Compositional Falsification of Cyber-Physical Systems with Machine Learning Components*

Tommaso Dreossi[1], Alexandre Donzé[2], and Sanjit A. Seshia[1]

[1] University of California, Berkeley
{dreossi,sseshia}@berkeley.edu
[2] Decyphir, Inc.
alex.r.donze@gmail.com

Abstract. Cyber-physical systems (CPS), such as automotive systems, are starting to include sophisticated machine learning (ML) components. Their correctness, therefore, depends on properties of the inner ML modules. While learning algorithms aim to generalize from examples, they are only as good as the examples provided, and recent efforts have shown that they can produce inconsistent output under small adversarial perturbations. This raises the question: can the output from learning components can lead to a failure of the entire CPS? In this work, we address this question by formulating it as a problem of falsifying signal temporal logic (STL) specifications for CPS with ML components. We propose a compositional falsification framework where a temporal logic falsifier and a machine learning analyzer cooperate with the aim of finding falsifying executions of the considered model. The efficacy of the proposed technique is shown on an automatic emergency braking system model with a perception component based on deep neural networks.

Keywords: Cyber-physical systems, machine learning, falsification, temporal logic

1 Introduction

Over the last decade, machine learning (ML) algorithms have achieved impressive results providing solutions to practical large-scale problems (see, e.g., [3, 15, 11, 9]). Not surprisingly, ML is being used in cyber-physical systems (CPS) — systems that are integrations of computation with physical processes. For example, semi-autonomous vehicles employ Adaptive Cruise Controllers (ACC) or Lane Keeping Assist Systems (LKAS) that rely heavily on image classifiers providing input to the software controlling electric and mechanical subsystems (see, e.g., [4]). The safety-critical nature of such systems involving ML raises the need for formal methods [19]. In particular, how do we systematically find bugs in such systems?

* This work is funded in part by the DARPA BRASS program under agreement number FA8750-16-C-0043, NSF grants CNS-1646208, CNS-1545126, and CCF-1139138, and by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. The second author did much of the work while affiliated with UC Berkeley.

We formulate this question as the falsification problem for CPS models with ML components (CPSML): given a formal specification φ in signal temporal logic (STL) [13], and a CPSML model M, find an input for which M does *not* satisfy φ. A falsifying input generates a counterexample trace that reveals a bug. To solve this problem, multiple challenges must be tackled. First, the input space to be searched can be intractable. For instance, a simple model of a semi-autonomous car already involves several control signals (e.g., the angle of the acceleration pedal, steering angle) and other sensor input (e.g., images captured by a camera). Second, CPSML are often designed using languages (such as C, C++, or Simulink), for which clear semantics are not given, and involve third-party components that are opaque or poorly-specified. This obstructs the development of formal methods for the analysis of CPSML models and may force one to treat them as gray/black-boxes. Third, the formal verification of ML components is a difficult, and somewhat ill-posed problem due to the complexity of the underlying ML algorithms, large feature spaces, and the lack of consensus on a formal definition of correctness [19]. Hence, we need a technique to systematically analyze ML components within the context of a CPS.

In this paper, we propose a framework for the falsification of CPSML addressing the issues described above. Our technique is compositional in that it divides the search space for falsification into that of the ML component and of the remainder of the system, while establishing a connection between the two. The obtained subspaces are respectively analyzed by a temporal logic falsifier and an ML analyzer that cooperate. This cooperation mainly comprises a series of input space projections, leads to small subsets in which counterexamples are easier to find. Further, our technique can handle any machine learning technique, including the methods based on deep neural networks [9] that have proved effective in many recent applications. The proposed ML analyzer identifies sets of misclassifying features, i.e., inputs that "fool" the ML algorithm. The analysis is performed by considering subsets of parameterized features spaces that are used to approximate the ML components by simpler functions. The information gathered by the temporal logic falsifier and the ML analyzer together reduce the search space, providing an efficient approach to falsification for CPSML models.

Example 1. As an illustrative example, let us consider a simple model of an Automatic Emergency Braking System (AEBS) as a closed-loop control system composed of a controller (automatic brake), a plant (car transmission), and a sensor (obstacle detector) (see Figure 1). The controller regulates the acceleration and braking of the plant using the velocity of the subject (ego) vehicle and the distance between it and an obstacle. The sensor used to detect the obstacle includes a camera along with an image classifier. In general, this sensor can provide

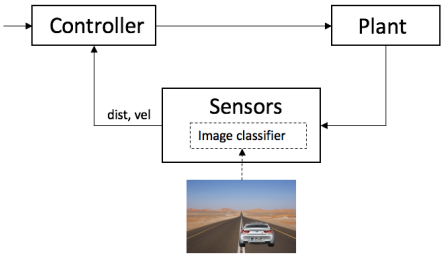


Fig. 1: Automatic Emergency Braking System. An image classifier is used to perceive vehicles in the frame of view.

noisy measurements due to incorrect image classifications which in turn can affect the correctness of the overall system.

Suppose we want to verify whether the distance between the subject vehicle and a preceding obstacle is always larger than 5 meters. Such a verification requires the exploration of an intractable input space comprising the control inputs (e.g., acceleration and braking pedal angles) and the ML component's feature space (e.g., all the possible pictures observable by the camera). Note that feature space of RGB 1000×600px pictures for an image classifier contains $256^{1000\times600\times3}$ elements. □

At first, the input space of the model described in Example 1 appears intractable. However, we can observe some interesting aspects of the relationship between the "pure CPS" input space and its ML feature space:

1. Under the assumption of "perfect ML components" (i.e., all feature vectors are correctly classified), we can study the CPSML model on a lower-dimensional input space (the "pure CPS" one) and identify regions of values that satisfy the specification but might be affected by the malfunctioning of some ML modules;
2. Instead of verifying the ML components on their whole feature spaces, we can focus only on those features related to the non-robust input values identified in the previous step, and
3. If we are able to determine misclassifications on the restricted feature space, then we can relate them back to CPSML input space, thus focusing the falsification on a smaller input space.

These three observations constitute the core idea of the compositional falsification method proposed in this paper. Specifically, we use a temporal logic falsifier, Breach [5], in Steps (1) and (3) to partition a given input set into values that do and do not satisfy a given specification, and an ML analyzer in Step (2) to determine subsets of feature vectors that are misclassified by the ML components.

The proposed method, however, presents certain challenges that need to be addressed. First, we need to construct a validity domain of a specification against a CPSML model with (assumed) correct ML components. Second, we need a method to relate the non-robust input areas to the feature space of the ML modules. Third, we need to systematically analyze the ML components with the goal of finding feature vectors leading to misclassifications. We describe in detail in Sections 3 and 4 how we tackle these challenges.

In summary, the main contributions of this paper are:

- A compositional framework for the falsification of temporal logic properties of CPSML models that works for any machine learning classifier.
- A machine learning analyzer that identifies misclassifications leading to system-level property violations, based on two main ideas:
  - An input space parameterization used to abstract the feature space and relate it to the CPSML input space, and
  - A classifier approximation method used to identify misclassifications that can lead to unsafe executions of the CPSML.

In Sec. 5, we demonstrate the effectiveness of our approach on an Automatic Emergency Braking System (AEBS) involving an image classifier for obstacle detection based on deep neural networks using leading software packages Caffe [11] and TensorFlow [14].

Related Work

The verification of both CPS and ML algorithms have attracted several research efforts, and we focus here on the most closely related work. Techniques for the falsification of temporal logic specifications against CPS models have been implemented based on nonlinear optimization methods and stochastic search strategies (e.g., Breach [5], S-TaLiRo [2], RRT-REX [6], C2E2 [7]). While the verification of ML programs is less well-defined [19], recent efforts [20] show how even well trained neural networks can be sensitive to small adversarial perturbations, i.e., small intentional modifications that lead the network to misclassify the altered input with large confidence. Other efforts have tried to characterize the correctness of neural networks in terms of risk [22] (i.e., probability of misclassifying a given input) or robustness [8] (i.e., the minimal perturbation leading to a misclassification), while others proposed methods to generate pictures [17] or perturbations [16, 10] in such a way to "fool" neural networks. To the best of our knowledge, our work is the first to address the verification of temporal logic properties of CPSML—the combination of CPS and ML systems.

2 Background

2.1 CPSML Models

In this work, we consider models of cyber-physical systems with machine learning components (CPSML). We assume that a system model is given as a gray-box simulator defined as a tuple $M = (S, U, sim)$, where S is a set of system states, U is a set of input values, and $sim : S \times U \times T \to S$ is a simulator that maps a state $\mathbf{s}(t_k) \in S$ and input value $\mathbf{u}(t_k) \in U$ at time $t_k \in T$ to a new state $\mathbf{s}(t_{k+1}) = sim(\mathbf{s}(t_k), \mathbf{u}(t_k), t_k)$, where $t_{k+1} = t_k + \Delta_k$ for a time-step $\Delta_k \in \mathbb{Q}_{>0}$.

Given an initial time $t_0 \in T$, an initial state $\mathbf{s}(t_0) \in S$, a sequence of time-steps $\Delta_0, \ldots, \Delta_n \in \mathbb{Q}_{>0}$, and a sequence of input values $\mathbf{u}(t_0), \ldots, \mathbf{u}(t_n) \in U$, a simulation trace of the model $M = (S, U, sim)$ is a sequence:

$$(t_0, \mathbf{s}(t_0), \mathbf{u}(t_0)), (t_1, \mathbf{s}(t_1), \mathbf{u}(t_1)), \ldots, (t_n, \mathbf{s}(t_n), \mathbf{u}(t_n))$$

where $\mathbf{s}(t_{k+1}) = sim(\mathbf{s}(t_k), \mathbf{u}(t_k), \Delta_k)$ and $t_{k+1} = t_k + \Delta_k$ for $k = 0, \ldots, n$.

The gray-box aspect of the CPSML model is that we assume some knowledge of the internal ML components. Specifically, these components, termed *classifiers*, are functions $f : X \to Y$ that assign to their input *feature vector* $\mathbf{x} \in X$ a *label* $y \in Y$, where X and Y are a feature and label space, respectively. Without loss of generality, we focus on binary classifiers whose label space is $Y = \{0, 1\}$. A ML algorithm selects a classifier using a training set $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(m)}, y^{(m)})\}$ where the $(\mathbf{x}^{(i)}, y^{(i)})$ are labeled examples with $\mathbf{x}^{(i)} \in X$ and $y^{(i)} \in Y$, for $i = 1, \ldots, m$. The quality of a classifier

can be estimated on a test set of examples comparing the classifier predictions against the labels of the examples. Precisely, for a given test set $T = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(l)}, y^{(l)})\}$, the number of false positives $fp_f(T)$ and false negatives $fn_f(T)$ of a classifier f on T are defined as:

$$\begin{aligned} fp_f(T) &= \mid \{\mathbf{x}^{(i)} \in T \mid f(\mathbf{x}^{(i)}) = 1 \text{ and } y^{(i)} = 0\} \mid \\ fn_f(T) &= \mid \{\mathbf{x}^{(i)} \in T \mid f(\mathbf{x}^{(i)}) = 0 \text{ and } y^{(i)} = 1\} \mid \end{aligned} \tag{1}$$

The error rate of f on T is given by:

$$err_f(T) = (fp_f(T) + fn_f(T))/l \tag{2}$$

A low error rate implies good predictions of the classifier f on the test set T.

2.2 Signal Temporal Logic

We consider Signal Temporal Logic [13] (STL) as the language to specify properties to be verified against a CPSML model. STL is an extension of linear temporal logic (LTL) suitable for the specification of properties of CPS.

A *signal* is a function $s : D \to S$, with $D \subseteq \mathbb{R}_{\geq 0}$ an interval and either $S \subseteq \mathbb{B}$ or $S \subseteq \mathbb{R}$, where $\mathbb{B} = \{\top, \bot\}$ and $\mathbb{R}$ is the set of reals. Signals defined on $\mathbb{B}$ are called *booleans*, while those on $\mathbb{R}$ are said *real-valued*. A *trace* $w = \{s_1, \ldots, s_n\}$ is a finite set of real-valued signals defined over the same interval D.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a finite set of predicates $\sigma_i : \mathbb{R}^n \to \mathbb{B}$, with $\sigma_i \equiv p_i(x_1, \ldots, x_n) \lhd 0$, $\lhd \in \{<, \leq\}$, and $p_i : \mathbb{R}^n \to \mathbb{R}$ a function in the variables $x_1, \ldots, x_n$.

An STL formula is defined by the following grammar:

$$\varphi := \sigma \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi U_I \varphi \tag{3}$$

where $\sigma \in \Sigma$ is a predicate and $I \subset \mathbb{R}_{\geq 0}$ is a closed non-singular interval. Other common temporal operators can be defined as syntactic abbreviations in the usual way, like for instance $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \varphi_2)$, $F_I\varphi := \top U_I \varphi$, or $G_I\varphi := \neg F_I \neg\varphi$. Given a $t \in \mathbb{R}_{\geq 0}$, a shifted interval I is defined as $t + I = \{t + t' \mid t' \in I\}$.

Definition 1 (Qualitative semantics). *Let w be a trace, $t \in \mathbb{R}_{\geq 0}$, and φ be an STL formula. The* qualitative semantics *of φ is inductively defined as follows:*

$$\begin{aligned} w, t &\models \sigma \text{ iff } \sigma(w(t)) \text{ is true} \\ w, t &\models \neg\varphi \text{ iff } w, t \not\models \varphi \\ w, t &\models \varphi_1 \wedge \varphi_2 \text{ iff } w, t \models \varphi_1 \text{ and } w, t \models \varphi_2 \\ w, t &\models \varphi_1 U_I \varphi_2 \text{ iff } \exists t' \in t + I \text{ s.t. } w, t' \models \varphi_2 \text{ and } \forall t'' \in [t, t'], w, t'' \models \varphi_1 \end{aligned} \tag{4}$$

A trace w satisfies a formula φ if and only if $w, 0 \models \varphi$, in short $w \models \varphi$. For given signal w, time instant $t \in \mathbb{R}_{\geq 0}$, and STL formula φ, the *satisfaction signal* $\mathcal{X}(w, t, \varphi)$ is $\top$ if $w, t \models \varphi$, $\bot$ otherwise.

Definition 2 (Quantitative semantics). *Let w be a trace, $t \in \mathbb{R}_{\geq 0}$, and φ be an STL formula. The* quantitative semantics *of φ is defined as follows:*

$$\begin{aligned} \rho(p(x_1,\dots,x_n) \lhd 0, w, t) &= p(w(t)) \text{ with } \lhd \in \{<, \leq\} \\ \rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\ \rho(\varphi_1 \wedge \varphi_2, w, t) &= \min(\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)) \\ \rho(\varphi_1 U_I \varphi_2, w, t) &= \sup_{t' \in t+I} \min(\rho(\varphi_2, w, t'), \inf_{t'' [t,t']} \rho(\varphi_1, w, t'')) \end{aligned} \tag{5}$$

The *robustness* of a formula φ with respect to a trace w is the signal $\rho(\varphi, w, \cdot)$.

3 Compositional Falsification Framework

In this section, we formalize the falsification problem for STL specifications against CPSML models, define our compositional falsification framework, and show its functionality on the AEBS system of Example 1.

Definition 3 (Falsification of CPSML). *Given a model $M = (S, U, sim)$ and an STL specification φ, find an initial state $\mathbf{s}(t_0) \in S$ and a sequence of input values $\mathbf{u} = \mathbf{u}(t_0), \dots, \mathbf{u}(t_n) \in U$ such that the trace of states $w = \mathbf{s}(t_0), \dots, \mathbf{s}(t_n)$ generated by the simulation of M from $\mathbf{s}(t_0) \in S$ under $\mathbf{u}$ does not satisfy φ, i.e., $w \not\models \varphi$. We refer to such $(\mathbf{s}(t_0), \mathbf{u})$ as* counterexamples *for φ. The problem of finding a counterexample is often called* falsification problem.

We now present the compositional framework for the falsification of STL formulas against CPSML models. Intuitively, the proposed method decomposes a given model into two abstractions: a version of the CPSML model under the assumption of perfectly correct ML modules and its actual ML components. The two abstractions are separately analyzed, the first by a temporal logic falsifier that builds the validity domain with respect to the given specification, the second by an ML analyzer that identifies sets of feature vectors that are misclassified by the ML components. Finally, the results of the two analysis are composed and projected back to a targeted input subspace of the original CPSML model where counterexamples can be found by invoking a temporal logic falsifier. Let us formalize this procedure.

Let $M = (S, U, sim)$ be a CPSML model and φ be an STL specification. Let M' be a version of M with perfectly behaving ML components, that is, every feature vector of the ML feature spaces is correctly classified. Let us denote by ml the isolated ML components of the model M.

Under the assumption of correct ML components, the lower-dimensional input space of M' can be analyzed by constructing the validity domain of φ, that is the partition of the input space into the sets U_φ and $U_{\neg\varphi}$ that do and do not satisfy φ, respectively. Note that considering the original model M, a possible misclassification of the ML components ml might affect the elements of U_φ and $U_{\neg\varphi}$. In particular, we are interested in the elements of U_φ that, due to misclassifications of ml, do not satisfy φ anymore. This corresponds to analyze the behavior of the ML components ml on the input set U_φ. We refer to this step as the ML analysis, that can be seen as the procedure of finding a subset $U^{ml} \subseteq U_\varphi$

of input values that are misclassified by the ML components ml. It is important to note that the input space of the CPS model M' and the feature spaces of the ML modules ml are different, thus the ML analyzer must adapt and relate the two different spaces. This important step will be clarified in Section 4.

Finally, the intersection $U_\varphi \cap U^{ml}$ of the subsets identified by the decomposed analysis of the CPS model and its ML components targets a small set of input values that are misclassified by the ML modules and are likely to falsify φ. Thus, counterexamples in $U_\varphi \cap U^{ml} \subseteq U$ can be determined by invoking a temporal logic falsifier on φ against M.

Algorithm 1 CPSML falsification scheme

```
1: function CompFalsfy(M, φ)                          ▷ M CPSML, φ STL specification
2:     [M', ml] ←Decompose(M)                         ▷ M' exact ML, ml ML modules
3:     [U_φ, U_¬φ] ←ValidityDomain(M', U, φ)          ▷ Validity domain of φ w.r.t. M'
4:     U^ml ← MLAnalysis(ml, U_φ)                     ▷ Find misclassified feature vectors
5:     U^ml_¬φ ←Falsify(M, U_φ ∩ U^ml, φ)             ▷ Falsify on targeted input
6:     return U_¬φ ∪ U^ml_¬φ
7: end function
```

The compositional falsification procedure is formalized in Algorithm 1. CompFalsfy receives as input a CPSML model M and an STL specification φ, and returns a set of falsifying counterexamples. At first, the algorithm decomposes M into M' and ml, where M' is an abstract version of M with perfectly working ML modules, and ml are the ML components of M (Line 2). Then, the validity domain of φ with respect to the abstraction M' is computed by ValidityDomain (Line 3) and subsets of input that are misclassified by ml are identified by MLAnalysis (Line 4). Finally, the targeted input set $U_\varphi \cap U^{ml}$, consisting in the intersection of the sets identified by the decomposed analysis, is searched by a temporal logic falsifier on the original model M (Line 5) and a collection of counterexamples is returned.

Example 2. Let us consider the model described in Example 1 and let us assume that the input space U of the model M consists of the initial velocity of the subject vehicle $vel(0)$, the initial distance between the vehicle and the proceeding obstacle $dist(0)$, and the set of pictures that can be captured by the camera. Let $\varphi := G_{[0,T]}(dist(t) \geq \tau)$ be a specification that requires the vehicle to be always farther than τ from the preceding obstacle. Instead of analyzing the whole input space U (including a vast number of pictures), we can adopt our compositional framework to target a specific subset of U. Let M' be the AEBS model with a perfectly working image classifier and ml be the actual classifier. We begin by computing the validity subsets U_φ and $U_{\neg\varphi}$ of φ against M', considering only $vel(0)$ and $dist(0)$ and assuming exact distance measurements during the simulation. Next, we analyze only the image classifier ml on pictures of obstacles whose distances fall in U_φ, say in $[d_m, d_M]$ (see Figure 2). Our ML analyzer generates only pictures of obstacles whose distances are in $[d_m, d_M]$, finds possible sets of images that are misclassified, and returns the corresponding distances that,

when projected back to U, yield the subset $U_\varphi \cap U^{ml}$. Finally, a temporal logic falsifier can be invoked over $U_\varphi \cap U^{ml}$ and a set of counterexamples is returned. □

This example illustrates how the compositional approach relies on tools, such as Breach [5], that compute validity domains and falsify STL specifications, as well as a ML analyzer. In the next section, we introduce our ML analyzer that identifies misclassifications of the ML component relevant to the overall CPSML input space.

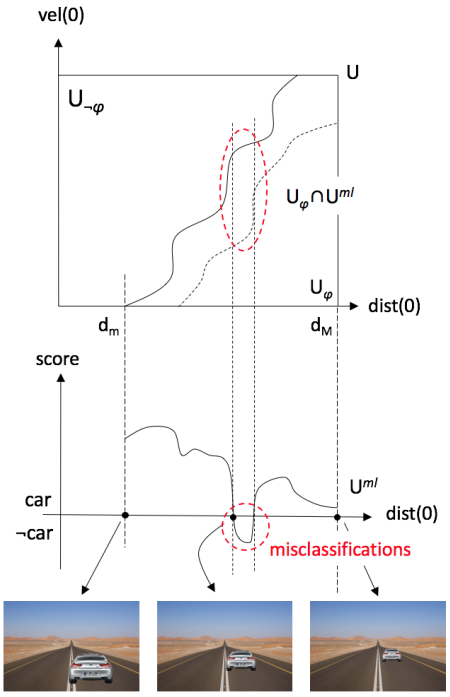


Fig. 2: Compositional falsification scheme on AEBS model.

4 Machine Learning Analyzer

In this section, we define an ML analyzer that adapts the input of a model to its classifiers feature spaces and identifies subsets of feature vectors for which wrong labels are predicted. The analysis involves the construction of an approximation function used to study the original classifiers. In particular, given a classifier $f : X \to Y$, the ML analyzer determines a simpler function $\tilde{f} : A \to Y$ that approximates f on the abstract domain A. The abstract domain of the function $\tilde{f}$ is analyzed and clusters of misclassifying abstract elements are identified. The concretizations of such elements are subsets of features that are misclassified by the original classifier f.

4.1 Feature Space Abstraction

Let $\tilde{X} \subseteq X$ be a subset of the feature space of $f : X \to Y$. Let $\leq$ be a total order on a set A called the abstract set. An abstraction function is an injective function $\alpha : \tilde{X} \to A$ that maps every feature vector $\mathbf{s} \in \tilde{X}$ to an abstract element $\alpha(\mathbf{s}) \in A$. Conversely, the concretization function $\gamma : A \to \tilde{X}$ maps every abstraction $\mathbf{a} \in A$ to a feature $\gamma(\mathbf{a}) \in \tilde{X}$.

The abstraction and concretization functions play a fundamental role in our falsification framework. First, they allow us to map the input space of the CPS model to the feature space of its classifiers. Second, the abstract space can be used to analyze the classifiers on a compact domain as opposite to intractable

feature spaces. These concepts are clarified in the following example, where a feature space of pictures is abstracted into a three-dimensional unit hyper-box.

Example 3. Let X be the set of RGB pictures of size 1000×600, i.e., $X = \{0, \ldots, 255\}^{1000 \times 600 \times 3}$. Suppose we are interested in analyzing an image classifier in the automotive context, i.e., on pictures of road scenarios rather than on the whole X. Suppose that we focus on the constrained feature space $\tilde{X} \subseteq X$ composed by the set of pictures of cars overlapped in different positions over a desert road background. We also consider the brightness level of the picture. The x and z positions of the car and the brightness level of the picture can be seen as the dimensions of an abstract set A. In this setting, we can define the abstraction and concretization functions α and γ that relate the abstract set $A = [0, 1]^3$ and $\tilde{X}$. For instance, the picture $\gamma(0, 0, 0)$ sees the car on the left, close to the observer, and low brightness; the picture $\gamma(1, 0, 0)$ places the car shifted to the right; on the other extreme, $\gamma(1, 1, 1)$ has the car on the right, far away from the observer, and with a high brightness level. Figure 3 depicts some car pictures of $\tilde{S}$ disposed accordingly to their position in the abstract domain A (the surrounding box).

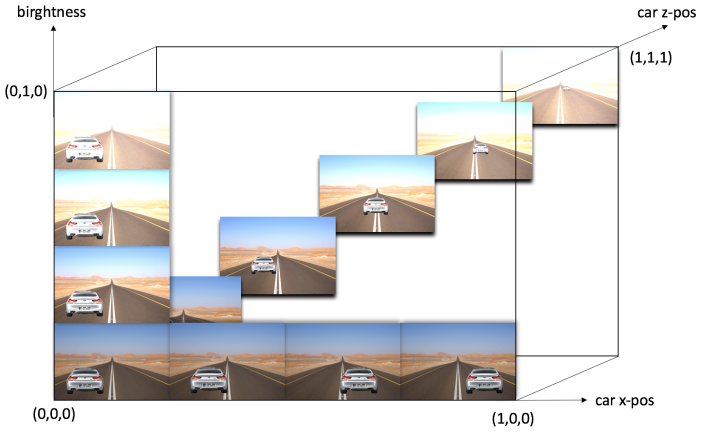


Fig. 3: Example of feature space abstraction A (the surrounding box) and some concretized element of the feature space $\tilde{X}$ (road pictures).

□

4.2 Approximation of Learning Components

We now describe how the feature space abstraction can be used to construct an approximation that helps the identification of misclassified feature vectors.

Given a classifier $f : X \to Y$ and a constrained feature space $\tilde{X} \subseteq X$, we want to determine an approximated classifier $\tilde{f} : A \to Y$, such that $err_{\tilde{f}}(T) \leq \epsilon$,

for some $0 \leq \epsilon \leq 1$ and test set $T = \{(\mathbf{a}^{(1)}, y^{(1)}), \ldots, (\mathbf{a}^{(l)}, y^{(l)})\}$, with $y^{(i)} = f(\gamma(\mathbf{a}^{(i)}))$, for $i = 1, \ldots, l$.

Intuitively, the proposed approximation scheme samples elements from the abstract set, computes the labels of the concretized elements using the analyzed learning algorithm, and finally, interpolates the abstract elements and the corresponding labels in order to obtain an approximation function. The obtained approximation can be used to reason on the considered feature space and identify clusters of potentially misclassified feature vectors.

Algorithm 2 Approximation construction of classifier $f : X \to Y$

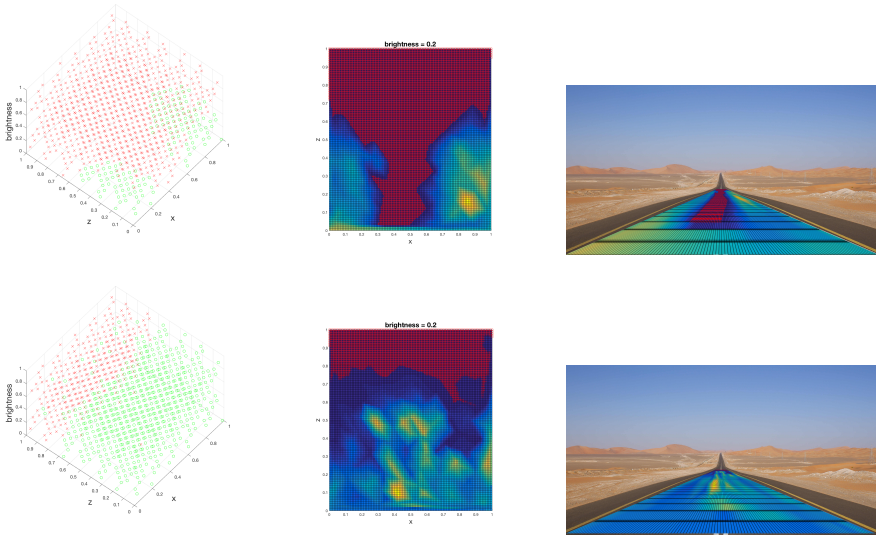
1: **function** APPROXIMATION(A, γ, ϵ) ▷ A abstract set ($\gamma : A \to \tilde{X}$), $0 \leq \epsilon \leq 1$
2: $T_I \leftarrow \emptyset$
3: **repeat**
4: $T_I \leftarrow T_I \cup$ SAMPLE(A, f)
5: $\tilde{f} \leftarrow$ INTERPOLATE(T_I)
6: $T_E \leftarrow$ SAMPLE(A, f)
7: **until** $err_{\tilde{f}}(T_E) \leq \epsilon$
8: **return** $\tilde{f}$
9: **end function**

The APPROXIMATION algorithm (Algorithm 2) formalizes the proposed approximation construction technique. It receives in input an abstract domain A for the concretization function $\gamma : A \to \tilde{X}$, with $\tilde{X} \subseteq X$, the error threshold $0 \leq \epsilon \leq 1$, and returns a function $\tilde{f} : A \to Y$ that approximates f on the constrained feature space $\tilde{X}$. The algorithm consists in a loop that iteratively improves the approximation $\tilde{f}$. At every iteration, the algorithm populates the interpolation test set T_I by sampling abstract features from A and computing the concretized labels accordingly to f (Line 4), i.e., SAMPLE(A, f)= $\{(\mathbf{a}, y) \mid \mathbf{a} \in \tilde{A}, y = f(\gamma(\mathbf{a}))\}$, where $\tilde{A} \subseteq A$ is a finite subset of samples determined with some sampling method. Next, the algorithm interpolates the points of T_I (Line 5). The result is a function $\tilde{f} : A \to Y$ that simplifies the original classifier f on the concretized constrained feature space $\tilde{X}$. The approximation is evaluated on the test set T_E. Note that at each iteration, T_E changes while T_I incrementally grows. The algorithm iterates until the error rate $err_{\tilde{f}}(T_E)$ is smaller than the desired threshold ϵ (Line 7).

The technique with which the samples in T_E and T_I are selected strongly influences the accuracy of the approximation. In order to have a good coverage of the abstract set A, we propose the usage of low-discrepancy sampling methods that, differently from uniform random sampling, cover sets quickly and evenly. In this work, we use the Halton and lattice sequences, that are two common and easy to implement sampling methods. For details see, e.g., [18].

Example 4. We now analyze two image classifiers: the Caffe [11] version of AlexNet [12] and the Inception-v3 model of Tensorflow [14], both trained on the ImageNet database [1]. We sample 1000 points from the abstract domain

defined in Example 3 using the lattice sampling techniques. These points encode the x and z displacements of a car in a picture and its brightness level (see Figure 3). Figure 4 (a) depicts the sampled points with their concretized labels. The green circles indicate correct classifications, i.e., the classifier identified a car, the red circles denote misclassifications, i.e., no car detected. The linear interpolation of the obtained points leads to an approximation function. The error rates $err_{\tilde{f}}(T_E)$ of the obtained approximations (i.e., the discrepancies between the predictions of the original image classifiers and their approximations) computed on 300 randomly picked test cases are 0.0867 and 0.1733 for Caffe and Tensorflow, respectively. Figure 4 (b) shows the projections of the approximation functions for the brightness value 0.2. The more red a region, the larger the sets of pictures for which the neural networks do not detect a car. For illustrative purposes, we superimpose the projections of Figure 4 (b) over the background used for the picture generation. These illustrations show the regions of the concrete feature vectors in which a vehicle is misclassified. □



(a) Sampling. (b) Interpolation projection. (c) Feature space analysis.

Fig. 4: ML analysis of Caffe (top) and Tensorflow (bottom) on a road scenario.

The analysis of Example 4 on Caffe and Tensorflow provides useful insights. First, we observe that Tensorflow outperforms Caffe on the considered road pictures since it correctly classifies more pictures that Caffe. Second, we notice that Caffe tends to correctly classify pictures in which the x abstract component is either close to 0 or 1, i.e., pictures in which the car is not in the middle of the street, but on one of the two lanes. This suggests that the model might not have been trained enough with pictures of cars in the center of the road. Third, using the lattice method on Tensorflow, we were able to identify a corner case misclassification in a cluster of correct predictions (note the isolated red circle

with coordinates $(0.1933, 0.0244, 0.4589)$). All this information provides insights on the classifiers that can be useful in the hunt for counterexamples.

5 Experimental Results

5.1 Implementation Details

The presented falsification framework has been implemented in a Matlab toolbox publicly available at `https://github.com/tommasodreossi/FalsifCPSML`. The tool deals with Simulink models of CPSML and STL specifications. It consists of a temporal logic falsifier and an ML analyzer that interact to falsify the given STL specification against the decomposed Simulink model. As an STL falsifier, we chose the existing tool Breach [5], while the ML analyzer has been implemented from scratch. The ML analyzer implementation includes the feature space abstractor and the ML approximation algorithm (see Section 4). The feature space abstractor implements a picture generator that concretizes the abstracted feature vectors. The approximation algorithm, that computes an approximation of the analyzed ML component, gives to the user the possibility of selecting the sampling sequence method, interpolation technique, and setting the desired error rate. Our tool is interfaced with the deep learning frameworks Caffe [11] and Tensorflow [14]. Our tool has been tested on a desktop computer Dell XPS 8900, Intel (R) Core(TM) i7-6700 CPU 3.40GHz, DIMM RAM 16 GB 2132 MHz, GPU NVIDIA GeForce GTX TITAN X, with Ubuntu 14.04.5 LTS and Matlab R2016b.

5.2 Case Studies

For the experimental evaluations, we consider a closed-loop Simulink model of a semi-autonomous vehicle with an Advanced Emergency Braking System (AEBS) [21] connected to an image classifier. The model mainly consists of a four-speed automatic transmission controller linked to an AEBS that automatically prevents collisions with preceding obstacles and alleviate the harshness of a crash when a collision is likely to happen (see Figure 5). The AEBS determines

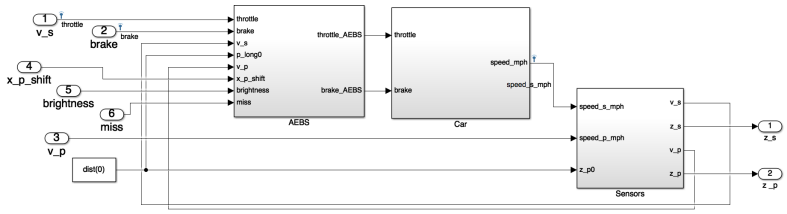


Fig. 5: Simulink model of a semi-autonomous vehicle with AEBS.

a braking mode depending on the speed of the vehicle v_s, the possible presence

of a preceding obstacle, its velocity v_p, and the longitudinal distance $dist$ between the two. The distance $dist$ is provided by radars having 30m of range. For obstacles farther than 30m, the camera, connected to an image classifier, alerts the AEBS that, in the case of detected obstacle, goes into warning mode.

Depending on $v_s, v_p, dist$, and the presence of obstacles detected by the image classifier, the AEBS computes the time to collision and longitudinal safety indices, whose values determine a controlled mode among safe, warning, braking, and collision mitigation. In safe mode, the car does not need to brake. In warning mode, the driver should brake to avoid a collision. If this does not happen, the system goes into braking mode, where the automatic brake slows down the vehicle. Finally, in collision mitigation mode, the system, determining that a crash is unavoidable, triggers a full braking action aimed to minimize the damage.

To establish the correctness of the system and in particular of its AEBS controller, we formalize the STL specification $G(\neg(dist(t)) \leq 0)$, that requires $dist(t)$ to always be positive, i.e., no collision happens. The input space is $v_s(0) \in [0, 40]$ (mph), $dist(0) \in [0, 60]$ (m), and the set of all RGB pictures of size 1000×600. The preceding vehicle is not moving, i.e., $v_p(t) = 0$ (mph).

At first, we compute the validity domain of φ assuming that the radars are able to provide exact measurements for any distance $dist(t)$ and the image classifier correctly detects the presence of a preceding vehicle. The computed validity domain is depicted in Figure 6: green for U_φ and red for $U_{\neg\varphi}$. Next, we identify candidate counterexamples that belong to the satisfactory set (i.e., the inputs that satisfy the specification) but might be influenced by a misclassification of the image classifier. Since the AEBS relies on the classifier only for distances larger than 30m, we can focus on the subset of the input space with $dist(0) \geq 30$. Specifically, we identify potential counterexamples by analyzing a pessimistic version of the model where the ML component always misclassifies the input pictures (see Figure 6, area with dashed boundary). From this sub-input space, we can identify candidate counterexamples, such as, for instance, $(25, 40)$ (i.e., $v_s(0) = 25$ and $dist(0) = 40$).

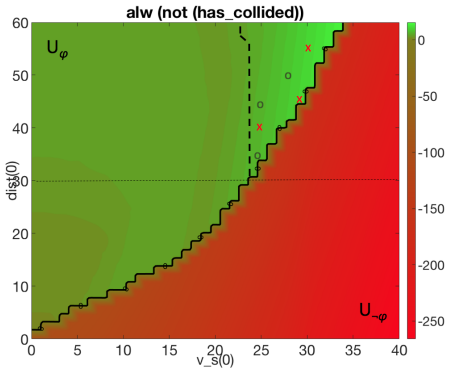


Fig. 6: Validity domain for $G(\neg(dist(t)) \leq 0)$. Proved (red crosses) and disproved (green circles) candidate counterexamples. Dotted (horizontal) line: image classifier activation threshold. Dashed (vertical) line: validity boundary of φ for worst-case misclassifications.

Next, let us consider the Caffe image classifier and the ML analyzer presented in Section 4 that generates pictures from the abstract feature space $A = [0, 1]^3$, where the dimensions of A determine the x and z displacements of a car and the brightness of a generated picture, respectively. The goal now is to determine an abstract feature $\mathbf{a}_c \in A$ related to the candidate counterexample $(25, 40)$, that

generates a picture that is misclassified by the ML component and might lead to a violation of the specification φ. The $dist(0)$ component of $\mathbf{u}_c = (25, 40)$ determines a precise z displacement $\mathbf{a}_2 = 0.2$ in the abstract picture. Now, we need to determine the values of the abstract x displacement and brightness. Looking at the interpolation projection of Figure 4 (b), we notice that the approximation function misclassifies pictures with abstract component $\mathbf{a}_1 \in [0.4, 0.5]$ and $\mathbf{a}_3 = 0.2$. Thus, it is reasonable to try to falsify the original model on the input element $v_s(0) = 25, dist(0) = 40$, and concretized picture $\gamma(0.5, 0.2, 0.2)$. For this targeted input, the temporal logic falsifier computed a robustness value for φ of -24.60, meaning that a falsifying counterexample has been found. Other counterexamples found with the same technique are, e.g., $(27, 45)$ or $(31, 56)$ that, associated with the correspondent concretized pictures with $\mathbf{a}_1 = 0.5$ and $\mathbf{a}_3 = 0.2$, lead to the robustness values -23.86 and -24.38, respectively (see Figure 6, red crosses). Conversely, we also disproved some candidate counterexamples, such as $(28, 50)$, $(24, 35)$, or $(25, 45)$, whose robustness values are $9.93, 7.40$, and 7.67 (see Figure 6, green circles).

For experimental purposes, we try to falsify a counterexample in which we change the x position of the abstract feature so that the approximation function correctly classifies the picture. For instance, by altering the counterexample $(27, 45)$ with $\gamma(0.5, 0.225, 0.2)$ to $(27, 45)$ with $\gamma(1.0, 0.225, 0.2)$, we obtain a robusteness value of 9.09, that means that the AEBS is able to avoid the car for the same combination of velocity and distance of the counterexample, but different x position of the preceding vehicle. Another example, is the robustness value -24.38 of the falsifying input $(31, 56)$ with $\gamma(0.5, 0.28, 0.2)$, that altered to $\gamma(0.0, 0.28, 0.2)$, changes to 12.41.

Finally, we test Tensorflow on the corner case misclassification identified in Section 4.2 (i.e, the picture $\gamma(0.1933, 0.0244, 0.4589)$). The distance $dist(0) = 4.88$ related to this abstract feature is below the activation threshold of the image classifier. Thus, the falsification points are exactly the same as those of the computed validity domain (i.e., $dist(0) = 4.88$ and $v_s(0) \in [4, 40]$). This study shows how a misclassification of the ML component might not affect the correctness of the CPSML model.

6 Conclusion

We presented a compositional falsification framework for STL specifications against CPSML models based on the separate analysis of a CPS system and its ML components. We introduced an ML analyzer able to abstract feature spaces, approximate ML classifiers, and provide sets of misclassified feature vectors that can be used to drive the falsification process. We implemented our framework and showed its effectiveness for an autonomous driving controller using perception based on deep neural networks.

This work lays the basis for future advancements. We intend to improve our ML analyzer exploring the automatic generation of feature space abstractions from given training sets. Another direction is to integrate other techniques for generating misclassifications of ML components (e.g. [16, 10]) into our approach. One could also apply our ML analyzer outside the falsification context, such

as for controller synthesis. Finally, our compositional methodology could be extended to other, non-cyber-physical, systems that contain ML components.

References

1. Imagenet. `http://image-net.org/`.
2. Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 254–257, 2011.
3. A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1):245–271, 1997.
4. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
5. A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification, CAV*, pages 167–170, 2010.
6. T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods, NFM*, pages 127–142, 2015.
7. P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2E2: a verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 68–82. Springer, 2015.
8. A. Fawzi, O. Fawzi, and P. Frossard. Analysis of classifiers' robustness to adversarial perturbations. *arXiv preprint arXiv:1502.02590*, 2015.
9. G. Hinton et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
10. X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. *CoRR*, abs/1610.06940, 2016.
11. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia Conference, ACMMM*, pages 675–678, 2014.
12. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
13. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.
14. Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
15. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.
16. S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.
17. A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Computer Vision and Pattern Recognition, CVPR*, pages 427–436. IEEE, 2015.
18. H. Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of number theory*, 30(1):51–70, 1988.
19. S. A. Seshia, D. Sadigh, and S. S. Sastry. Towards verified artificial intelligence. *CoRR*, abs/1606.08514, 2016.

20. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.
21. L. Taeyoung, Y. Kyongsu, K. Jangseop, and L. Jaewan. Development and evaluations of advanced emergency braking system algorithm for the commercial vehicle. In *Enhanced Safety of Vehicles Conference, ESV*, pages 11–0290, 2011.
22. V. Vapnik. Principles of risk minimization for learning theory. In *NIPS*, pages 831–838, 1991.