



On the Detection of Doped Software by Falsification^{*}

Sebastian Biewer¹ (✉)  and Holger Hermanns^{1,2} 

¹ Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
`biewer@depend.uni-saarland.de`

² Institute of Intelligent Software, Guangzhou, China

Abstract. Software doping is a phenomenon that refers to the presence of hidden software functionality, whose existence is only in the interest of the manufacturer. The most prominent example is the diesel emissions scandal. There is a need for methods that identify software doping, and such methods are bound to be applied to the final product with no or rare knowledge about its internals. Black-box analysis techniques have recently been developed for this purpose, harvesting the formal foundations of software doping. This paper integrates them with established falsification techniques for the purpose of real-world applicability. With a focus on the diesel scandal and emissions tests on chassis dynamometers we make the testing procedures significantly more effective in terms of time and cost. The theoretical results are implemented in a prototypical doping tester.

1 Introduction

Embedded software is the innovation driver of our times. Software-defined systems are permeating our communication, perception, and storage technology as well as our personal interactions with technical systems at an unprecedented pace. “*Software-defined everything*” is among the hottest buzzwords in IT technology today [2, 18].

There is a tremendous problem hiding behind this apparently unstoppable trend: The owners of the physical “hull” of *everything* will not be the ones owning the software defining *everything*, nor will they have the right to look at what and how *everything* is defined. This is because commercial software typically is protected by intellectual property rights of the software manufacturer. This prohibits any attempt to disassemble the software or to reconstruct its inner working, albeit it is the very software that is forecasted to be defining *everything*. The use of machine-learned software components amplifies the problem considerably. Since commercial interests of the software manufacturers seldomly are aligned

^{*} This work is partly supported by DFG grant 389792660 as part of [TRR 248 – CPEC](#), the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101008233, and by the Key-Area Research and Development Program Grant 2018B010107004 of Guangdong Province.

with the interest of end users, the promise of *software-defined everything* might well become a dystopia from the perspective of individual digital sovereignty.

A massive example of software-defined collective damage is the diesel emissions scandal. Over a period of more than 10 years, millions of diesel-powered cars have been equipped with illegal software that altogether polluted the environment for the sake of commercial advantages of the car manufacturers. At its core, this was made possible by the fact that only a single, precisely defined test setup was put in place for checking conformance with exhaust emissions regulations. This made it a trivial software engineering task to identify the test particularities and to turn off emission cleaning outside these particular conditions. This is an archetypical instance of software doping.

Against this background, there is an urgent need to establish stronger and enforceable requirements on the systems we are interacting with, and this is indeed echoed in legislative frameworks [24]. However, the roll-out of such requirements in everyday practice needs a firm understanding of the technological basis for enforcing such requirements, respectively for identifying violations thereof.

This paper is part of ongoing research addressing this challenge. It harvests the outcomes of three recent scientific achievements: (i) formal definitions of software doping based on contracts enforcing well-defined software behaviour in the vicinity of standardised behaviour [12], (ii) a solid foundation for doping tests to be carried out in practice [6], and (iii) probabilistic falsification techniques developed to guide the search for property violations in cyber-physical system engineering [1, 20]. By combining the above ingredients, this paper addresses the question how to perform cost-effective doping tests that are indeed likely to succeed in uncovering actual cases of doped software. It approaches this question both from a foundational and from a practical perspective. On the foundational side, we introduce a temporal hyperlogic to reason about signals which we use to characterise the falsifiable fragment of a software doping contract. Great care is taken for this to work on the actual time-discrete traces that are recorded from the real system which itself is running in continuous time. On the practical side, we discuss a novel approach to probabilistic falsification that overcomes the problem that in many practical cases the possibility to carry out masses of highly-controlled experiments with a physical system is severely limited by cost or time budgets. To account for this, we add a passive recording component to the concept of falsification which observes the system in-the-wild to propose only few candidate traces to be inspected under lab conditions. All this is instantiated in the context of automotive emissions, where lab conditions correspond to expensive test runs on a chassis dynamometer, while observing the system in-the-wild is nothing else than collecting statistics while driving on normal roads.

The paper makes the following distinguished contributions: (i) a linear temporal logic for hyperproperties over continuous signals that enables quantitative reasoning across traces, (ii) a logical reformulation of the falsifiable fragment of a software doping contract, (iii) a probabilistic falsification technique that uses passive recording for cost-effective doping testing, and (iv) an exemplary instantiation of these concepts in the context of automotive emissions.

Related Work. Software doping theory provides a formal basis for enlarging the requirements on vehicle exhaust emissions beyond too narrow lab test conditions. That conceptual limitation has by now been addressed by the official authorities responsible for car type approval [24, 25]: The old NEDC-based test procedure is replaced by the newer *Worldwide Harmonised Light Vehicles Test Procedure* (WLTP), which is deemed to be more realistic. WLTP replaces the NEDC test by a new WLTC test, but WLTC still is just a single test scenario. In addition, WLTP embraces so called *Real Driving Emissions* (RDE) tests to be conducted on public roads. A recently launched mobile phone app [8], *LolaDrives*, harvests runtime monitoring technology for making low-cost RDE tests accessible to everyone.

Learning or approximating the behaviour of a system under test has been studied intensively. Meinke and Sindhu [19] were among the first to present a testing approach incrementally learning a Kripke structure representing a reactive system. Volpato and Tretmans [27] propose a learning approach which gradually refines an under- and over-approximation of an input-output transition system representing the system under test. The correctness of this approach needs several assumptions, e.g., an oracle indicating when, for some trace, all outputs, which extend the trace to a valid system trace, have been observed.

2 Background

This section introduces the necessary background regarding temporal logics for hyperproperties and for continuous signals, probabilistic falsification basics, and reviews the formal definitions of software doping.

2.1 Temporal Logics

Linear Temporal Logic (LTL) [22] is a popular formalism to reason about properties of traces. A trace is an infinite word where each literal is a subset of AP , the set of atomic propositions. Programs are interpreted as sets $S_{\text{LTL}} \subseteq (2^{\text{AP}})^\omega$ of such traces. LTL provides expressive means to characterise sets of traces, often called *trace properties*.

Temporal Logics for Hyperproperties. For some set of traces T , a trace property defines a subset of T , whereas a *hyperproperty* defines a *set of* subsets of T . In this way it specifies which traces are valid in combination with one another. Many temporal logics have been extended to corresponding hyperlogics supporting the specification of hyperproperties. HyperLTL [11] is such a temporal logic for the specification of hyperproperties of reactive systems. It extends LTL with trace quantifiers and trace variables that make it possible to refer to multiple traces within a logical formula. A *HyperLTL formula* is defined by the following grammar where π is drawn from a set \mathcal{V} of *trace variables*:

$$\begin{aligned} \psi &::= \exists \pi. \psi \mid \forall \pi. \psi \mid \phi \\ \phi &::= a_\pi \mid \neg \phi \mid \phi \wedge \phi \mid \text{X} \phi \mid \phi \cup \phi \end{aligned}$$

The quantifiers \exists and \forall quantify existentially and universally, respectively, over the set of traces. For example, the formula $\forall\pi. \exists\pi'. \phi$ means that for every trace π there exists another trace π' such that ϕ holds over the pair of traces. To account for distinct valuations of atomic propositions across distinct traces, the atomic propositions are indexed with trace variables: for some atomic proposition $a \in \mathbf{AP}$ and some trace variable $\pi \in \mathcal{V}$, a_π states that a holds in the initial position of trace π . The temporal operators and Boolean connectives are interpreted as usual for LTL. In particular, $\mathbf{X}\phi$ means that ϕ holds in the next state of every trace under consideration. Likewise, $\phi \mathbf{U} \phi'$ means that ϕ' eventually holds in every trace under consideration at the same point in time, provided ϕ holds in every previous instant in all such traces. Further operators are derivable: $\mathbf{F}\phi \equiv \mathbf{true} \mathbf{U} \phi$ enforces ϕ to eventually hold in the future, $\mathbf{G}\phi \equiv \neg \mathbf{F} \neg \phi$ enforces ϕ to always hold, and the weak-until operator $\phi \mathbf{W} \phi' \equiv \phi \mathbf{U} \phi' \vee \mathbf{G}\phi$ allows ϕ to always hold as an alternative to the obligation for ϕ' to eventually hold. We refer to [11] for the formal semantics.

Temporal Logics over Continuous Domains. LTL enables reasoning over traces $\sigma \in (2^{\mathbf{AP}})^\omega$ which are of discrete nature with respect to the time domain they represent. With each literal in the trace representing a time step, σ can equivalently be viewed as a function $\mathbb{N} \rightarrow 2^{\mathbf{AP}}$. One extension of LTL is *Signal Temporal Logic* (STL) [13, 17], which instead is used for reasoning over real-valued signals that may change in value along an underlying time domain. A signal is a function $s : \mathcal{T} \rightarrow \mathbb{R}$ where \mathcal{T} is the time domain. The time domain \mathcal{T} can be either \mathbb{N} (*discrete-time* signals), or $\mathbb{R}_{\geq 0}$ (*continuous-time* signals). This can be lifted to multi-dimensional signals $w(t) = (s_1(t), \dots, s_n(t))$, mapping each time point to some element of \mathbb{R}^n . We refer to such a $w : \mathcal{T} \rightarrow \mathbb{R}^n$ as a (discrete-time or continuous-time) *trace* of *width* n in the sequel.

STL formulas can express properties of systems modelled as sets $\mathbf{S}_{\text{STL}} \subseteq (\mathcal{T} \rightarrow \mathbb{R}^n)$ of traces of some fixed width n , basically by making the atomic properties refer to booleanizations of the signal values. The syntax of the variant of STL that we use in this paper is as follows, where $f \in \mathbb{R}^n \rightarrow \mathbb{R}$:

$$\phi ::= \top \mid f > 0 \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathbf{U} \phi.$$

STL replaces atomic propositions by *threshold predicates* of the form $f > 0$, which hold if and only if function f applied to the signal values at the current time returns a positive value. The Boolean operators and the Until operator \mathbf{U} are very similar to those of HyperLTL. The Next operator \mathbf{X} is not part of STL, because “next” is without precise meaning in continuous time. The definitions of the derived operators \mathbf{F} , \mathbf{G} and \mathbf{W} are the

$$\begin{array}{ll} w, t \models \top & \\ w, t \models f > 0 & \text{iff } f(s_1(t), \dots, s_n(t)) > 0 \\ w, t \models \neg\phi & \text{iff } w, t \not\models \phi \\ w, t \models \phi \wedge \psi & \text{iff } w, t \models \phi \text{ and } w, t \models \psi \\ w, t \models \phi \mathbf{U} \psi & \text{iff exists } t' \geq t \text{ s.t. } w, t' \models \psi \text{ and} \\ & \text{for all } t'' \in [t, t'), w, t'' \models \phi \end{array}$$

Fig. 1: Boolean semantics of STL formulas

same as for HyperLTL. Formally, the *Boolean semantics* of an STL formula ϕ at time point $t \in \mathcal{T}$ for a trace $w = (s_1, \dots, s_n)$ is defined inductively in Fig. 1.

Quantitative Interpretation. STL has been extended by a *quantitative semantics* [1, 13, 14] as presented in Fig. 2. This semantics is designed in such a way that whenever $\rho(\phi, w, t) \neq 0$, its sign indicates whether $w, t \models \phi$ holds in the Boolean semantics. For any STL formula ϕ , trace w and time t , if $\rho(\phi, w, t) > 0$, then $w, t \models \phi$ holds, and if $\rho(\phi, w, t) < 0$, then $w, t \models \phi$ does not hold. For the scope of this paper,

we work with the untimed Until operator, instead of allowing $U_{[a,b]}$ for arbitrary bounds $a, b \in \mathbb{R}$. With only the untimed Until operator, the continuous and discrete semantics [14] coincide.

$$\begin{aligned} \rho(\top, w, t) &= \infty \\ \rho(f > 0, w, t) &= f(s_1(t), \dots, s_n(t)) \\ \rho(\neg\phi, w, t) &= -\rho(\phi, w, t) \\ \rho(\phi \wedge \psi, w, t) &= \min(\rho(\phi, w, t), \rho(\psi, w, t)) \\ \rho(\phi \cup \psi, w, t) &= \sup_{t' \geq t} \min\{\rho(\psi, w, t'), \inf_{t'' \in [t, t']} \rho(\phi, w, t'')\} \end{aligned}$$

Fig. 2: Quantitative semantics of STL formulas

Robustness and Falsification. The value of the quantitative semantics can serve as a *robustness estimate* and as such be used to search for a violation of the property at hand, i.e., to falsify it. The robustness of STL formula ϕ is its quantitative value at time 0, that is, $\mathcal{R}_\phi(w) := \rho(\phi, w, 0)$.

So, falsifying a formula ϕ for a system S_{STL} boils down to a search problem with the goal condition $\mathcal{R}_\phi(w) < 0$. Successful falsification algorithms solve this problem by understanding it as the optimisation problem $\text{minimise}_{w \in S_{\text{STL}}} \mathcal{R}_\phi(w)$. Algorithm 1 [1, 20] sketches an algorithm for Monte-Carlo Markov

Algorithm 1 Monte-Carlo falsification

Input: w : Initial trace, \mathcal{R} : Robustness function, PS: Proposal Scheme

Output: $w \in S_{\text{STL}}$

```

1: while  $\mathcal{R}(w) > 0$  do
2:    $w' \leftarrow \text{PS}(w)$ 
3:    $\alpha \leftarrow \exp(-\beta(\mathcal{R}(w') - \mathcal{R}(w)))$ 
4:    $r \leftarrow \text{UniformRandomReal}(0, 1)$ 
5:   if  $r \leq \alpha$  then
6:      $w \leftarrow w'$ 
7:   end if
8: end while
```

Chain falsification, which is based on acceptance-rejection sampling [10]. Our version of the algorithm works on system traces instead of an input space. An input to the algorithm is an initial trace w and a computable robustness function \mathcal{R} . Robustness computation for finite timed traces of simulations of a system has been discussed in the literature [13, 14]; we omit this discussion here. The third input PS is a proposal scheme that proposes a new trace to the algorithm based on the previous one (line 2). The parameter β (used in line 3) can be adjusted during the search and is a means to avoid being trapped in local minima, preventing to find a global minimum. Any two traces w and $w' \in S_{\text{STL}}$ with robustness values $\mathcal{R}(w)$ and $\mathcal{R}(w')$ are sampled with probability proportional to

$\frac{e^{-\beta\mathcal{R}(w)}}{e^{-\beta\mathcal{R}(w')}} \xi_{\text{STL}}$ (lines 3-6). The algorithm seeks to minimise \mathcal{R} over the system's traces ξ_{STL} , and terminates when it finds a trace with a negative robustness value, i.e., a trace that violates the STL property from which \mathcal{R} is derived.

2.2 Software Doping

Contracts and Robustness. Earlier work [12] has developed a formal basis for the purpose of characterising software doping, by providing precise definitions of when the system's behaviour is *clean*, i.e., does not contain hidden functionalities not in the interest of the user. If a program exhibits behaviour that is not clean, it is *doped*.

All cleanness definitions are based on the assumption that there is some well-defined and agreed standard input/output behaviour of the system. *Robust cleanness*, the cleanness definition that we work with in this paper, extends this behaviour to the vicinity around the inputs and outputs close to the standard behaviour. The definition of “vicinity” and of “standard behaviour” is assumed to be part of a *contract* between software manufacturer and user. The contract entails the standard behaviour, distance functions for input and output values, and distance thresholds to define the input and output vicinity, respectively. With this, a system behaviour is considered clean, if its output is (or stays) in the output vicinity of the standard, unless the input is (or moves) outside the standard's input vicinity.

Example 1. A concrete contract for diesel-powered cars will, for instance, enforce bounded deviations in exhaust emissions provided the driving profile stays in the bounded vicinity of the standardised tests (such as NEDC or WLTC). Recent experiments [6] have considered contracts based on NEDC with speed values as inputs and NO_x emissions as output values, together with distance functions computing the absolute difference of speed inputs and NO_x outputs, respectively, and value thresholds were 15 km/h for inputs and 80 mg/km for outputs.

A function $d : X \times X \rightarrow \mathbb{R}_{\geq 0}$ is a pseudometric function if it satisfies $d(x, x) = 0$, $d(x, y) = d(y, x)$ and $d(x, y) \leq d(x, z) + d(z, y)$ for all $x, y, z \in X$. We let $\sigma[k]$ denote the k -th literal of the infinite word σ .

Reactive Execution Model. We can view a (nondeterministic) reactive program as a function $S_R : \text{In}^\omega \rightarrow 2^{(\text{Out}^\omega)}$ perpetually mapping inputs In to sets of outputs Out [12]. A *contract* is a tuple $\mathcal{C} = \langle \text{StdIn}, d_{\text{In}}, d_{\text{Out}}, \kappa_i, \kappa_o \rangle$ where $\text{StdIn} \subseteq \text{In}^\omega$ is the input space of the system designated to define the standard behaviour, $d_{\text{In}} : (\text{In}^* \times \text{In}^*) \rightarrow \mathbb{R}_{\geq 0}$ and $d_{\text{Out}} : (\text{Out}^* \times \text{Out}^*) \rightarrow \mathbb{R}_{\geq 0}$ are pseudometric distance functions on finite words over inputs, respectively outputs, and $\kappa_i \in \mathbb{R}_{\geq 0}$ is a constant defining the maximum distance to the standard input allowed, and similarly $\kappa_o \in \mathbb{R}_{\geq 0}$ is the maximum distance between two outputs such that they are still considered sufficiently close. For the purpose of this paper, we assume the distance functions to be induced by pointwise pseudometric functions of the form $d_{\text{In}} : (\text{In} \times \text{In}) \rightarrow \mathbb{R}_{\geq 0}$ and $d_{\text{Out}} : (\text{Out} \times \text{Out}) \rightarrow \mathbb{R}_{\geq 0}$ in a past-forgetful manner.

Definition 1. A reactive program $S_R : \text{In}^\omega \rightarrow 2^{(\text{Out}^\omega)}$ is robustly clean w.r.t. to contract $C = \langle \text{StdIn}, d_{\text{In}}, d_{\text{Out}}, \kappa_i, \kappa_o \rangle$ if for all input sequences $i, i' \in \text{In}^\omega$ with $i \in \text{StdIn}$, it holds for arbitrary $k \geq 0$ that whenever $d_{\text{In}}(i[j], i'[j]) \leq \kappa_i$ for all $j \leq k$, then

1. for all $o \in S_R(i)$ there exists $o' \in S_R(i')$ such that $d_{\text{Out}}(o[k], o'[k]) \leq \kappa_o$, and
2. for all $o' \in S_R(i')$ there exists $o \in S_R(i)$ such that $d_{\text{Out}}(o[k], o'[k]) \leq \kappa_o$.

The definition enforces that whenever an input i' remains within κ_i vicinity around the standard input i , then the output sets generated by i and i' are at most κ_o away from each other.

HyperLTL Characterisation. D'Argenio et al. [12] prove that the following two HyperLTL formulas characterise robust cleanness in the sense of Definition 1.

$$\forall \pi_1. \forall \pi_2. \exists \pi'_2. \text{StdIn}_{\pi_1} \rightarrow \left(G(i_{\pi_2} = i_{\pi'_2}) \wedge \right. \quad (1)$$

$$\left. ((\hat{d}_{\text{Out}}(o_{\pi_1}, o_{\pi'_2}) \leq \kappa_o) \text{ W } (\hat{d}_{\text{In}}(i_{\pi_1}, i_{\pi'_2}) > \kappa_i)) \right)$$

$$\forall \pi_1. \forall \pi_2. \exists \pi'_1. \text{StdIn}_{\pi_1} \rightarrow \left(G(i_{\pi_1} = i_{\pi'_1}) \wedge \right. \quad (2)$$

$$\left. ((\hat{d}_{\text{Out}}(o_{\pi'_1}, o_{\pi_2}) \leq \kappa_o) \text{ W } (\hat{d}_{\text{In}}(i_{\pi'_1}, i_{\pi_2}) > \kappa_i)) \right)$$

The non-atomic propositions in the formulas above are syntactic sugar; the input and output values in system S_{LTL} give rise to a binary encoding into sets of atomic propositions.

Mixed-IO Model. The reactive execution model and the HyperLTL characterisation above have the strict requirement that for every input, the system produces exactly one output. Recent work [5, 6] instead considers mixed-IO models, where a program $S_{\text{IO}} \subseteq (\text{In} \cup \text{Out})^\omega$ is a subset of traces containing both inputs and outputs, but without any restriction on the order or frequency in which inputs and outputs appear in the trace. In particular, they are not required to strictly alternate (but they may, and in this way the reactive execution model can be considered a special case). A particularity of this model is the distinct output symbol δ for quiescence, i.e., the absence of an output. For example, finite behaviour can be expressed by adding infinitely many δ symbols to a finite trace. In this model, standard behaviour is captured by subset $\text{Std} \subseteq S_{\text{IO}}$ of traces of a system S_{IO} . To capture the notion of robust cleanness in the mixed-IO model, every trace is projected into an input, respectively output domain. The set of input symbols contains one additional element \neg_i , that indicates that in the respective steps an output was produced, but masking the concrete output. Similarly, the set of output symbols contains the additional element \neg_o to mask a concrete input symbol. *Projection on inputs* $\downarrow_i : (\text{In} \cup \text{Out})^\omega \rightarrow (\text{In} \cup \{\neg_i\})^\omega$ and *projection on outputs* $\downarrow_o : (\text{In} \cup \text{Out})^\omega \rightarrow (\text{Out} \cup \{\neg_o\})^\omega$ are defined for all traces $\sigma \in (\text{In} \cup \text{Out})^\omega$ and $k \in \mathbb{N}$ as follows: $\sigma \downarrow_i[k] := \text{if } \sigma[k] \in \text{In} \text{ then } \sigma[k] \text{ else } \neg_i$ and similarly $\sigma \downarrow_o[k] := \text{if } \sigma[k] \in \text{Out} \text{ then } \sigma[k] \text{ else } \neg_o$. The distance functions

\bar{d}_{In} and \bar{d}_{Out} apply on input and output symbols or their respective masks, i.e. they are pseudometrics in $(\text{In} \cup \{-i\}) \times (\text{In} \cup \{-i\}) \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ and, respectively, $(\text{Out} \cup \{-o\}) \times (\text{Out} \cup \{-o\}) \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$. As for the reactive model, we define a contract formally as a tuple $\mathcal{C} = \langle \text{Std}, \bar{d}_{\text{In}}, \bar{d}_{\text{Out}}, \kappa_i, \kappa_o \rangle$ (where StdIn is replaced by Std , d_{In} by \bar{d}_{In} , and d_{Out} by \bar{d}_{Out}). Its satisfaction is defined by the adapted robust cleanness definition below [6].

Definition 2. A system $S_{\text{IO}} \subseteq (\text{In} \cup \text{Out})^\omega$ is robustly clean w.r.t. contract $\mathcal{C} = \langle \text{Std}, \bar{d}_{\text{In}}, \bar{d}_{\text{Out}}, \kappa_i, \kappa_o \rangle$ if and only if $\text{Std} \subseteq S_{\text{IO}}$ and for all $\sigma \in \text{Std}$, $\sigma' \in S_{\text{IO}}$ and $k \geq 0$ it holds that whenever $\bar{d}_{\text{In}}(\sigma[j] \downarrow_i, \sigma'[j] \downarrow_i) \leq \kappa_i$ for all $j \leq k$ then

1. there exists $\sigma'' \in S_{\text{IO}}$ such that $\sigma' \downarrow_i = \sigma'' \downarrow_i$ and $\bar{d}_{\text{Out}}(\sigma[k] \downarrow_o, \sigma''[k] \downarrow_o) \leq \kappa_o$,
2. there exists $\sigma'' \in \text{Std}$ such that $\sigma \downarrow_i = \sigma'' \downarrow_i$ and $\bar{d}_{\text{Out}}(\sigma'[k] \downarrow_o, \sigma''[k] \downarrow_o) \leq \kappa_o$.

Def. 2 contains two requirements, numbered as 1. and 2. In the following, we will sometimes explicitly address either of these conditions by referring to it as the *first*, respectively *second condition of robust cleanness*.

3 Logical characterisation for mixed IO

This section discusses how to reformulate robust cleanness to make it amenable to probabilistic falsification. For this, we translate eq. (2) into a HyperSTL formula, subsequently remove its quantifiers by means of a highly efficient parallel composition on the level of traces and, finally, carefully adapt this quantifier-free representation to the mixed-IO model.

Hyperlogics over Continuous Domains. Previous work [21] extends STL to HyperSTL echoing the extension of LTL to HyperLTL. A major challenge of the robustness computation for HyperSTL formulas is the adequate handling of the continuous time domain when comparing two execution traces of a system. For systems that can be simulated, this can be avoided [21] by composing one or more copies of the simulation model in parallel to itself [11]. Snapshots of the composed system are effectively snapshots of the individual copies of the model at exactly the same time point. This approach is not available when interacting with (black-box) real-world cyber-physical systems (CPS). In such scenarios, a suitable logics is HyperSTL* [7], an extension of STL* [9], which enables the comparison of different time points in different traces by means of a freeze operator. We use a variant of this idea, but with a HyperSTL syntax similar to [21].

$$\begin{aligned} \psi &::= \exists \pi. \psi \mid \forall \pi. \psi \mid \phi \\ \phi &::= \top \mid f > 0 \mid \neg \phi \mid \phi \wedge \phi \mid \phi \mathbf{U} \phi. \end{aligned}$$

The meaning of the universal and existential quantifier is as for HyperLTL. A crucial difference to the other logics presented above is the proposition $f > 0$. In contrast to HyperLTL and to the existing definition of HyperSTL, we consider it insufficient to allow propositions to refer to only a single trace. In HyperLTL that does not cause harm, because atomic propositions of individual traces can

be compared by means of the Boolean connectives. To formulate thresholds for real values, however, we feel the need to allow real values from multiple traces to be combined in the function f , and thus to appear as arguments of f . Hence, in our semantics of HyperSTL, $f > 0$ holds if and only if the result of f , applied to all traces quantified over, is greater than 0. For this to work formally, the arity of function f is the product of the trace width n and the number m of traces quantified over at the occurrence of $f > 0$ in the formula, so $f : (\mathbb{R}^n)^m \rightarrow \mathbb{R}$.

A trace assignment [11] $\Pi : \mathcal{V} \rightarrow S_{\text{STL}}$ is a partial function assigning traces of S_{STL} to variables. Let $\Pi[\pi := w]$ denote the same function as Π , except that π is mapped to trace w . The quantitative semantics of a HyperSTL formula ψ , at time point $t \in \mathcal{T}$, for a system $S \subseteq (\mathcal{T} \rightarrow \mathbb{R}^n)$ and a trace assignment Π is defined inductively:

$$\begin{aligned}
\rho(\exists \pi. \psi, S, \Pi, t) &= \max_{w \in S} \rho(\psi, S, \Pi[\pi := w], t) \\
\rho(\forall \pi. \psi, S, \Pi, t) &= \min_{w \in S} \rho(\psi, S, \Pi[\pi := w], t) \\
\rho(\top, S, \Pi, t) &= \infty \\
\rho(f > 0, S, \Pi, t) &= f(\Pi(\pi_1)(t), \dots, \Pi(\pi_m)(t)) \\
&\quad \text{for } \text{dom}(\Pi) = \{\pi_1, \dots, \pi_m\}^1 \\
\rho(\neg \phi, S, \Pi, t) &= -\rho(\phi, S, \Pi, t) \\
\rho(\phi_1 \wedge \phi_2, S, \Pi, t) &= \min(\rho(\phi_1, S, \Pi, t), \rho(\phi_2, S, \Pi, t)) \\
\rho(\phi_1 \cup \phi_2, S, \Pi, t) &= \sup_{t' \geq t} \min\{\rho(\phi_2, S, \Pi, t'), \inf_{t'' \in [t, t']} \rho(\phi_1, S, \Pi, t'')\}
\end{aligned}$$

It is an easy exercise to show that for continuous-time signals this quantitative semantics of HyperSTL is a conservative extension of the quantitative semantics of STL discussed above. For discrete-time signals it is important to understand that discrete time points often represent points in continuous time. It is widely accepted, that this can be cast into a (strictly monotonic) timing function $\tau : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ [3, 14]. The HyperSTL semantics given above is meaningful in a discrete-time setting if all traces share the same timing function.

HyperSTL characterisation. As discussed in Section 2.2, robust cleanness is a hyperproperty. Recent work on testing and monitoring of robust cleanness [6] explains the difficulties of monitoring such hyperproperties. In essence, it turns out that the first condition of Definition 2 cannot be refuted by observing a real system. Intuitively, this is because this condition effectively puts a constraint on the lower bound of the size of the sets of outputs that a system must be able to produce whereas the second condition enforces an upper bound. A violation of the upper-bound constraint is irrevocable, i.e., once observed, the system is for sure not robustly clean. However not having observed an output that is larger than the lower bound, does not exclude the possibility for observing such an output in the future. We therefore follow [6], and focus only on the second

¹ We admit some sloppiness; the set $\text{dom}(\Pi)$ should have a fixed order.

condition of the robust cleanness definition in our work on falsification. For the HyperLTL characterisation this means that we only work with the second formula, labelled (2).

The HyperLTL characterisation (2) assumes the system to be a subset of $(2^{\text{AP}})^\omega$ and works with distances between traces by means of a Boolean encoding into atomic propositions. We will describe how to transform the HyperLTL formula (2) into a HyperSTL formula, where systems are given as subsets of $(\mathcal{T} \rightarrow \mathbb{R}^n)$ for some width $n \in \mathbb{N}$. Robust cleanness distinguishes between inputs and outputs, and we assume that the input set In and the output set Out are represented as signals of width m , respectively width l . The system space then is $\text{S}_{\text{STL}} \subseteq (\mathcal{T} \rightarrow \mathbb{R}^{m+l})$. Solely for the sake of clarity, we will in the sequel, unless otherwise stated, restrict to $m = l = 1$, i.e., $\text{In} \subseteq \mathbb{R}$ and $\text{Out} \subseteq \mathbb{R}$, and thus work with a fixed width of 2, hence $\text{S}_{\text{STL}} \subseteq (\mathcal{T} \rightarrow \text{In} \times \text{Out})$.

We can assume a set $\text{Std} \subseteq \text{S}_{\text{STL}}$ as given, which defines all standard behaviours of the system. The HyperSTL characterisation of the HyperLTL formula (2) is then

$$\begin{aligned} & \forall \pi_1. \forall \pi_2. \exists \pi'_1. \text{Std}_{\pi_1} > 0 \rightarrow \\ & \left(\text{G}(|i_{\pi_1} - i_{\pi'_1}| \leq 0 \wedge \text{Std}_{\pi'_1} > 0) \wedge \right. \\ & \left. ((d_{\text{Out}}(o_{\pi'_1}, o_{\pi_2}) - \kappa_o \leq 0) \text{ W } (d_{\text{In}}(i_{\pi'_1}, i_{\pi_2}) - \kappa_i > 0)) \right) \end{aligned} \quad (3)$$

The quantifiers remain unchanged relative to (2). The predicate $\text{Std}_{\text{In}_{\pi_1}}$ that holds if and only if π_1 is a standard input, is replaced by the function Std_{π_1} which returns a positive value if π_1 is in Std , and a non-positive value otherwise. The input equality requirement of π_1 and π'_1 is ensured by globally enforcing $|i_{\pi_1} - i_{\pi'_1}| \leq 0$.

Since we switched from the concept of standard inputs to the concept of standard traces, we must also check that π'_1 is a standard trace. This echoes the setup in Definition 2, where the second requirement asks for a trace $\sigma'' \in \text{Std}$ instead of a trace from S_{IO} , see [6] for an elaborate discussion. In the operands of the Weak-Until operator W , we replace the AP-encoded versions of \hat{d}_{In} and \hat{d}_{Out} by the original distance functions d_{In} and d_{Out} , and we perform simple arithmetic operations to match the syntactic requirements of HyperSTL.

We remark that for encoding Std_{π} , due to the absence of the Next-operator in HyperSTL, it might be necessary to add a clock signal $s(t) = t$ to traces in a preprocessing step, not considered here for the sake of avoiding cluttered notation.

Quantifier Elimination. In many practical settings—where the different standard behaviours are spelled out upfront explicitly, as in NEDC and WLTC—it can be assumed that the number of distinct standard behaviours Std is finite (while there are infinitely many possible behaviours in S_{STL}). Finiteness of Std makes it possible to remove the quantifiers by enumeration, and opens the way to work with the STL fragment of HyperSTL, after proper adjustments.

Let $\text{Std} = \{w_1, \dots, w_c\}$ be an arbitrary standard set with c unique standard traces. We will demonstrate the quantifier elimination by substituting by the placeholder $V(\pi_1, \pi_2)$ the subformula (starting with $\exists\pi'_1 \dots$) of formula (3) behind the second quantification. We can switch the order of the \forall -quantifiers without changing the semantics of the formula, so we are working with $\forall\pi_2. \forall\pi_1. V(\pi_1, \pi_2)$. Then, by replacing the second quantifier with the infinite conjunction [23], we get

$$\forall\pi_2. \bigwedge_{w \in \text{S}_{\text{STL}}} V(w, \pi_2).$$

The latter can be split into a finite and an infinite conjunction

$$\forall\pi_2. \bigwedge_{w \in \text{Std}} V(w, \pi_2) \wedge \bigwedge_{w \in \text{S}_{\text{STL}} \setminus \text{Std}} V(w, \pi_2). \quad (4)$$

Let $W(\pi_1, \pi_2, \pi'_1)$ be the placeholder, such that $V(\pi_1, \pi_2) = \exists\pi'_1. \text{Std}_{\pi_1} > 0 \rightarrow W(\pi_1, \pi_2, \pi'_1)$. Unfolding V in the right (infinite) conjunction in formula (4) reveals

$$\bigwedge_{w \in \text{S}_{\text{STL}} \setminus \text{Std}} \exists\pi'_1. \text{Std}_w > 0 \rightarrow W(w, \pi_2, \pi'_1).$$

It follows directly from the definition of Std_π that for all $w \notin \text{Std}$, Std_w is non-positive. Hence that fragment of the formula is trivially fulfilled, and formula (4) is equivalent to

$$\forall\pi_2. \bigwedge_{w \in \text{Std}} V(w, \pi_2).$$

Combined with similar reasoning for the \exists -operator and disjunctions we can altogether rewrite formula (3) into

$$\bigwedge_{w \in \text{Std}} \bigvee_{w' \in \text{Std}} \left(\mathbf{G}(|i_w - i_{w'}| \leq 0) \wedge \right. \quad (5) \\ \left. ((d_{\text{Out}}(\mathbf{o}_{w'}, \mathbf{o}) - \kappa_{\mathbf{o}} \leq 0) \mathbf{W} (d_{\text{In}}(i_{w'}, i) - \kappa_i > 0)) \right),$$

where the \forall -quantification over π_1 is replaced by the conjunction over standard traces w , the \exists -quantification of π'_1 by the disjunction over standard traces w' , and the remaining \forall -quantification of π_2 is eliminated by rewriting into a trace formula and removing the trace indices from i_{π_2} and \mathbf{o}_{π_2} .

Self-composition in logic. Formula (5) is not yet an STL formula, because the distance function d_{In} needs to compare the trace input with inputs of constant traces from the set Std . A popular technique to analyse hyperproperties is self-composition of a system [4, 15]. We use a syntactic variant of parallel self-composition as follows. For a trace width n , we compose the signals of the trace under investigation $w = (s_1, \dots, s_n)$ and the signals of each of the, say, c standard traces $\{(s_{11}, \dots, s_{1n}), \dots, (s_{c1}, \dots, s_{cn})\} = \text{Std}$. The composed trace then is of width $n + nc$, it is $w' = (s_1, \dots, s_n, s_{11}, \dots, s_{n1}, \dots, s_{1c}, \dots, s_{nc})$. For

the restricted case considered here (one-dimensional input and output signals), $w' = (i, o, i_1, o_1, \dots, i_c, o_c)$ is of trace width $2 + 2c$. The resulting STL formula for monitoring robust cleanness is

$$\bigwedge_{1 \leq a \leq c} \bigvee_{1 \leq b \leq c} \left(G(|i_a - i_b| \leq 0) \wedge \right. \quad (6) \\ \left. ((d_{\text{Out}}(o_b, o) - \kappa_o \leq 0) \mathbf{W} (d_{\text{In}}(i_b, i) - \kappa_i > 0)) \right).$$

Recall that a discrete time interpretation of such a formula requires all system traces to share the same timing function τ .

Embedding into the mixed-IO model. The STL formula (6) still is bound to inputs and outputs forming pairs synchronized in time. A more realistic scenario is that of inputs and outputs occurring independently of each other. In particular, when testing a real-world CPS, the testing interface can either pass an input to the system under test or receive an output, but not both at the same time. Furthermore, certain tests require to pass a series of inputs before receiving an output at all [6]. The mixed-IO model supports such real-world testing scenarios. Mixed-IO signals are always defined in the discrete time domain. A mixed-IO signal $s \in (\text{In} \cup \text{Out})^\omega$ (or, equivalently, $s : \mathbb{N} \rightarrow \text{In} \cup \text{Out}$) is similar to a real-valued discrete-time signal but the value domain \mathbb{R} is replaced by the domain $\text{In} \cup \text{Out}$. A discrete-time mixed-IO trace $w = (s_1, \dots, s_n) \in ((\text{In} \cup \text{Out})^\omega)^n$ is a tuple of n mixed-IO signals. Accordingly, predicates of the form $f > 0$ must use functions f that produce real values for mixed-IO signals. Formula (6) requires that all traces share the same timing function. For continuous-time signals, we ensure that this condition is met by transforming all traces into traces with a common value frequency (say, 1 Hz) by averaging the values observed in a time unit (of one second). Let $w = (s_1, \dots, s_n) \in (\mathbb{N} \rightarrow \text{In} \cup \text{Out})^n$ be a recorded trace with some timing function $\tau : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$, that is sampled with at least one value per time unit, i.e., $\tau(i+1) - \tau(i) \leq 1$ for all $i \in \mathbb{N}$. This trace is condensed to a new trace $w' = (s'_1, \dots, s'_n)$ with timing function $\tau'(i) = i$, and $s'_j(t) := \text{average}(\cup_{\tau(i) \in [t, t+1)} s_j(i))$ for $1 \leq j \leq n$, i.e., each signal s'_j is piecewise constant: for each unit time interval $[t, t+1)$ the signal value is set to the average of all signal values originally recorded in that unit time interval.

For adjusting formula (6), let $\text{Std} = \{s_1, \dots, s_c\} \subseteq (\text{In} \cup \text{Out})^\omega$ be a set of standard traces, each in the form of a single mixed-IO signal. Following the syntactic self-composition idea from above, the composition of a trace w under investigation with Std is the trace $w' = (w, s_1, \dots, s_c) \in ((\text{In} \cup \text{Out})^\omega)^{c+1}$. This needs two subtle adjustments of the formula. First, the distances d_{In} and d_{Out} are replaced by their mixed-IO counterparts \bar{d}_{In} and \bar{d}_{Out} , and instead of directly accessing inputs and outputs, the current value is projected to the input and output domain, respectively. Second, since the set In and Out is opaque, the expression $|i_a - i_b|$ is not evaluable any more, it is replaced by the distance \bar{d}

based on \bar{d}_{In} and \bar{d}_{Out} . The resulting formula is

$$\bigwedge_{1 \leq a \leq c} \bigvee_{1 \leq b \leq c} \left(\mathsf{G}(\bar{d}(s_a \downarrow_i, s_b \downarrow_i) \leq 0) \wedge \right. \quad (7) \\ \left. ((\bar{d}_{\text{Out}}(s_b \downarrow_o, s \downarrow_o) - \kappa_o \leq 0) \mathsf{W} (\bar{d}_{\text{In}}(s_b \downarrow_i, s \downarrow_i) - \kappa_i > 0)) \right),$$

where \bar{d} is defined for some $\varepsilon > 0$ as

$$\bar{d}(x, y) := \begin{cases} 0, & \text{if } x = y \\ \bar{d}_{\text{In}}(x, y) + \varepsilon, & \text{if } x \neq y \wedge x, y \in \text{In} \\ \bar{d}_{\text{Out}}(x, y) + \varepsilon, & \text{if } x \neq y \wedge x, y \in \text{Out} \\ \infty, & \text{otherwise.} \end{cases}$$

In the second and third clause of the above definition we add some positive value ε to the result of \bar{d}_{In} and \bar{d}_{Out} , because they are pseudometrics, and $\bar{d}_{\text{In}}(i_1, i_2)$ could be 0 even if $i_1 \neq i_2$. For the correctness of formula (7), however, it is crucial that $\bar{d}(x, y) = 0$ if and only if $x = y$. For a good performance of the falsification algorithm, we will nevertheless want to make use of \bar{d}_{In} and \bar{d}_{Out} if $i_1 \neq i_2$. We remark that \bar{d} is not a metric, because the triangle inequality requirement now is violated.

The discussion above has assembled all the details to formally back the following theorem, stating that a system satisfies formula (7) if and only if it satisfies the second condition of robust cleanness in Definition 2.

Theorem 1. *Let $\mathcal{C} = \langle \text{Std}, \bar{d}_{\text{In}}, \bar{d}_{\text{Out}}, \kappa_i, \kappa_o \rangle$ be a contract for some system $\mathsf{S}_{\text{IO}} \subseteq (\text{In} \cup \text{Out})^\omega$ with $\text{Std} = \{\sigma_1, \dots, \sigma_c\} \subseteq \mathsf{S}_{\text{IO}}$, and let ϕ denote formula (7). Then, for all $\sigma' \in \mathsf{S}_{\text{IO}}$, it holds $(\sigma', \sigma_1, \dots, \sigma_c), 0 \models \phi$ if and only if for all $\sigma \in \text{Std}$ and $k \geq 0$ such that $\bar{d}_{\text{In}}(\sigma[j] \downarrow_i, \sigma'[j] \downarrow_i) \leq \kappa_i$ holds for all $j \leq k$, there exists $\sigma'' \in \text{Std}$ such that $\sigma \downarrow_i = \sigma'' \downarrow_i$ and $\bar{d}_{\text{Out}}(\sigma'[k] \downarrow_o, \sigma''[k] \downarrow_o) \leq \kappa_o$.*

Example 2. We consider $\mathcal{C} = \langle \text{Std}, \bar{d}_{\text{In}}, \bar{d}_{\text{Out}}, \kappa_i, \kappa_o \rangle$ where $\text{Std} = \{w_1, w_2\}$ contains the two standard traces $w_1 = 1_i 2_i 3_i 7_o 0_i \delta^\omega$ and $w_2 = 0_i 1_i 2_i 3_i 6_o \delta^\omega$. We here decorate inputs with index i and outputs with index o , i.e., w_1 describes a system receiving the three inputs 1, 2, and 3, then producing the output 7, and finally receiving input 0 before entering quiescence. We take

$$\bar{d}_{\text{Out}}(o_1, o_2) = \begin{cases} |o_1 - o_2|, & \text{if } o_1, o_2 \in \text{Out} \setminus \{\delta\} \\ 0, & \text{if } o_1 = o_2 = \text{--}_o \text{ or } o_1 = o_2 = \delta \\ \infty, & \text{otherwise,} \end{cases}$$

and similarly for \bar{d}_{In} . The contractual value thresholds are assumed to be $\kappa_i = 1$ and $\kappa_o = 6$.

Assume we are observing the trace $w = 0_i 1_i 2_i 6_o 0_i \delta^\omega$ to be monitored with formula (7). First notice, that for combinations of a and b in (7), where $a \neq b$, the subformula $\mathsf{G}(\bar{d}(s_a \downarrow_i, s_b \downarrow_i) \leq 0)$ is always false, because s_1 and s_2 (i.e., the

combination of w_1 and w_2) have different values at time point 0. Hence, it remains to show that

$$(\bar{d}_{\text{Out}}(w_1 \downarrow_{\circ}, w \downarrow_{\circ}) - \kappa_{\circ} \leq 0) \text{ W } (\bar{d}_{\text{In}}(w_1 \downarrow_i, w \downarrow_i) - \kappa_i > 0) \wedge \\ (\bar{d}_{\text{Out}}(w_2 \downarrow_{\circ}, w \downarrow_{\circ}) - \kappa_{\circ} \leq 0) \text{ W } (\bar{d}_{\text{In}}(w_2 \downarrow_i, w \downarrow_i) - \kappa_i > 0).$$

For the first part, the input distance between inputs in w and w_1 is always 1 at positions 1 to 3, it is 0 at position 4 (because \neg_i is compared to \neg_i) and in position 5 and beyond. Thus, $\bar{d}_{\text{In}}(w_1 \downarrow_i, w \downarrow_i) - \kappa_i$ is always at most 0, and the right hand-side of the W operator is always false. Consequently, by definition of W , the left operand of W must always hold, i.e., $\bar{d}_{\text{Out}}(w_1 \downarrow_{\circ}, w \downarrow_{\circ})$ must always be less or equal to 6. This is the case for w_1 and w : at all positions except for 4, \neg_{\circ} is compared to \neg_{\circ} (or δ to δ), so the difference is 0, and at position 4, the distance of 6 and 7 is 1.

For the second W -formula, w is compared to w_2 . These two traces are comparable only to a limited extent: the order of input and output is altered at the last two positions of the signals before quiescence. Hence, the right operand of W is true at position 4, and the formula holds for the remaining trace. For positions 1 to 3, the input distances are 0, because the input values are identical. At these positions, the left operand must hold. The values are input values, so \neg_{\circ} is compared to \neg_{\circ} at each position. This distance is defined to be 0, so it holds that $-6 \leq 0$, and the formula is satisfied. Since both formulas hold, the conjunction of both holds, too, and trace w is qualified as robustly clean. There could however be other system traces not considered in this example, that overall could violate robust cleanness of the system.

Restriction of input space. Robust cleanness puts semantic requirements on fragments of a system's input space, outside of which the system's behaviour remains unspecified. Typically, the fragment of the input space covered is rather small. To falsify the STL formula (7), the falsifier has two challenging tasks. First, it has to find a way to stay in the relevant input space, i.e., select inputs with a distance of at most κ_i from the standard behaviour. Only if this is assured it can search for an output large enough to violate the κ_{\circ} requirement. In this, a large robustness estimate provided by the quantitative semantics of STL cannot serve as an indicator for deciding whether an input is too far off or whether an output stays too close to the standard behaviour.

The general strength of the falsification technique is its proven ability to discover outputs of a black-box system violating a property. That is why the technique is considered suitable for real-world robust cleanness tests. We can improve its efficiency significantly by narrowing upfront the input space the falsifier uses.

In practice, test execution traces will always be finite. In previous real-life doping tests, test execution lengths have been bounded by some constant $B \in \mathbb{N}$ [6], i.e., systems are represented as sets of finite traces $S \subseteq (\text{In} \cup \text{Out})^B$ (which for formality reasons each can be considered suffixed with δ^{ω}). In this bounded horizon, we can provide a predicate discriminating between relevant

and irrelevant input sequences. Formally, the restriction to the relevant input space fragment of a system $S \subseteq (\text{In} \cup \text{Out})^B$ is given by the set $\text{In}_{\text{Std}, \kappa_i} = \{w \in S \mid \exists w' \in \text{Std}. \bigwedge_{k=0}^{B-1} (\bar{d}_{\text{In}}(w(k) \downarrow_i, w'(k) \downarrow_i) \leq \kappa_i)\}$. Since Std and B are finite, membership is computable.

There are rare cases in which this optimisation may prevent the falsifier from finding a counterexample. This is only the case if there is an input prefix leading to a violation of the formula for which there is no suffix such that the whole trace satisfies the κ_i constraint. Below is a pathological example in which this could make a difference.

Example 3. Apart from NO_x emissions, NEDC (and WLTC) tests are used to measure fuel consumption. Consider a contract similar to the contracts above, but with fuel rate as the output quantity. Assuming a “normal” fuel rate behaviour during the standard test, there might be a test within a reasonable κ_i distance, where the fuel is wasted insanely. Then, the fuel tank might run empty before the intended end of the test, which therefore could not be finished within the κ_i distance, because speed would be constantly 0 at the end. The actually driven test is not in set $\text{In}_{\text{Std}, \kappa_i}$, but there is a prefix within κ_i distance that violates the robust cleanness property.

4 Diesel Emissions

This section discusses how to tailor the generic probabilistic falsification approach for STL based on Algorithm 1 to the particular case of diesel emissions, and reports on empirical observations when putting the approach into practice.

Robustness. In the case of diesel emissions doping, the only standard behaviour is either the NEDC or the WLTC. Assuming, for example, NEDC, let $\mathcal{C} = \langle \{\text{NEDC} \circ \text{o}\}, \kappa_i, \kappa_o, d_{\text{In}}, d_{\text{Out}} \rangle$ be a diesel emissions specific contract, where NEDC is the sequence of 1180 inputs with the k th input defining the speed of the car after k seconds from the beginning of the test. Here, the output o suffixed to NEDC is the (average) amount of emitted NO_x during the NEDC drive. By restricting the input space to $\text{In}_{\{\text{NEDC} \circ \text{o}\}, \kappa_i}$ as explained in Section 3, formula (7) can be simplified to

$$\mathbf{G}(\bar{d}_{\text{Out}}((\text{NEDC} \circ \text{o}) \downarrow_o, s \downarrow_o) - \kappa_o \leq 0). \quad (8)$$

This is because the conjunction and disjunction over standard traces becomes obsolete for only a single standard trace. For the same reason, the requirement $\mathbf{G}(\bar{d}(s_a \downarrow_i, s_b \downarrow_i) \leq 0)$ becomes obsolete, as the compared traces are always identical. In the \mathbf{W} subformula, the right proposition is always false, because of the restricted input space: the proposition collapses to $\bar{d}_{\text{In}}(\text{NEDC} \circ \text{o} \downarrow_i, s \downarrow_i) - \kappa_i > 0$ and the input domain $\text{In}_{\{\text{NEDC} \circ \text{o}\}, \kappa_i}$ is $\{(s) \in S \mid \forall k \in [0, 1180]. \bar{d}_{\text{In}}(s(k) \downarrow_i, (\text{NEDC} \circ \text{o})(k) \downarrow_i) \leq \kappa_i\}$. And thus, by the definition of \mathbf{W} and \mathbf{U} , the \mathbf{W} subformula is equivalent to formula (8). We implemented Algorithm 1 for the robustness computation according to formula (8).

Emissions Approximation. In practice, running tests like NEDC with real cars is a time consuming and expensive endeavour. Furthermore, tests on chassis dynamometers are usually prohibited to be carried out with rented cars by the rental companies. On the other hand, car emission models for simulation are not available to the public—and models provided by the manufacturer cannot be considered trustworthy. To carry out our experiments, we instead use an approximation technique that estimates the amount of NO_x emissions of a car along a certain trajectory based on data recorded during previous trips with the same car, sampled at a frequency of 1 Hz (one sample per second). Notably, these trips do not need to have much in common with the trajectory to be approximated. A trip is represented as a finite sequence $\mathcal{T} \in (\mathbb{R} \times \mathbb{R} \times \mathbb{R})^*$ of triples, where each such triple (v, a, n) represents the speed, the acceleration and the absolute amount of NO_x emitted at a particular time instant in the sample. Speed and acceleration can be considered as the main parameters influencing the instant emission of NO_x . This is, for instance, reflected in the RDE regulation [16, 24] where the decisive quantities to validate the test route and driving behaviour during RDE tests are speed and acceleration.

A recording \mathcal{D} is the union of finitely many trips \mathcal{T} . We can turn such a recording into a predictor \mathcal{P} of the NO_x values given pairs of speed and acceleration as follows:

$$\mathcal{P}(v, a) = \text{average}[n \mid (\exists v', a'. (|v - v'| \leq 2 \wedge |a - a'| \leq 2 \wedge (v', a', n) \in \mathcal{D}))].$$

The amount of NO_x assigned to a pair (v, a) here is the average of all NO_x values seen in the recording \mathcal{D} for $v \pm \ell$ and $a \pm \ell$, with $0 \leq \ell \leq 2$. To overcome measurement inaccuracies and to increase the robustness of the approximated emissions, the speed and acceleration may deviate up to 2 km/h, and 2 m/s², respectively. This tolerance is adopted from the official NEDC regulation [26], which allows up to 2 km/h of deviations while driving the NEDC.

Experiment setup. To demonstrate the practical applicability of our implementation of Algorithm 1 and our NO_x approximation, we report here on two experiments. For the first experiment, we use recordings from Biewer et al. [8]. They used the app *LolaDrives* to perform low-cost RDE tests and recorded the data received from a car’s diagnosis port. Using the two RDE recordings that appear in their work, the above predictor can be used to estimate the NO_x emission during NEDC to be 86 mg/km. Their car was an Audi A6 Avant Diesel, admitted in June 2020. We rented the successor of this car model, admitted in 2021, and recorded three low-cost RDE trips with the help of *LolaDrives*. The new version of this car turned out to have a significantly better emission cleaning system: the estimated amount of NO_x emitted during the NEDC is 9 mg/km. In the sequel, we will refer to the first car as *A20* and to the second as *A21*. Car A20 has previously been falsified w.r.t. the RDE specification. Neither A20 nor A21 has been falsified w.r.t. robust cleanness.

Contracts. Before turning to falsification, we need to spell out meaningful contracts. The input domain $\text{In} \subseteq \mathbb{R}^*$ is the set of finite speed trajectories, and

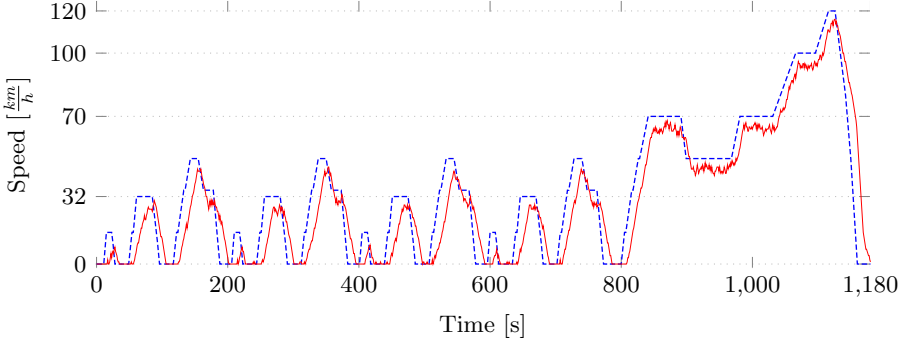


Fig. 3: NEDC speed profile (blue, dashed) and input falsifying \mathcal{C} for $\kappa_o = 88$ mg/km (red) with 182 mg/km of emitted NO_x .

the set $\text{Out} \subseteq \mathbb{R}$ represents the average amount of NO_x emitted during the test. d_{In} must be past-forgetful, hence only the last speed value in each trace must be considered. A natural distance function for inputs is $d_{\text{In}}(v_1, v_2) = |v_1 - v_2|$. Similarly, a measurement for the distance of outputs is $d_{\text{Out}}(o_1, o_2) = |o_1 - o_2|$. Adding the necessary technicalities for the mixed-IO setting, we get \bar{d}_{In} and \bar{d}_{Out} as defined in Example 2. For κ_i , it turned out that $\kappa_i = 15$ km/h is a reasonable choice, as it leaves enough flexibility for human-caused driving mistakes and intended deviations [6]. The threshold for NO_x emissions under lab conditions is 80 mg/km. The emission limits for RDE tests depend on the admission date of the car. Cars admitted in 2020 or earlier, must emit 168 mg/km at most, and cars admitted later must adhere to the limit of 120 mg/km. For our experiments, we use $\kappa_o = 88$ mg/km for A20 and $\kappa_o = 40$ mg/km for A21 to have the same tolerances as for RDE tests. Effectively, the upper threshold for A20 is $84 + 88 = 172$ mg/km, and for A21 the limit is $9 + 40 = 49$ mg/km. Notice that for software doping analysis, the output observed for a certain standard behaviour and the constant κ_o define the effective threshold; this threshold is typically different from the threshold defined by the regulation.

Evaluation. We modified Algorithm 1 by adding a timeout condition, i.e., if the algorithm is not able to find a falsifying counterexample within 3,000 iterations, it terminates and returns both the trace for which the smallest robustness has been observed and its corresponding robustness value. Hence, if falsification of robust cleanness for a system is not possible, the algorithm outputs an upper bound on how robust the system satisfies robust cleanness.

For the concrete case of the diesel emissions, the robustness value during the first 1180 inputs (sampled from the restricted input space $\text{In}_{\text{Std}, \kappa_i}$) is always κ_o . When the NEDC output o_{NEDC} and the non-standard output o are compared, the robustness value is $\kappa_o - |o_{\text{NEDC}} - o|$ (cf., eq. (8), the quantitative semantics of STL, and definition of \bar{d}_{Out}). Hence, for test cycles with small robustness values,

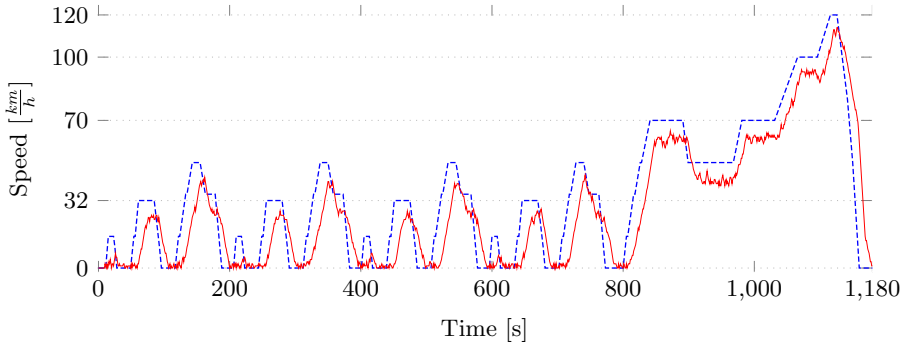


Fig. 4: NEDC speed profile (blue, dashed) and input maximising NO_x emissions to 11 mg/km (red).

we get NO_x emissions o that are either very small or very large compared to o_{NEDC} . We ran the modified Algorithm 1 on A20 and A21 for the contracts defined above. For A20, it found a robustness value of -8 , i.e., it was able to falsify robust cleanness relative to the assumed contract and found a test cycle for which NO_x emissions of 182 mg/km are predicted. The test cycle is shown in Fig. 3. For A21, the smallest robustness estimate found—even after 100 independent executions of the algorithm—was 38, i.e., A21 is predicted to satisfy robust cleanness with a very high robustness upper bound. The corresponding test cycle is shown in Fig. 4.

5 Conclusion & Future Work

This paper marks an important milestone in making software doping tests of real-world CPS practically feasible. Regarding test execution effort, real-world testing of CPS is not scalable; the number of tests realistically executable is usually very limited. Probabilistic falsification has its strength in repetitive testing of a system model in a strategic way. We improved this approach by embedding it into a very natural problem solving strategy: Patiently observing the system in-the-wild for the purpose of eventually conducting a small set of doping tests in an even more strategic way. With this paper, we have laid the formal foundations, and we have carved out the aspects that dominate practical applicability. For the latter we focussed on the automotive emissions context. In that context, we are currently spending considerable effort on the acquisition of more high-quality training data. We are building a car data platform (CDP) as a central place for automotive data, which, most importantly, includes the app LolaDrives for convenient recording, uploading and crowd-sourcing of data. With increasing amounts of data collected we hope to be able to roll out predictions that are more and more precise. Finally, we will extend the approach to broader application contexts, to make software doping tests available across the wider CPS domain.

References

1. Abbas, H., Fainekos, G.E., Sankaranarayanan, S., Ivancic, F., Gupta, A.: Probabilistic temporal logic falsification of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.* **12**(2s), 95:1–95:30 (2013). <https://doi.org/10.1145/2465787.2465797>
2. Adroit, A.: Software-defined everything (SDE) market perspective (2021–2027): Cisco Systems Inc, Dell Inc, EMC Corp, Extreme Networks, Fujitsu Ltd, Hewlett Packard Enterprise. New Mexico Tribune (2021), <https://nmtribune.com/uncategorized/199383/software-defined-everything-sde-market-perspective-2021-2027-cisco-systems-inc-dell-inc-emc-corp-extreme-networks-fujitsu-ltd-hewlett-packard-enterprise/>, Online; accessed: 2021-07-13
3. Alur, R., Henzinger, T.A.: Real-time logics: Complexity and expressiveness. In: *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90)*, Philadelphia, Pennsylvania, USA, June 4–7, 1990. pp. 390–401. IEEE Computer Society (1990). <https://doi.org/10.1109/LICS.1990.113764>
4. Barthe, G., D’Argenio, P.R., Rezk, T.: Secure information flow by self-composition. *Mathematical Structures in Computer Science* **21**(6), 1207–1252 (2011). <https://doi.org/10.1017/S0960129511000193>
5. Biewer, S., D’Argenio, P., Hermanns, H.: Doping tests for cyber-physical systems. In: Parker, D., Wolf, V. (eds.) *Quantitative Evaluation of Systems*, 16th International Conference, QEST 2019, Glasgow, UK, September 10–12, 2019, *Proceedings. Lecture Notes in Computer Science*, vol. 11785, pp. 313–331. Springer (2019). https://doi.org/10.1007/978-3-030-30281-8_18
6. Biewer, S., D’Argenio, P.R., Hermanns, H.: Doping tests for cyber-physical systems. *ACM Trans. Model. Comput. Simul.* **31**(3), 16:1–16:27 (2021). <https://doi.org/10.1145/3449354>
7. Biewer, S., Dimitrova, R., Fries, M., Gazda, M., Heinze, T., Hermanns, H., Mousavi, M.R.: Conformance Relations and Hyperproperties for Doping Detection in Time and Space. *Logical Methods in Computer Science* **18**(1), 14:1–14:39 (2022). [https://doi.org/10.46298/lmcs-18\(1:14\)2022](https://doi.org/10.46298/lmcs-18(1:14)2022)
8. Biewer, S., Finkbeiner, B., Hermanns, H., Köhl, M.A., Schnitzer, Y., Schwenger, M.: RT Lola on board: Testing real driving emissions on your phone. In: Groote, J.F., Larsen, K.G. (eds.) *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II. Lecture Notes in Computer Science*, vol. 12652, pp. 365–372. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_20
9. Brim, L., Dluhos, P., Safranek, D., Vejpustek, T.: STL*: Extending signal temporal logic with signal-value freezing operator. *Inf. Comput.* **236**, 52–67 (2014). <https://doi.org/10.1016/j.ic.2014.01.012>
10. Chib, S., Greenberg, E.: Understanding the metropolis-hastings algorithm. *The american statistician* **49**(4), 327–335 (1995). <https://doi.org/10.1080/00031305.1995.10476177>
11. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) *POST 2014. LNCS*, vol. 8414, pp. 265–284. Springer (2014). https://doi.org/10.1007/978-3-642-54792-8_15
12. D’Argenio, P.R., Barthe, G., Biewer, S., Finkbeiner, B., Hermanns, H.: Is your software on dope? - Formal analysis of surreptitiously “enhanced” programs. In:

- Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Proceedings. LNCS, vol. 10201, pp. 83–110. Springer (2017). https://doi.org/10.1007/978-3-662-54434-1_4
13. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13–19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 264–279. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_19
 14. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.* **410**(42), 4262–4291 (2009). <https://doi.org/10.1016/j.tcs.2009.06.021>
 15. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Kroening, D., Pasareanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 30–48. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_3
 16. Köhl, M.A., Hermanns, H., Biewer, S.: Efficient monitoring of real driving emissions. In: Colombo, C., Leucker, M. (eds.) Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10–13, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11237, pp. 299–315. Springer (2018). https://doi.org/10.1007/978-3-030-03769-7_17
 17. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22–24, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3253, pp. 152–166. Springer (2004). https://doi.org/10.1007/978-3-540-30206-3_12
 18. Mathews, M.: Are You Ready for Software-Defined Everything? Wired, <https://www.wired.com/insights/2013/05/are-you-ready-for-software-defined-everything/>, Online; accessed: 2021-07-13
 19. Meinke, K., Sindhu, M.A.: Incremental learning-based testing for reactive systems. In: Gogolla, M., Wolff, B. (eds.) Tests and Proofs - 5th International Conference, TAP@TOOLS 2011, Zurich, Switzerland, June 30 - July 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6706, pp. 134–151. Springer (2011). https://doi.org/10.1007/978-3-642-21768-5_11
 20. Nghiem, T., Sankaranarayanan, S., Fainekos, G.E., Ivancic, F., Gupta, A., Pappas, G.J.: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In: Johansson, K.H., Yi, W. (eds.) Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12–15, 2010. pp. 211–220. ACM (2010). <https://doi.org/10.1145/1755952.1755983>
 21. Nguyen, L.V., Kapinski, J., Jin, X., Deshmukh, J.V., Johnson, T.T.: Hyperproperties of real-valued signals. In: Talpin, J., Derler, P., Schneider, K. (eds.) Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017, Vienna, Austria, September 29 - October 02, 2017. pp. 104–113. ACM (2017). <https://doi.org/10.1145/3127041.3127058>
 22. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). <https://doi.org/10.1109/SFCS.1977.32>
 23. Rosen, K.H., Krithivasan, K.: Discrete mathematics and its applications: with combinatorics and graph theory. Tata McGraw-Hill Education (2012)

24. The European Parliament and the Council of the European Union: Commission Regulation (EU) 2017/1151 (June 2017), <http://data.europa.eu/eli/reg/2017/1151/oj>
25. Tutuianu, M., Bonnel, P., Ciuffo, B., Haniu, T., Ichikawa, N., Marotta, A., Pavlovic, J., Steven, H.: Development of the world-wide harmonized light duty test cycle (wltc) and a possible pathway for its introduction in the european legislation. Transportation Research Part D: Transport and Environment **40**(Supplement C), 61 – 75 (2015). <https://doi.org/10.1016/j.trd.2015.07.011>
26. United Nations: UN Vehicle Regulations - 1958 Agreement, Revision 2, Addendum 100, Regulation No. 101, Revision 3 — E/ECE/324/Rev.2/Add.100/Rev.3 (2013), <http://www.unece.org/trans/main/wp29/wp29regs101-120.html>
27. Volpato, M., Tretmans, J.: Approximate active learning of nondeterministic input output transition systems. Electron. Commun. Eur. Assoc. Softw. Sci. Technol. **72** (2015). <https://doi.org/10.14279/tuj.eceasst.72.1008>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

