets us accurately model the system's behavior in the `dark` scenario. Similarly, to analyze an autonomous race car, we define a separate scenario for each kind of track segment (*straight*, *left turn*, or *right turn*) to efficiently analyze safety for a diverse set of track configurations. While scenario-based decomposition is inspired by the non-probabilistic analysis of [20], our probabilistic abstraction of the learning-enabled component of each scenario allows our analysis to scale to arbitrarily complex sensor and perception systems. Scenario-based decomposition also allows our approach to efficiently compute safety guarantees for long time horizons.

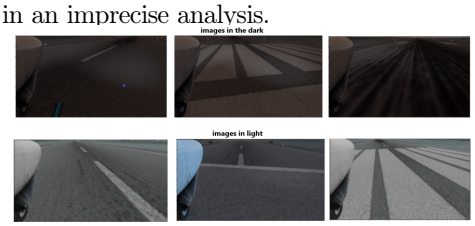


Fig. 2: TaxiNet achieves 91.25% accuracy when estimating heading for a *bright* image dataset but only 53.87% accuracy for a *dark* image dataset.

Contributions. Overall, we make the following contributions: (1) we present an approach for compactly modeling autonomous systems that accounts for changing environment conditions via decomposition into environment-specific scenarios; (2) we extend prior work [28] by allowing a system designer to define separate probabilistic abstractions, representing DNN performance in different environment conditions; we further generalize the confusion matrices to account for arbitrary state estimate definitions, not just the underlying system state as in previous work; (3) we describe

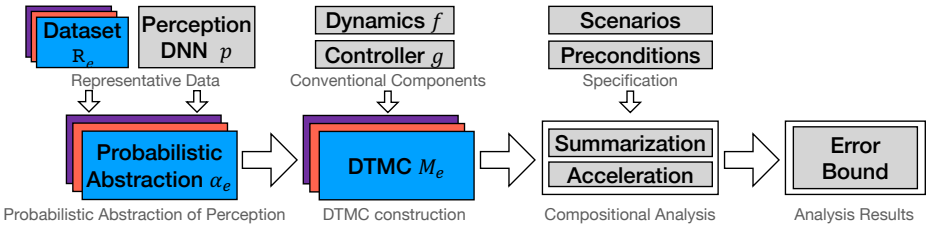


Fig. 3: Scenario-based compositional analysis of autonomous systems.

an efficient symbolic approach for analyzing finite sequences of scenarios with respect to initial conditions specified as a set of distributions over the system states; (4) we also give a novel family of compositional proof rules for accelerated analysis of finite sequences with arbitrary interleavings of scenarios; these rules enable developers to compute a bound on error probability even when the precise configuration of the environment is not known a priori; (5) we illustrate our approach on two case studies: a model of an experimental autonomous system that guides airplanes on taxiways using high-dimensional perception DNNs and a model of an F1Tenth autonomous car using LiDAR observations.

Related Work. We have already discussed closely related work on formal analysis of learning-enabled systems. Other related work includes approaches based on statistical simulation [35] that require less computation but provide a different type of guarantee than probabilistic model checking. Moreover, statistical model checking is not suitable for models that include nondeterminism, thus cannot be compared with the guarantees we obtain for arbitrary interleavings of scenarios. Another related line of work is compositional probabilistic model checking [34], which, however does not address the central challenge of incorporating machine-learning components.

Also related is the work on probabilistic predicate transformers [26]. Each of our scenarios can be seen as a probabilistic assignment, and the backward analysis of Section 3.1 is a special case of the weakest precondition computation. Much work on predicate transformers concerns general loops, which are not immediately relevant to modeling autonomous systems over bounded horizons. To our knowledge, no analogue of our acceleration rules exists in the literature.

2 Scenario-based Probabilistic Abstractions of Perception

Figure 3 illustrates our approach. The goal is to analyze safety properties of autonomous systems, subject to changing environment conditions. We assume that a domain expert can identify a finite set E of environment conditions, e.g. *bright* vs *dark* lighting. For each environment condition $e \in E$, the domain expert provides a dataset $\mathtt{R}_e$ that represents the behavior of the *Sensor* subject to that environment condition. This, combined with black-box access to the *Perception DNN*, allows us to construct a *probabilistic abstraction of perception* α_e for each environment condition, which can in turn be composed with a model of the conventional components to yield a DTMC model M_e of the closed-loop system subject to each environment condition.

For simplicity of presentation, we assume that only the behavior of learning-enabled components changes with the environment conditions, more generally, the conventional components may also change. Our techniques are applicable to this more general setting, and in Section 4 we consider such a case.

A scenario represents the behavior of the autonomous system subject to a particular environment condition for a fixed finite time horizon. The inputs to our compositional analysis are a sequence of scenarios and a precondition that constrains the set of initial state distributions. A *summarization* procedure allows us to efficiently compute a worst-case error bound for a fixed scenario sequence, and our *acceleration* proof rules allow us to derive a bound on error probability that is parametric with respect to the length of a scenario sequence of unbounded length. We make these notions precise in the following sections.

2.1 Modeling the Closed-loop Autonomous System

A discrete-time closed-loop autonomous system is composed of conventional components *Controller* and *Dynamics* as well learning-enabled components *Sensor* and *Perception DNN*. Given a fixed environment condition $e \in E$, at each timestep t, the *Sensor* $\rho_e : S \to \mathbf{Dist}(R)$[5] (which may be a camera or LiDAR system) observes the current underlying system state $s_t \in S$ and probabilistically produces a sensor reading $r_t \in R$ sampled from the distribution $\rho_e(s_t)$ (where r_t is an image, LiDAR reading, etc). The *Perception DNN* $p : R \to Y$ processes r_t to yield an estimate $y_t \in Y$ of the system state. The *Controller* $g : Y \to U$ receives this state estimate and outputs a control signal $u_t \in U$. At the start of the next timestep $t+1$, the *Dynamics* $f : S \times U \to S$ produces a new system state $s_{t+1} \in S$ in response to the control input u_t and the cycle continues. The sets S, Y, and U are assumed to be finite.

Probabilistic Abstraction of Perception. Constructing models of the *Sensor* and the *Perception DNN* is a central challenge given their complexity. We extend past work [28] and build compact abstractions $\alpha_e : S \to \mathbf{Dist}(Y)$ of the learning-enabled components.

The key idea is that the input-output behavior of the learning-enabled components (as a function of type $S \to \mathbf{Dist}(Y)$) can be estimated using an offline dataset of sensor readings gathered using the *Sensor* onboard the autonomous system. A limitation of previous work was that they built a single abstraction, regardless of environment conditions, leading to overly imprecise results, as illustrated in Figure 2.

In this work, we address this important limitation by building a separate probabilistic abstraction of perception α_e for each environment condition e. For each pair of environment condition e and state s, we assume access to an offline dataset $\mathtt{R}_{e,s}$ of sensor readings collected while the autonomous system was in state s and subject to condition e.

Another limitation of previous work was that the probabilistic abstraction was based on computing confusion matrices under the restricted assumption that the

[5] We write $\mathbf{Dist}(X)$ for the set of probability distributions over a finite set X and identify each $\boldsymbol{x} \in \mathbf{Dist}(X)$ with its representation as a row vector in $\mathbb{R}^{|X|}$.

DNN outputs correspond precisely to the state definition. In practice, DNNs can be used to estimate only some aspects of the underlying state. Thus, we generalize confusion matrices to account for arbitrary state estimate definitions, not just the underlying system state as in previous work.

The construction of probabilistic abstraction α_e for condition e proceeds as follows. First, for each $s \in S$, we apply the *Perception DNN* function $p: R \to Y$ to each r_s in $\mathtt{R}_{e,s}$ to obtain the pair $(s, p(r_s))$; here, r_s is a sensor reading gathered in state s because it is from $\mathtt{R}_{e,s}$. We combine all of these pairs to form a labeled dataset of $S \times Y$ pairs that describes the behavior of the learning-enabled components subject to e. From this dataset we can build a contingency matrix[6] $\boldsymbol{C}_e$. For any $s \in S$ and $y \in Y$, we write $\boldsymbol{C}_e(s,y)$ for the frequency of the pair (s,y) in the labeled dataset. We define the probabilistic abstraction of perception $\alpha_e : S \to \mathbf{Dist}(Y)$ as

$$\alpha_e(s)(y) := \frac{\boldsymbol{C}_e(s,y)}{\sum_{y' \in Y}(\boldsymbol{C}_e(s,y'))}$$

System Model. The probabilistic abstraction α_e of the learning-enabled component in environment condition e can be composed with the discrete models of the conventional components to yield a discrete-time Markov chain (DTMC) that models the closed-loop system as in [3, 7, 28]. We denote this DTMC as $\mathcal{M}_e = (S, \boldsymbol{P}_e)$ in which S is a finite set of states (in particular, the set of states of the autonomous system) and $\boldsymbol{P}_e$ is a $|S| \times |S|$ right-stochastic matrix. At each timestep, the probability of a transition from some $s \in S$ to some $s' \in S$ is:

$$\boldsymbol{P}_e(s,s') := \sum_{y \in Y} \alpha_e(s)(y)[f(g(y)) = s']$$

where $[f(g(y)) = s']$ evaluates to 1 whenever $f(g(y)) = s'$ and 0 otherwise. For our case studies, we follow the procedure of [28] to represent this DTMC using PRISM, see section 4.1 for details.

Without loss of generality, we assume that the autonomous system has a distinguished *error state*, which we denote s_{err}. We assume that the conventional dynamics ensure that the error state is *absorbing*, i.e., once the system reaches s_{err}, it stays there. This modeling assumption both reflects the natural meaning of an "error state" and facilitates the compositional analysis.

2.2 Scenarios

We are interested in modeling the behavior of autonomous systems under changing environmental conditions. For this purpose, we introduce the notion of *scenarios*. Consider an autonomous system with states S and environment conditions E. For each $e \in E$, let $M_e = (S, \boldsymbol{P}_e)$ be the DTMC that models execution of the autonomous system subject to environment condition e.

[6] This is a *confusion matrix* when $S = Y$.

A *scenario* is a pair $(e,H) \in E \times \mathbb{N}$ that comprises an environment condition and a time horizon. Semantically, each scenario induces a map $[\![(e,H)]\!] : \mathbf{Dist}(S) \to \mathbf{Dist}(S)$ defined as

$$[\![(e,H)]\!](\boldsymbol{x}) := \boldsymbol{x}\boldsymbol{P}_e^H$$

that maps a distribution $\boldsymbol{x} \in \mathbf{Dist}(S)$ to the transient distribution reached after taking H steps in M_e.

To model transitions between environment conditions, we consider non-empty sequences of scenarios. Each scenario is also a *scenario sequence*. Given two scenario sequences w_1 and w_2, the sequential composition $w_1;w_2$ is also a scenario sequence. Sequential composition is associative. Semantically, we define $[\![w_1;w_2]\!] : \mathbf{Dist}(S) \to \mathbf{Dist}(S)$ inductively as:

$$[\![w;w']\!] := [\![w']\!] \circ [\![w]\!]$$

Scenario Summary. When representing the semantics $[\![w]\!]$ of a scenario $w = (e,H)$, we find it useful to treat the error state s_{err} in a special manner as this facilitates the compositional analysis presented in Section 3.2. To treat the error state separately, we first partition the state space S of M_e as $S = \underline{S} \sqcup \{s_{err}\}$. Then, our explicit representation of $[\![(e,H)]\!]$ takes the form of the *summary* $(\boldsymbol{A},\boldsymbol{b})$, such that for any initial distribution $\boldsymbol{x} \in \mathbf{Dist}(\underline{S})$ (we assume here that the initial distribution places no weight on s_{err}), executing the scenario (e,H) causes the autonomous system to transition to the error state with probability $\boldsymbol{x} \cdot \boldsymbol{b}$ and induces the subdistribution[7] $\boldsymbol{x}\boldsymbol{A} \in \mathbf{SDist}(\underline{S})$ over the non-error states. Since we assume that the error state is absorbing, the error probability $\boldsymbol{x} \cdot \boldsymbol{b}$ is the probability that the system reaches the error state at any timestep during the scenario. The summary $(\boldsymbol{A},\boldsymbol{b})$ of an atomic scenario $w = (e,H)$ is related to the stochastic matrix $\boldsymbol{P}_e$ as follows:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{0} & 1 \end{bmatrix} = \boldsymbol{P}_e^H$$

We lift the definition of summary to scenario sequences inductively. If a scenario w has summary C, then the singleton scenario sequence w also has summary C. If the scenario sequences w_1 and w_2 have summaries $C_1 = (\boldsymbol{A}_1,\boldsymbol{b}_1)$ and $C_2 = (\boldsymbol{A}_2,\boldsymbol{b}_2)$, respectively, then the scenario sequence $w_1;w_2$ has the summary $C_1C_2 = (\boldsymbol{A}_1\boldsymbol{A}_2, \boldsymbol{b}_1 + \boldsymbol{A}_1\boldsymbol{b}_2)$. This composition is well-defined because each summary is defined with respect to the set of system states S, which is common to all scenarios.

3 Compositional Analysis with Scenario Summaries

We analyze the probability that the autonomous system will reach the distinguished error state s_{err} during the execution of a scenario sequence. We propose two types of analysis: (1) a symbolic analysis that provides a tight analysis of the error probability of a fixed scenario sequence with respect to a symbolic precondition and (2) a novel *acceleration rule* that can be used to derive an upper bound on error probability that is parametric with respect to the length of an arbitrary interleaving of a set of scenarios.

[7] A subdistribution $\boldsymbol{x} \in \mathbf{SDist}(\underline{S})$ may have $|\boldsymbol{x}| < 1$.

3.1 Symbolic Analysis of Fixed Scenarios

For a fixed scenario sequence w with summary $(\boldsymbol{A},\boldsymbol{b})$, the expression $\boldsymbol{x}\cdot\boldsymbol{b}$ represents the probability that the autonomous system will reach the error state when started from an initial state distribution $\boldsymbol{x}$. This linear expression allows us to efficiently perform *forward* and *backward* analyses that relate preconditions over initial distributions with bounds on error probability.

Forward Analysis. A domain expert may identify a precondition ϕ over the set of initial state distributions and wish to compute the worst-case probability of reaching the error state during the execution of the scenario sequence from an initial distribution $\boldsymbol{x}\in\mathbf{Dist}(\underline{S})$ that satisfies ϕ. Recall that we represent each distribution as a vector in $\mathbb{R}^{|\underline{S}|}$. We consider preconditions that can be expressed as an intersection of affine constraints, i.e. of the form $\boldsymbol{x}\boldsymbol{a}_1\leq\theta_1\wedge\ldots\wedge\boldsymbol{x}\boldsymbol{a}_d\leq\theta_d$ for some coefficient vectors $\boldsymbol{a}_1,\ldots,\boldsymbol{a}_d\in\mathbb{R}^{|\underline{S}|}$ and offset scalars $\theta_1,\ldots,\theta_d\in\mathbb{R}$.

The worst-case error probability is given by the following linear program:

$$\begin{aligned}\max_{\boldsymbol{x}}\quad & \boldsymbol{x}\cdot\boldsymbol{b}\\ \text{subject to}\quad & \boldsymbol{x}\cdot\boldsymbol{a}_i\leq\theta_i,\quad \forall i,\in\{1,\ldots,d\}\\ & \boldsymbol{x}\geq 0,\\ & \|\boldsymbol{x}\|_1=1.\end{aligned}$$

where the first d constraints encode ϕ and the last two constraints enforce that $\boldsymbol{x}$ represents a probability distribution. This linear program can be solved using optimization software such as Gurobi [13].

Backward Analysis. If a domain expert provides a maximum allowable error probability ϵ for the execution of the scenario sequence, the precondition $\phi\coloneqq\boldsymbol{x}\cdot\boldsymbol{b}\leq\epsilon$ is the weakest precondition over initial state distributions that ensures the error probability does not exceed ϵ.

Comparison with PRISM. The symbolic analyses allow the domain expert to reason about the error probability of the autonomous system with respect to a set of initial distributions. This complements the analyses supported by the probabilistic model checker PRISM, which permits forward analysis from a single initial state (or finite set of initial states).

3.2 Acceleration Rules

The aforementioned analyses are useful when the domain expert fixes a particular scenario sequence to consider. However, such foresight is not always possible. To address this, we introduce novel *acceleration* rules that allow us to bound the probability of error without fixing the length of the scenario sequence or the order in which the scenarios appear *a priori*. with respect to an unpredictable sequence of environment conditions.

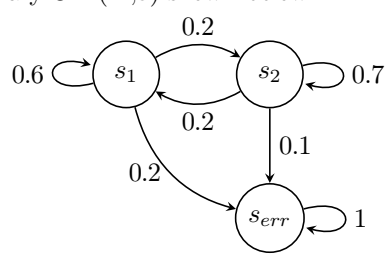
Hoare-style Assertions. We reason about scenario sequences using Hoare-style assertions with special treatment of error states. An assertion is a quadruple of the form $\{\phi\}C\{\psi\}\{\epsilon\}$ where $C \equiv (\boldsymbol{A},\boldsymbol{b})$ denotes the summary of a scenario sequence, the precondition ϕ and postcondition ψ are predicates over $\mathbf{Dist}(\underline{S})$ expressed as intersections of affine constraints over $\mathbb{R}^{|\underline{S}|}$ and $0 \leq \epsilon \leq 1$ is the required bound on error probability. We write $\phi(\boldsymbol{x})$ when $\boldsymbol{x}$ satisfies ϕ. The assertion $\{\phi\}C\{\psi\}\{\epsilon\}$ holds exactly when

$$\forall \boldsymbol{x} \in \mathbf{Dist}(\underline{S}), \phi(\boldsymbol{x}) \Rightarrow \boldsymbol{x}\cdot\boldsymbol{b} \leq \epsilon \wedge \psi(norm(\boldsymbol{x}\boldsymbol{A}))$$

Here *norm* is an operation that (L1) *normalizes* a subdistribution to make it a proper distribution, i.e., $norm(\boldsymbol{x}) = \frac{\boldsymbol{x}}{|\boldsymbol{x}|}$ We normalize the output sub-distribution $\boldsymbol{x}\boldsymbol{A}$ of C as ψ is defined with respect to $\mathbf{Dist}(\underline{S})$. Given the explicit summary of the scenario, such assertions can be checked efficiently using an off-the shelf solver, such as Z3 [9] as we will show in the following example.

Example. Consider the scenario with summary $C \equiv (\boldsymbol{A},\boldsymbol{b})$ shown below:

$$\boldsymbol{A} = \begin{bmatrix} 0.6 & 0.2 \\ 0.2 & 0.7 \end{bmatrix} \boldsymbol{b} = \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}$$



Let $\boldsymbol{x} = [x_1, x_2]$ denote an initial distribution over the system states s_1 and s_2. Upon executing the scenario, the probability of transitioning to error state is $\boldsymbol{x}\cdot\boldsymbol{b} = 0.2x_1 + 0.1x_2$ and the resulting distribution over the system states (conditioned on not reaching s_{err}) will be:

$$\boldsymbol{x}' = norm(\boldsymbol{x}\boldsymbol{A}) = \frac{\boldsymbol{x}\boldsymbol{A}}{1-\boldsymbol{x}\cdot\boldsymbol{b}} = \frac{[0.6x_1+0.2x_2, 0.2x_1+0.7x_2]}{(1-(0.2x_1+0.1x_2))}$$

Let us consider the precondition $\phi := x_1 \leq 0.7 \wedge x_2 \leq 0.7$. We will show how to check the assertion $\{\phi\}C\{\phi\}\{0.15\}$ using Z3.

We can encode the set of distributions that satisfy ϕ as a polyhedron in $\mathbb{R}^{|\underline{S}|}$ with H-representation $\boldsymbol{M}_\phi \boldsymbol{x}^\top \leq \theta_\phi$ for some 6×2 matrix $\boldsymbol{M}_\phi$ and 6-dimensional vector $\boldsymbol{\theta}_\phi$ of offsets. The H-representation comprises 6 inequalities since four inequalities define the subset of $\mathbb{R}^{|\underline{S}|}$ that represents $\mathbf{Dist}(\underline{S})$ and the predicate ϕ contains two additional constraints. In order to check whether $\phi(\boldsymbol{x}) \Rightarrow \boldsymbol{x}\cdot\boldsymbol{b} \leq \epsilon \wedge \psi(norm(\boldsymbol{x}\boldsymbol{A}))$ we can ask Z3 whether $\boldsymbol{M}_\phi \boldsymbol{x}^\top \leq \boldsymbol{\theta}_\phi \wedge \neg(\boldsymbol{x}\cdot\boldsymbol{b} \leq 0.15)$ and $\boldsymbol{M}_\phi \boldsymbol{x}^\top \leq \boldsymbol{\theta}_\phi \wedge \neg(\boldsymbol{M}(norm(\boldsymbol{x}\boldsymbol{A}))^\top \leq \boldsymbol{\theta}_X)$ are both unsatisfiable. If both are unsatisfiable (as they are for this example), then we can conclude $\{\phi\}C\{\phi\}\{0.15\}$.

Rule for Sequential Composition. The building block for our acceleration rules is the following compositional rule which bounds the error probability for the sequential composition of two scenarios.

$$\frac{\{\phi\}C\{\psi\}\{\epsilon\},\{\phi'\}C'\{\psi'\}\{\epsilon'\},\psi \Longrightarrow \phi'}{\{\phi\}CC'\{\psi'\}\{1-(1-\epsilon)(1-\epsilon')\}} \text{ Rule 1}$$

Rule 1 generalizes to our first *acceleration rule*, where C^k is shorthand for sequential composition of C applied $k \in \mathbb{N}$ times

$$\frac{\{\phi\}C\{\phi\}\{\epsilon\}}{\{\phi\}C^k\{\phi\}\{1-(1-\epsilon)^k\}} \text{ Rule 2}$$

Rule 2 can be seen as a family of rules (parameterized by k) that provides a *recipe* for bounding the error probability of a k-ary sequential composition. A user only needs to prove the premise of the rule once (using constraint solving as described above) and can then apply the rule to obtain a bound on error probability for any desired k. We are now ready to present our main *acceleration rule* that generalizes the preceding rules:

$$\frac{\{\phi\}C_1\{\phi\}\{\epsilon\},...,\{\phi\}C_m\{\phi\}\{\epsilon\}}{\{\phi\}(C_1|...|C_m)^k\{\phi\}\{1-(1-\epsilon)^k\}} \text{ Rule 3}$$

Using Rule 3, we can estimate the error probability for scenarios formed through arbitrary sequential compositions of C_1, C_2, C_m up to length k (here we use '|' to represent choice). Like Rule 2, Rule 3 can be seen as a family of rules; it provides a *recipe* for bounding the error probability for sequences of scenarios of length k that can be generated from the regular expression $(C_1|...|C_m)^k$.

Note also that while the above rules are sound (see Appendix A for proofs) the error bound is not *tight*. Nevertheless, the acceleration rules give a domain expert the flexibility to efficiently bound the error probability for scenario sequences, without *a priori* knowing the sequence length or the order of scenarios.

Choice of ϕ and ϵ. To apply Rule 3, we must find a precondition ϕ and $\epsilon \in \mathbb{R}$ that satisfy the rule's premise. This ϕ needs to be "invariant" in the sense that ϕ serves as both a pre- and post-condition. We can use Z3 to check whether a particular ϕ satisfies the rule's premise as described in the context of our example above. Before introducing our heuristic search to find an invariant ϕ, we note that the vacuous precondition $\phi = \top$ satisfies Rule 3's premise $\{\phi\}C_1\{\phi\}\{\epsilon\},...,\{\phi\}C_m\{\phi\}\{\epsilon\}$ when ϵ is chosen as:

$$\epsilon = \max_{i \in \{1,...,m\}} (||\boldsymbol{b}_i||_\infty)$$

where $C_i \equiv (\boldsymbol{A}_i, \boldsymbol{b}_i)$ for each i. This ϵ is the worst-case *local* error probability achieved from any initial distribution by any C_i. In practice, this ϵ may be too high to provide a useful safety guarantee.

We describe our heuristic search for an invariant ϕ that allows us to apply Rule 3 to bound the error probability of arbitrary sequential compositions of a finite set of scenario sequences with summaries $(\boldsymbol{A}_1, \boldsymbol{b}_1),...,(\boldsymbol{A}_m, \boldsymbol{b}_m)$. We use the same procedure in the context of Rule 2, which is equivalent to Rule 3 when $m = 1$. Our search

TaxiNet Accuracy (Bright)

	(0,0)	**(0,1)**	**(0,2)**	**(1,0)**	⋮
(0,0)	964	44	30	18	
(0,1)	4	233	0	2	
(0,2)	3	1	258	0	
(1,0)	45	39	3	475	
		⋯			

TaxiNet Accuracy (Bright)

	(0,0)	**(0,1)**	**(0,2)**	**(1,0)**	⋮
(0,0)	499	621	14	13	
(0,1)	1	473	0	3	
(0,2)	188	23	84	0	
(1,0)	203	765	1	125	
		⋯			

Table 1: Excerpted rows and columns from the TaxiNet *bright* (left) and *dark* (right) confusion matrices. The rows and columns are labeled by states, expressed as (*cte*,*he*) pairs.

considers only the preconditions that are *weakest preconditions* with respect to a particular value of ϵ.

1. Choose desired local error probability $0 \leq \epsilon \leq 1$ and define $\phi := \wedge_{i \in \{1,\ldots,m\}} \boldsymbol{x} \cdot \boldsymbol{b}_i \leq \epsilon$.
2. Check whether $\forall i \in \{1,\ldots,m\}, \forall \boldsymbol{x} \in \mathbf{Dist}(\underline{S}), \phi(\boldsymbol{x}) \Rightarrow \phi(\frac{\boldsymbol{x}\boldsymbol{A}_i}{1-\boldsymbol{x}\cdot\boldsymbol{b}_i})$ using a numerical solver such as Z3.
3. If the check from the previous step succeeds, then ϕ is an invariant precondition and we are done. Otherwise, we return to step 1 and choose a different, higher value of ϵ.

4 Experimental Evaluation

We apply our analysis techniques to two case studies, one based on the real-world TaxiNet perception system and another based on a simulated F1Tenth race car.

TaxiNet. TaxiNet is a neural network used for perception in an experimental system developed by Boeing for autonomous centerline tracking on airport taxiways. It takes as input a picture of taxiway and estimates the plane's position with respect to the centerline in terms of two outputs: *cross track error* (*cte*), which is the distance in meters of the plane from the centerline and *heading error* (*he*), which is the angle of the plane with respect to the centerline. This regression is performed by a DNN containing 24 layers including five convolution layers and three dense layers (with 100/50/10 ELU neurons) before the output layer. The resulting state estimates are fed to a controller, which maneuvers the plane to follow the centerline. Error is defined as excessive deviation from the centerline. We analyze a discretized version of the system, for details see Appendix B.1.

Prior work [28] builds a probabilistic abstraction of perception based on the performance of TaxiNet on a dataset provided by Boeing, and performs an analysis of the closed-loop system to estimate the probability of error In our current work, we refine this analysis by first partitioning the dataset into a *bright* dataset and a *dark* dataset, then building a separate probabilistic abstraction of perception for each condition.

Table 1 shows excerpts from the confusion matrices made from the *bright* and *dark* partitions. Interestingly, the frequency and kind of misclassifications is markedly different between lighting conditions. To investigate how different lighting conditions (and transitions between lighting conditions) affect system level safety, we construct

two scenarios `bright`$=(\mathit{bright},20)$ and `dark`$=(\mathit{dark},20)$ that represent execution of the autonomous system for 20 timesteps in *bright* or *dark* lighting, respectively and apply the compositional analysis techniques introduced in Section 3. To provide a point of comparison with prior work that builds a single probabilistic abstraction, [28], we also consider a *pooled* environment condition, which corresponds to the lighting condition being identically distributed at each timestep, proportionately to the frequency of light vs. dark images in the original, unpartitioned dataset. We define the scenario `pooled`$=(\mathit{pooled},20)$ to demonstrate that our `bright` and `dark` scenarios permit a refined analysis.

F1Tenth. Our second case study is inspired by the F1Tenth autonomous racing competition [27], in which a model race car navigates a track. In particular, we analyze an idealized discrete-state model of driving dynamics. The car's *Sensor* produces 21-dimensional LiDAR observations, which are processed by a multilayer perceptron with ReLU activations, two hidden layers, and 128 neurons per hidden layer, to produce state estimates. We trained the MLP using 1000 simulated observations gathered from each system state in each track segment and evaluated its performance using a separate, identically constructed dataset. Each simulated LiDAR observation was constructed from a position sampled uniformly within a small L1 radius of the discrete state's canonical position.

Inspired by [20], we consider the set of environment conditions to be the set of track segments, i.e. $E=\{\mathit{left},\mathit{right},\mathit{straight}\}$ and define the scenarios `left` $=(\mathit{left},30)$, `right`$=(\mathit{right},30)$, and `straight`$=(\mathit{straight},30)$. We allow the *Dynamics* $f:E\times S\times U\rightarrow S$ and *Controller* $g:E\times Y\rightarrow U$ to condition their behavior on the current environment condition. The dependency of f on the current track segment is necessary to model the track configuration; the dependency of g on the current track segment corresponds to a controller equipped with a perfect mode predictor that identifies the current scenario.

The meaning of a state $(\mathit{pos}_{\mathit{side}},\mathit{pos}_{\mathit{front}},\mathit{heading},\tau)\in S\setminus\{s_{err}\}$ is defined with respect to the current scenario and diagrammed in Figure 4. The components are:

- $\mathit{pos}_{\mathit{side}}\in\{-7,...,7\}$, which denotes sideways position.
- $\mathit{pos}_{\mathit{front}}\in\{0,...,16\}$, which denotes front/back position.
- $\mathit{heading}\in\{0,...7\}$ which denotes the car's heading, measured in increments of $\pi/4$ radians where $\mathit{heading}=0$ denotes facing towards the front wall.
- $\tau\in\{1,...,30\}$ which denotes the timesteps spent in the current scenario.

The set of control inputs are $U=\{-1,0,1\}$ which represent adjustments to the current heading. At each timestep, the car's heading is updated based on the control input u. Then, the car moves one grid-position in the direction of the updated heading.

To model walls and the final goal of each track segment, we define the subsets of the car positions as shown in Figure 4. During the `left` scenario, the car's goal is to reach the final region F_{left}. Failure to reach the final region F_{left} in 30 timesteps triggers a transition to s_{err}, as does attempting to enter any position not in $\mathit{Track}\cup\mathit{Track}_{\mathit{left}}$. This is defined similarly for the other track segments, for complete details, including technical treatment of the transitions between track segments, see Appendix B.2.

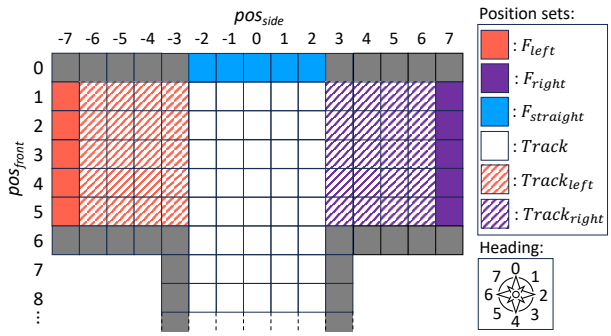


Fig. 4: Each F1Tenth scenario corresponds to a track segment.

4.1 Implementation

We adapt the PRISM model construction of [28] to allow us to compute a *summary* for each atomic scenario. The PRISM construction reflects the system decomposition illustrated in Figure 1. In particular, the PRISM model takes three timesteps (one for the learning-enabled components, one for the *Controller*, and one for the *Dynamics*) to model one timestep of the autonomous system. This coordination is mediated by bookkeeping variables, for details, see Appendix B.

Consider an autonomous system with state set S and the atomic scenarios $(e_1,H_1),...,(e_m,H_m)$. Our PRISM model contains a set of variables V_S such that each valuation to V_S uniquely determines a system state $s \in \underline{S}$. Our model also contains a variable `scenario` that maintains the index $i \in \{1,...,m\}$ of the current scenario and a variable `timer` that maintains the timestep of the original autonomous system. To model the horizon of each scenario we define the PRISM formula `exit_condition` as `! error & ((scenario = 0 & timer = H_i) | ... | (scenario = m & timer = H_m))` and ensure that each PRISM state that satisfies `exit_condition` has no outgoing transitions. To compute the summary $(\boldsymbol{A}_i,\boldsymbol{b}_i)$ of scenario (e_i,H_i), we do the following for each $s \in \underline{S}$:

1. Modify the PRISM model so that initially `scenario` $= i$, `timer` $= 1$ and the valuation to V_S determines s.
2. For each $s' \in S$, execute the PRISM query `P=? F[exit_condition &` s'`]` and fill this value into $\boldsymbol{A}_i(s,s')$. In the preceding PRISM query, s' stands for the state predicate that holds iff the valuation to V_S determines s'.
3. Execute the PRISM query `P=? F[error]` and fill this value into $\boldsymbol{b}_i(s)$.

4.2 Fixed Scenario Analysis

We apply the analysis techniques described in Section 3.1 to fixed sequences of the scenarios `bright` and `dark` from the TaxiNet case study to explore how a shift in lighting conditions affects system level safety. For the F1Tenth case study, we consider sequences of the scenarios `straight`, `left`, and `right` that represent different track configurations.

TaxiNet	$\phi^{taxi}_{nominal}$	ϕ^{taxi}_{center}	ϕ^{taxi}_{right}
`bright;bright`	0.030	0.099	0.215
`bright;dark`	0.323	0.371	0.452
`dark;bright`	0.368	0.570	0.455
`dark;dark`	0.551	0.694	0.613
`pooled;pooled`	0.307	0.462	0.399

F1Tenth	$\phi^{f1tenth}_{nominal}$	$\phi^{f1tenth}_{center}$	$\phi^{f1tenth}_{right}$
`l;l;l;l`	0.338	0.586	0.804
`r;r;r;r`	0.069	0.118	0.213
`s;s;s;s`	0.00	0.00	0.00
`r;s;s;l`	0.514	0.522	0.513
`r;s;s;r`	0.060	0.110	0.203

Table 2: Forward analysis for TaxiNet (left) and F1Tenth (right). For each scenario sequence, we report the worst-case error probability given a precondition over the initial state distribution. In F1Tenth, `l`, `r`, and `s` abbreviate `left`, `right`, and `straight`, respectively.

Forward Analysis. For the *forward analysis*, we define preconditions over the initial state distribution and report the worst-case error probability in Table 2. For TaxiNet, we consider the precondition $\phi^{taxi}_{nominal}$ that requires the airplane to start with *heading error* and *cross track error* both equal to 0 with probability >0.9, the more lax precondition ϕ^{taxi}_{center} that merely requires the airplane to start close to the center of the runway with probability >0.9, and the precondition ϕ^{taxi}_{right} that requires the airplane to start on the far right side of the runway with probability >0.9. We encode these preconditions as affine constraints. For some boolean-valued expression ψ over the variables *cte* and *he* we let $\boldsymbol{a}_\psi$ denote the *indicator vector* such that $\boldsymbol{a}_\psi(s)=1$ iff a valuation to PRISM variables that determines s satisfies ψ.

$$\phi^{taxi}_{nominal} = \boldsymbol{x}\boldsymbol{a}_{cte\neq 0 \vee he\neq 0} \leq 0.1$$
$$\phi^{taxi}_{center} = \boldsymbol{x}\boldsymbol{a}_{cte>2} \leq 0.1$$
$$\phi^{taxi}_{right} = \boldsymbol{x}\boldsymbol{a}_{cte\neq 4} \leq 0.1$$

We define the analogous preconditions $\phi^{f1tenth}_{nominal}$, $\phi^{f1tenth}_{center}$, and $\phi^{f1tenth}_{right}$ for the F1Tenth case study. As a technical detail, each F1Tenth precondition refines the precondition $\phi^{f1tenth}_{forward}$, which ensures the car at the beginning of the track segment and facing within $\pi/4$ radians of straight forward with probability 1.

$$\phi^{f1tenth}_{forward} = \boldsymbol{x}\boldsymbol{a}_{|pos_{side}|>2 \vee 1<heading<7 \vee \tau\neq 0 \vee pos_{front}\neq 15} \leq 0$$
$$\phi^{f1tenth}_{nominal} = \phi^{f1tenth}_{forward} \wedge \boldsymbol{x}\boldsymbol{a}_{pos_{side}\neq 0 \vee heading\neq 0} \leq 0.1,$$
$$\phi^{f1tenth}_{center} = \phi^{f1tenth}_{forward} \wedge \boldsymbol{x}\boldsymbol{a}_{|pos_{side}|>1} \leq 0.1,$$
$$\phi^{f1tenth}_{right} = \phi^{f1tenth}_{forward} \wedge \boldsymbol{x}\boldsymbol{a}_{pos_{side}\neq 2} \leq 0.1.$$

Environment Conditions. In the TaxiNet case study, the sequence `bright;bright` has low error probability, which shows that the learning-enabled component and controller work well in well-lit conditions. On the other hand, each sequence that includes the `dark` scenario has extremely high error probability. This dependence on light conditions would not have been detectable if we had not built separate probabilistic abstractions of perception for light and dark operating conditions. We include hypothetical results based on the naive probabilistic abstraction of perception that pools data collected in both bright and dark conditions as the scenario sequence `pooled;pooled`.

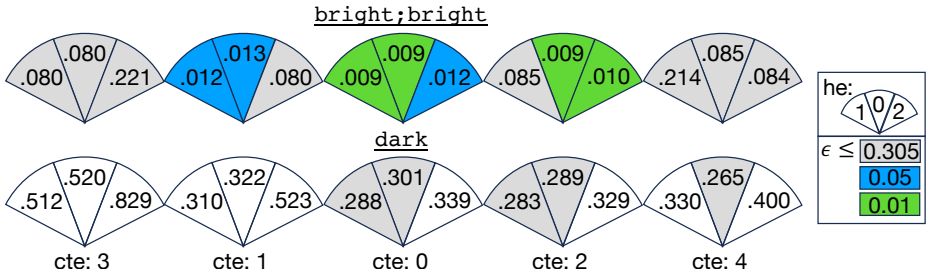


Fig. 5: Error probability by initial point distribution for `bright;bright` and `dark` scenarios. Initial point distributions that ensure error probability at most 0.305, 0.05 and 0.01 are highlighted in gray, blue, and green, respectively.

Similarly, we observe that when started from an initial state distribution that satisfies $\phi^{f1tenth}_{nominal}$, the F1Tenth car achieves low worst case error probability (0.069) for the track configuration `r;r;r;r` but a relatively high worst case error probability (0.338) for `l;l;l;l`. A system designer could use this information to understand that it may be acceptable to deploy the car on a track that loops clockwise, but the system should be retooled before deployment on a counterclockwise loop.

Initial Distributions. Our forward analysis also shows how the initial state distribution affects system-level safety. This effect is best illustrated in the F1Tenth case study. Returning to Table 2, we observe that the precondition $\phi^{f1tenth}_{nominal}$ ensures the lowest error probability of any of the considered preconditions for each scenario that we consider. Some scenarios, namely namely `s;s;s;s` and `r;s;s;l` are relatively robust to less-than-ideal initial distributions and exhibit a similar worst-case error probability for the more lax precondition $\phi^{f1tenth}_{center}$ as they do for the strict $\phi^{f1tenth}_{nominal}$. Interestingly, we also observe that $\phi^{f1tenth}_{right}$, which requires the car to start on the right side of the track with probability >0.9 leads to extremely high error probability for the scenario `l;l;l;l` in which the car must navigate a counterclockwise loop. Our forward analysis can help a system designer discover which combinations of track segments and initial state distributions will permit safe deployment of the autonomous system. The results from our F1Tenth case study would cause a system designer to focus on improving performance on track configurations that include `left` turn segments.

Backward Analysis. For the *backward analysis*, we assume that the domain expert provides a scenario sequence along with a maximum allowable error probability ϵ for the entire execution of the scenario sequence. We then compute the summary $(\boldsymbol{A},\boldsymbol{b})$ of the scenario sequence and observe that the weakest precondition that ensures the error probability does not exceed ϵ is $\phi_\epsilon := \boldsymbol{x}\cdot\boldsymbol{b} \leq \epsilon$. We visualize these preconditions in Figure 5 by highlighting the set of point distributions that ensure error probabilities less than 0.305, 0.05, and 0.01 for the scenario sequence `bright;bright` and the (singleton) scenario sequence `dark`.

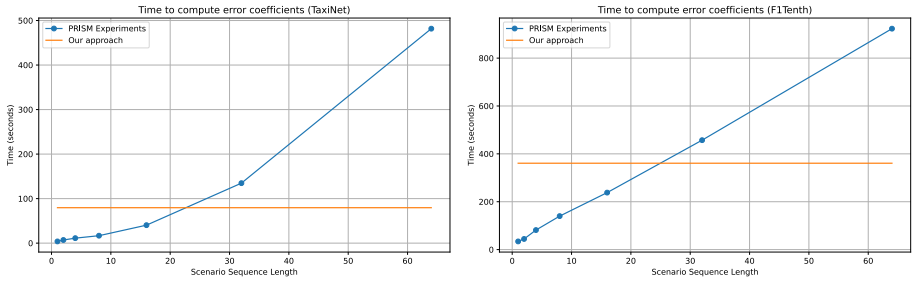


Fig. 6: Time needed to construct the error coefficients vector $\boldsymbol{b}$ using our compositional approach vs. a non-compositional application of PRISM's Experiments feature. For our approach, we report the time needed for the one-time summary generation, which can then be composed with minimal overhead for any sequence of scenarios. For PRISM experiments, we report the time needed to compute $\boldsymbol{b}$ for fixed scenario sequences of varying length. For TaxiNet we chose scenario sequences containing only `bright`, for F1Tenth we chose scenario sequences containing only `right`.

Computational Overhead. Both the *forward* and *backward* analyses are very computationally efficient, assuming that the summary of each atomic scenario has already been computed. The linear programs generated during our forward analysis can be quickly solved by Gurobi; it took less than 0.2 seconds to compute all the values in Table 2. The backward analysis requires no computation beyond the construction of $\boldsymbol{b}$. The upfront summary computation only needs to be performed once per atomic scenario. Generating all summaries for the TaxiNet and F1Tenth case studies took 79.56 seconds and 360.86 seconds, respectively.[8]

Our analyses consider *sets* of initial distributions, which are not naturally expressible in PRISM, so a direct comparison of computational efficiency vs. PRISM is not possible. However, one could use PRISM compute the error probability coefficients $\boldsymbol{b}$ by first building a monolithic DTMC that represents the execution of a particular scenario, then using either (1) PRISM's parametric model checking feature or (2) PRISM's experiments feature. Neither method scales gracefully: for our TaxiNet case study using parametric model checking to compute $\boldsymbol{b}$ for the scenario `bright` takes 6.912 seconds and exceeds PRISM's default 1GB of allocated RAM for `bright; bright`. PRISM's experiments feature can calculate $\boldsymbol{b}$ for `bright;bright` in 7.027s, however computing $\boldsymbol{b}$ for the long scenario $\texttt{bright}^{64}$ takes 481.692s much more than the $\sim$79.56 seconds needed by our approach. We visualize the computational scalability in Figure 6.

4.3 Using the Acceleration Rule

The acceleration rules introduced in Section 3.2 allow us to bound error probability of sequence scenarios, without putting an *a priori* bound on sequence length. Empirically, we find that our procedure to guess invariant preconditions works well for our case

[8] All durations are reported based on single-threaded execution using a commodity laptop with a 2 GHZ processor.

studies. Out of all of our singleton scenario sequences (`bright`, `dark`, `straight`, `left`, and **right**) and the ϵ values $\{0,0.01,0.02,...0.99\}$, the only instances where our guessed ϕ_ϵ did not satisfy the premise of Rule 2 were:

- TaxiNet's `dark` scenario with the ϵ values 0.27, 0.28, 0.29, and 0.30.
- F1Tenths's `left` scenario with $0.02 \leq \epsilon \leq 0.13$.
- F1Tenth's `right` scenario with ϵ values 0.00, 0.01, and 0.02.

Acceleration. Practically, the procedure yields an upper bound on the error probability of an unbounded sequential iteration of a single scenario. For example, for a desired *local* error probability of $\epsilon = 0.01$, we apply Rule 2 to TaxiNet's `bright` scenario with $\phi = \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{bright}} \leq 0.01$ to prove:

$$\{\phi\}(\texttt{bright})^k\{\phi\}\{1-(1-0.01)^k\}$$

for any $k \in \mathbb{N}$. This bound on error probability is parametric in the number of iterated executions of the `bright` scenario. Though not necessarily tight, such a parametric upper bound can help a system designer reason about how the cumulative probability of error increases during the execution of the autonomous system.

Rule 2 is particularly useful for autonomous systems that can maintain perfect safety. In our F1Tenth case study, we found that for the `straight` scenario, the precondition $\phi = \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{straight}} \leq 0$ is invariant, so we can prove:

$$\{\phi\}(\texttt{straight})^k\{\phi\}\{0\}$$

for any k. The precondition ϕ is actually quite permissive, in particular, it is more permissive than any of $\phi^{f1tenth}_{nominal}$, $\phi^{f1tenth}_{center}$, and $\phi^{f1tenth}_{right}$ introduced in *Section* 3.1.

Acceleration with Choice. For the more complex Rule 3, once again, we found that our proposed procedure successfully finds recurrent preconditions in the context of our case studies. For the TaxiNet case study, applying this procedure for $\epsilon = 0.306$ and $\phi_{0.306} := \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{bright}} \leq 0.306 \wedge \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{dark}} \leq 0.306$ allows us to prove:

$$\{\phi_{0.306}\}(\texttt{bright}|\texttt{dark})^k\{\phi_{0.306}\}\{1-(1-0.306)^k\}$$

this form of error bound is useful to a system designer who cannot predict the sequence of atomic scenarios the autonomous system will encounter during operation. We plot this upper bound on error probability against the true error probability of an adversarially chosen scenario sequence in Figure 7.

In the context of the F1Tenth case study, Rule 3 lets us derive an error bound for arbitrary track configurations. Importantly, we only need to collect datasets for each atomic scenario and our compositional reasoning allows us to generalize our guarantee to any track configuration formed from these atomic scenarios. Trying our procedure to guess an invariant precondition that satisfies the premise of Rule 3 for the scenarios `left`, `right`, and `straight`, we found that for $\epsilon \in \{0.74,0.75,...,0.99\}$

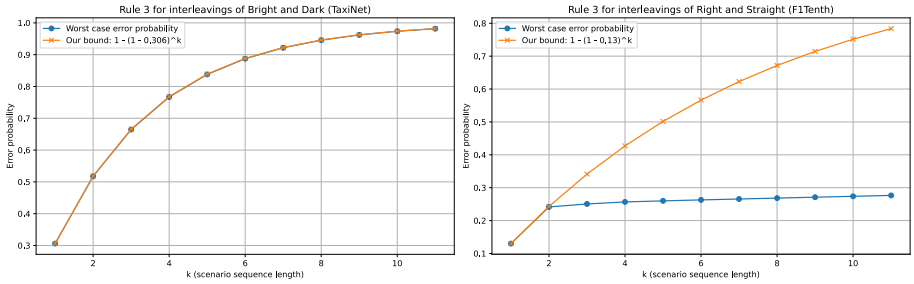


Fig. 7: The error bounds found by applications of Rule 3 vs. true worst-case error probability. For the worst case error probabilities, we consider all possible scensario interleavings and all starting distributions that satisfy the precondition ϕ_ϵ used in our application of Rule 3.

the precondition $\phi_\epsilon \coloneqq \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{left}} \leq \epsilon \wedge \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{right}} \leq \epsilon \wedge \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{straight}} \leq \epsilon$ is invariant and nontrivial. We can thus conclude:

$$\{\phi_{0.74}\}(\texttt{left}|\texttt{right}|\texttt{straight})^k\{\phi_{0.74}\}\{1-(1-0.74)^k\}$$

This bound allows extremely high error probability. A domain expert might restrict the deployment of the autonomous car to tracks formed from `right` and `straight` segments. Here, we find that $\phi_\epsilon \coloneqq \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{right}} \leq \epsilon \wedge \boldsymbol{x} \cdot \boldsymbol{b}_{\texttt{straight}} \leq \epsilon$ is nontrivial and invariant for $\epsilon \in \{0.13, 0.14, ..., 0.99\}$. In particular, we can conclude

$$\{\phi_{0.13}\}(\texttt{right}|\texttt{straight})^k\{\phi_{0.13}\}\{1-(1-0.13)^k\}$$

Concerning the relatively high error probabilities, we currently work with high-level abstractions of actual systems so computed error probabilities should be taken with a grain of salt. Still, our acceleration rules can reveal useful trends to system developers. Moreover, we expect lower error probabilities when applied to more accurate models of real-world systems; the analysis of fixed scenario sequences (Section 3.1) is tight.

5 Conclusion

Our verification framework decomposes the autonomous system and its operating conditions into scenarios to enable efficient probabilistic analysis. Our compositional proof rules enable system designers to obtain a bound on error probability that is parametric with respect to the length of an arbitrary interleaving of scenarios, which is useful for reasoning about the system under unpredictable changes in operating conditions. In future work, we plan to investigate more nuanced approaches to discover the invariant preconditions that enable application of our acceleration rule.

Building a probabilistic abstraction of perception with respect to a discretized state space can introduce inaccuracies. In our work, we avoid this concern by assuming that the distribution of state estimates yielded by the learning-enabled component is uniquely determined by the current environment condition and (discrete) state. In future work we plan to develop (possibly symbolic) abstractions of perception for continuous-state system models and to account for distribution shifts that are continuous, instead of discrete as we do here.

5.1 Acknowledgments.

This research was conducted during the first author's internship at the NASA Ames Research Center. This research was partially supported by NSF award SLES 2331783 and by a gift from AWS AI to Penn Engineering's ASSET Center for research in Trustworthy AI.

References

1. Arjomand Bigdeli, A., Mata, A., Bak, S.: Verification of neural network control systems in continuous time. 7th Symposium on AI Verification (SAIV) (2024)
2. Astorga, A., Hsieh, C., Madhusudan, P., Mitra, S.: Perception contracts for safety of ml-enabled systems. Proc. ACM Program. Lang. **7**(OOPSLA2), 2196–2223 (2023). https://doi.org/10.1145/3622875, https://doi.org/10.1145/3622875
3. Badithela, A., Wongpiromsarn, T., Murray, R.M.: Leveraging classification metrics for quantitative system-level analysis with temporal logic specifications. In: 2021 60th IEEE Conference on Decision and Control (CDC). pp. 564–571. IEEE (2021)
4. Badithela, A., Wongpiromsarn, T., Murray, R.M.: Evaluation metrics of object detection for quantitative system-level analysis of safety-critical autonomous systems. In: 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 8651–8658. IEEE (2023)
5. Cai, F., Fan, C., Bak, S.: Scalable surrogate verification of image-based neural network control systems using composition and unrolling (2024)
6. Calinescu, R., Imrie, C., Mangal, R., Pasareanu, C.S., Santana, M.A., Vázquez, G.: Discrete-event controller synthesis for autonomous systems with deep-learning perception components. CoRR **abs/2202.03360** (2022), https://arxiv.org/abs/2202.03360
7. Calinescu, R., Imrie, C., Mangal, R., Rodrigues, G.N., Păsăreanu, C., Santana, M.A., Vázquez, G.: Controller synthesis for autonomous systems with deep-learning perception components. IEEE Transactions on Software Engineering pp. 1–22 (2024). https://doi.org/10.1109/TSE.2024.3385378
8. Cruz, U.S., Shoukry, Y.: Certified vision-based state estimation for autonomous landing systems using reachability analysis. In: 2023 62nd IEEE Conference on Decision and Control (CDC). pp. 6052–6057 (2023). https://doi.org/10.1109/CDC49753.2023.10384107
9. De Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008), http://dl.acm.org/citation.cfm?id=1792734.1792766
10. Fremont, D.J., Chiu, J., Margineantu, D.D., Osipychev, D., Seshia, S.A.: Formal analysis and redesign of a neural network-based aircraft taxiing system with VerifAI. In: 32nd International Conference on Computer Aided Verification (CAV) (Jul 2020)
11. Fremont, D.J., Kim, E., Pant, Y.V., Seshia, S.A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q., Mehta, S.: Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In: 23rd IEEE International Conference on Intelligent Transportation Systems (ITSC) (Sep 2020)
12. Grigorescu, S.M., Trasnea, B., Cocias, T.T., Macesanu, G.: A survey of deep learning techniques for autonomous driving. CoRR **abs/1910.07738** (2019)
13. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2024), https://www.gurobi.com

14. Habeeb, P., Deka, N., D'Souza, D., Lodaya, K., Prabhakar, P.: Verification of camera-based autonomous systems. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **42**(10), 3450–3463 (2023). https://doi.org/10.1109/TCAD.2023.3240131, https://doi.org/10.1109/TCAD.2023.3240131
15. Habeeb, P., D'Souza, D., Lodaya, K., Prabhakar, P.: Interval image abstraction for verification of camera-based autonomous systems. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. (2024)
16. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker Storm. International Journal on Software Tools for Technology Transfer **24**(4), 589–610 (Aug 2022)
17. Hsieh, C., Koh, Y., Li, Y., Mitra, S.: Assuring safety of vision-based swarm formation control. In: American Control Conference (ACC) (2024)
18. Hsieh, C., Li, Y., Sun, D., Joshi, K., Misailovic, S., Mitra, S.: Verifying controllers with vision-based perception using safe approximate abstractions. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **41**(11), 4205–4216 (2022). https://doi.org/10.1109/TCAD.2022.3197508, https://doi.org/10.1109/TCAD.2022.3197508
19. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Wu, M., Yi, X.: A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. Computer Science Review **37**, 100270 (2020)
20. Ivanov, R., Jothimurugan, K., Hsu, S., Vaidya, S., Alur, R., Bastani, O.: Compositional learning and verification of neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(5s) (Sep 2021). https://doi.org/10.1145/3477023, https://doi.org/10.1145/3477023
21. Kadron, I.B., Gopinath, D., Pasareanu, C.S., Yu, H.: Case study: Analysis of autonomous center line tracking neural networks. In: Bloem, R., Dimitrova, R., Fan, C., Sharygina, N. (eds.) Software Verification - 13th International Conference, VSTTE 2021, New Haven, CT, USA, October 18-19, 2021, and 14th International Workshop, NSV 2021, Los Angeles, CA, USA, July 18-19, 2021, Revised Selected Papers. pp. 104–121. Lecture Notes in Computer Science (2021)
22. Katz, S.M., Corso, A.L., Strong, C.A., Kochenderfer, M.J.: Verification of image-based neural network controllers using generative models. Journal of Aerospace Information Systems **19**(9), 574–584 (2022)
23. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
24. Li, J., Nuzzo, P., Sangiovanni-Vincentelli, A., Xi, Y., Li, D.: Stochastic assume-guarantee contracts for cyber-physical system design under probabilistic requirements (2017), https://arxiv.org/abs/1705.09316
25. Li, Y., Yang, B.C., Jia, Y., Zhuang, D., Mitra, S.: Refining perception contracts: Case studies in vision-based safe auto-landing (2023)
26. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Trans. Program. Lang. Syst. **18**(3), 325–353 (May 1996). https://doi.org/10.1145/229542.229547, https://doi.org/10.1145/229542.229547
27. O'Kelly, M., Zheng, H., Karthik, D., Mangharam, R.: F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. In: Escalante, H.J., Hadsell, R. (eds.) Proceedings of the NeurIPS 2019 Competition and Demonstration Track. Proceedings of Machine Learning Research, vol. 123, pp. 77–89. PMLR (08–14 Dec 2020), https://proceedings.mlr.press/v123/o-kelly20a.html

28. Păsăreanu, C.S., Mangal, R., Gopinath, D., Getir Yaman, S., Imrie, C., Calinescu, R., Yu, H.: Closed-loop analysis of vision-based autonomous systems: A case study. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. pp. 289–303. Springer Nature Switzerland, Cham (2023)
29. Pasareanu, C.S., Mangal, R., Gopinath, D., Yu, H.: Assumption generation for learning-enabled autonomous systems. In: Katsaros, P., Nenzi, L. (eds.) Runtime Verification - 23rd International Conference, RV 2023, Thessaloniki, Greece, October 3-6, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14245, pp. 3–22. Springer (2023). https://doi.org/10.1007/978-3-031-44267-4_1, https://doi.org/10.1007/978-3-031-44267-4_1
30. Santa Cruz, U., Shoukry, Y.: Nnlander-verif: A neural network formal verification framework for vision-based autonomous aircraft landing. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds.) NASA Formal Methods. pp. 213–230. Springer International Publishing, Cham (2022)
31. Sun, D., Yang, B., Mitra, S.: Learning-based inverse perception contracts and applications. In: International Conference on Robotics and Automation (2024)
32. Tabernik, D., Skocaj, D.: Deep learning for large-scale traffic-sign detection and recognition. CoRR **abs/1904.00649** (2019)
33. Waite, T., Robey, A., Hamed, H., Pappas, G.J., Ivanov, R.: Data-driven modeling and verification of perception-based autonomous systems (2023)
34. Watanabe, K., Eberhart, C., Asada, K., Hasuo, I.: Compositional probabilistic model checking with string diagrams of mdps. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. pp. 40–61. Springer Nature Switzerland, Cham (2023)
35. Yalcinkaya, B., Torfah, H., Fremont, D.J., Seshia, S.A.: Compositional simulation-based analysis of ai-based autonomous systems for markovian specifications. In: Katsaros, P., Nenzi, L. (eds.) Runtime Verification. pp. 191–212. Springer Nature Switzerland, Cham (2023)

A Soundness Proofs

We prove soundness of the rules introduced in Section 3.2 with respect to the definition of sequential composition of *summaries* found at the end of Section 2.2.

$$\frac{\{\phi\}C\{\psi\}\{\epsilon\},\{\phi'\}C'\{\psi'\}\{\epsilon'\},\psi \Longrightarrow \phi'}{\{\phi\}CC'\{\psi'\}\{1-(1-\epsilon)(1-\epsilon')\}}\ \textsc{Rule}\ 1$$

Proof. Let $C \equiv (\boldsymbol{A},\boldsymbol{b})$ and $C' \equiv (\boldsymbol{A}',\boldsymbol{b}')$.
By the definition of sequential composition of scenarios, we know $CC' \equiv (\boldsymbol{A}\boldsymbol{A}',\boldsymbol{b}+\boldsymbol{A}\boldsymbol{b}$.
We prove the two obligations:

- To show that $\phi(\boldsymbol{x}) \Rightarrow \psi'(norm(\boldsymbol{x}\boldsymbol{A}\boldsymbol{A}'))$ we fix $\boldsymbol{x}$ w.l.o.g. and apply the first premise to obtain $\psi(norm(\boldsymbol{x}\boldsymbol{A}))$. We then apply the third premise to obtain $\phi(norm(\boldsymbol{x}\boldsymbol{A}))$ and the second premise to obtain $\psi'(norm(norm(\boldsymbol{x}\boldsymbol{A})\boldsymbol{A}'))$. Expansion of the definition of *norm* and algebraic manipulation completes this branch of the proof.
- To show that $\phi(\boldsymbol{x}) \rightarrow \boldsymbol{x}\boldsymbol{b}+\boldsymbol{x}\boldsymbol{A}\boldsymbol{b}' \leq 1-(1-\epsilon)(1-\epsilon')$ we fix $\boldsymbol{x}$ w.l.o.g. Next we observe that $\boldsymbol{x}\boldsymbol{b}+\boldsymbol{x}\boldsymbol{A}\boldsymbol{b}' = \boldsymbol{x}\boldsymbol{b}+(1-\boldsymbol{x}\boldsymbol{b})\boldsymbol{x}\boldsymbol{A}\boldsymbol{b}'$ by the definition of norm and of a summary. We can rewrite this as $1-(1-\boldsymbol{x}\boldsymbol{b})(1-norm(\boldsymbol{x}\boldsymbol{A})\boldsymbol{b}')$. So it suffices to show $1-(1-\boldsymbol{x}\boldsymbol{b})(1-norm(\boldsymbol{x}\boldsymbol{A})\boldsymbol{b}') \leq 1-(1-\epsilon)(1-\epsilon')$. Premise 1 lets us

bound $\boldsymbol{xb} \leq \epsilon$ and premise 2 lets us bound $norm(\boldsymbol{xA})\boldsymbol{b}' \leq \epsilon'$ so we can bound $(1-\boldsymbol{xb}) \geq 1-\epsilon$ and $1-norm(\boldsymbol{xA})\boldsymbol{b}' \geq 1-\epsilon'$. Together these imply the desired bound.

$$\frac{\{\phi\}C\{\phi\}\{\epsilon\}}{\{\phi\}C^k\{\phi\}\{1-(1-\epsilon)^k\}}\ \textsc{Rule 2}$$

Proof. Direct application of Rule 3, which we prove below.

$$\frac{\{\phi\}C_1\{\phi\}\{\epsilon\},...,\{\phi\}C_\ell\{\phi\}\{\epsilon\}}{\{\phi\}(C_1|...|C_m)^k\{\phi\}\{\epsilon\}\}}\ \textsc{Rule 3}$$

Proof. By induction over k. The case $k=1$ is immediate and the case $k=2$ follows by Rule 1. For the inductive step, assume w.l.o.g. that the conclusion of this instance of Rule 3 is $\{\phi\}C_{i_1}...C_{i_{k-1}}C_{i_k}\{\phi\}\{1-(1-\epsilon)^k\}$. By the inductive hypothesis, we know $\{\phi\}C_{i_1}...C_{i_{k-1}}\{\phi\}\{1-(1-\epsilon)^{(k-1)}\}$, so we can apply Rule 1 to complete the proof.

B PRISM Model Construction

We provide additional details about the our PRISM model construction introduced in Section 4.1 and adapted from [28]. Assume that we are modeling an autonomous system in which the dynamics uses state set S, the perception model outputs state estimates from Y, the controller issues commands from U, and there are m distinct scenarios which we will refer to by the counting numbers $\{1,...,m\}$. For each scenario i, we denote the probabilistic abstraction of perception as $\alpha_i : S \rightarrow \mathbf{Dist}(Y)$ the time horizon as $H_i \in \mathbb{N}$. Let H_{max} denote $\max_{i\in\{1,...,m\}}(H_i)$.

Our PRISM model has a set of state variables V which can be partitioned as $V = V_S \sqcup V_Y \sqcup V_U \sqcup \{\texttt{pc},\texttt{scenario},\texttt{timer}\}$. Each valuation V_S, V_Y, or V_U uniquely determines an element of S, Y, or U, respectively. For each $s \in S$, we write s as shorthand for the PRISM state predicate that evaluates to true exactly when the current valuation of V_S determines s.

Our construction introduces a *program counter* variable `pc` with range $\{0,1,2\}$, a variable `scenario` with range $\{1,...,m\}$, and a variable `timer` with range $\{1,...,H_{max}\}$. PRISM indexes timesteps starting at 0. We maintain `pc` and `timer` such that at PRISM timestep t, we have $\texttt{pc} = mod(t,3)$ and $\texttt{timer} = \min(quotient(t,3)+1, H_{max})$. The other variables are updated as follows:

- **Perception:** When $pc=0$, the valuation of V_Y will be updated according to the valuation of V_S and on the next step. This update's transition probabilities are given by the *probabilistic abstraction* for the current scenario, which we represent using the value of `scenario`. The value of `scenario` is never updated, later on we will explain how we initialize the value of scenario to compute each summary.
- **Controller:** When $pc = 1$, the valuation of V_U is updated according to the valuation of V_Y.
- **Dynamics:** When $pc = 2$, the valuation of V_S is updated according to the valuation of V_U and the current valuation of V_S.

We can now apply the summary generation technique detailed in Section 4.1 to this model.

B.1 Additional TaxiNet Details

For the purpose of our analysis, we discretize TaxiNet's outputs and treat it as a classifier. cte ∈ [-8.0 m, 8.0 m] and he ∈ [-35.0 deg, 35.0 deg] are translated into cte ∈ 0, 1, 2, 3, 4 and he ∈ 0, 1, 2 as shown below.

$$cte = \begin{cases} 3 \text{ if } -8.0 \text{ m } <= cte < -4.8 \text{ m} \\ 1 \text{ if } -4.8 \text{ m } <= cte < -1.6 \text{ m} \\ 0 \text{ if } -1.6 \text{ m } <= cte <= 1.6 \text{ m} \\ 2 \text{ if } 1.6 \text{ m } < cte <= 4.8 \text{ m} \\ 4 \text{ if } 4.8 \text{ m } < cte <= 8.0 \text{ m} \end{cases} \qquad he = \begin{cases} 1 \text{ if } -35.0\ deg <= he < -11.67\ deg \\ 0 \text{ if } -11.67\ deg <= he <= 11.66\ deg \\ 2 \text{ if } 11.66\ deg < he <= 35.0\ deg \end{cases}$$

Yielding the discretized set of system states $\underline{S} := [0..4] \times [0..2]$. Any airplane position with in which the magnitude of the cross-track error exceeds 8m or the magnitude of the heading error exceeds 35° represents the error state s_{err}.

B.2 Additional F1Tenth Details

We provide additional details about the F1Tenth case study. At each timestep, the car's heading *heading* is updated based on the control input u. Then, the car moves one grid-position in the direction of the updated heading. The dynamics function $f: E \times S \times U$ also accounts for walls and transitions between track segments.

We define one scenario (e,30) for each track segment e. We wish for the scenario (e,30) to end when the car reaches the end of the current track segment, which we represent as F_e. This requires careful definition of the dynamics function, since the car may take more or less than the horizon value of 30 timesteps to reach F_e. We present pseudocode for the dynamics function in f. At a high level, we introduce a time limit 30 and transition to s_{err} if the car does not reach F_e in fewer than 30 steps. To address cars that reach F_e in less than 30 timesteps, we stop updating the position and heading of the car until the start of the next scenario.

We built a *Controller* $g: E \times Y \to U$. The dependence on Y corresponds to the autonomous system being equipped with an oracle that detects which track segment is being navigated currently. This controller can navigate each track segment safely when it recieves the ground-truth state estimate at each timestep and begins from an initial state with heading $heading = 0$. When operating under these conditions, the controller completes each track segment with its heading such that it starts the next segment with $heading = 0$. This implies that the controller can safely navigate any sequence of track segments assuming perfect behavior of the learning-enabled component.

Algorithm 2: F1Tenth Dynamics

Input: Track segment $e \in E$; Current state $s \in S$; control input $u \in U$
Output: Next state $s' \in S$

```
if s = s_err then
    return s_err
else
    (pos_side, pos_front, heading, τ) ← s;
    if τ < 30 then
        heading' ← mod(heading + u, 8);
        pos'_side ← pos_side − 𝟙_{1,2,3}(heading) + 𝟙_{5,6,7}(heading);
        pos'_front ← pos_front − 𝟙_{0,1,7}(heading) + 𝟙_{3,4,5}(heading);
        (pos'_side, pos'_front) ← f^pos((pos_side, pos_front, heading), u);
        else if (pos'_side, pos'_front) ∈ Track ∪ Track_e ∪ F_e then
            return (pos'_side, pos'_front, heading', τ + 1);
        else
            return s_err;
    else if (pos_side, pos_front) ∈ F_e then
        if e = left then
            return (3 − pos_front, 15, mod(heading − 2, 8), 1);
        else if e = right then
            return (pos_front − 3, 15, mod(heading + 2, 8), 1);
        else
            return (pos_side, 15, heading, 1);
    else
        return s_err
```