Doping Tests for Cyber-Physical Systems

Sebastian Biewer[1(✉)], Pedro D'Argenio[1,2,3], and Holger Hermanns[1,4]

[1] Saarland University, Saarland Informatics Campus, Saarbücken, Germany
biewer@depend.uni-saarland.de
[2] FAMAF, Universidad Nacional de Córdoba, Córdoba, Argentina
[3] CONICET, Córdoba, Argentina
[4] Institute of Intelligent Software, Guangzhou, China

Abstract. The software running in embedded or cyber-physical systems (CPS) is typically of proprietary nature, so users do not know precisely what the systems they own are (in)capable of doing. Most malfunctionings of such systems are not intended by the manufacturer, but some are, which means these cannot be classified as bugs or security loopholes. The most prominent examples have become public in the diesel emissions scandal, where millions of cars were found to be equipped with software violating the law, altogether polluting the environment and putting human health at risk. The behaviour of the software embedded in these cars was intended by the manufacturer, but it was not in the interest of society, a phenomenon that has been called *software doping*. Doped software is significantly different from buggy or insecure software and hence it is not possible to use classical verification and testing techniques to discover and mitigate software doping.

The work presented in this paper builds on existing definitions of software doping and lays the theoretical foundations for conducting software doping tests, so as to enable attacking evil manufacturers. The complex nature of software doping makes it very hard to effectuate doping tests in practice. We explain the biggest challenges and provide efficient solutions to realise doping tests despite this complexity.

1 Introduction

Embedded and cyber-physical systems are becoming more and more widespread as part of our daily life. Printers, mobile phones, smart watches, smart home equipment, virtual assistants, drones and batteries are just a few examples. Modern cars are even composed of a multitude of such systems. These systems can have a huge impact on our lives, especially if they do not work as expected. As a result, numerous approaches exist to assure quality of a system. The classical and most common type of malfunctioning is what is widely called "bug". Usually, a bug is a very small mistake in the software or hardware that causes a behaviour that is not intended or expected. Other types of malfunctioning are caused by incorrect or wrongly interpreted sensor data, physical deficiencies of a component, or are simply radiation-induced.

Another interesting kind of malfunction (also from an ethical perspective [4]) arises if the expectation of how the system should behave is different for two (or

D. Parker and V. Wolf (Eds.): QEST 2019, LNCS 11785, pp. 313–331, 2019.
https://doi.org/10.1007/978-3-030-30281-8_18

more) parties. Examples for such scenarios are widespread in the context of personal data privacy, where product manufacturers and data protection agencies have notoriously different opinions about how a software is supposed to handle personal data. Another example is the usage of third-party cartridges in printers. Manufacturers and users do not agree on whether their printer should work with third-party cartridges (the user's opinion) or only with those sold by the manufacturer (the manufacturer's opinion). Lastly, an example that received very high media attention are emission cleaning systems in diesel cars. There are regulations for dangerous particles and gases like CO_2 and NO_2 defining how much of these substances are allowed to be emitted during car operation. Part of these regulations are emissions tests, precisely defined test cycles that a car has to undergo on a chassis dynamometer [28]. Car manufacturers have to obey to these regulations in order to get admission to sell a new car model. The central weakness of these regulations is that the relevant behaviour of the car is only a trickle of the possible behaviour on the road. Indeed, several manufacturers equipped their cars with defeat devices that recognise if the car is undergoing an official emissions test. During the test, the car obeys the regulation, but outside test conditions, the emissions extruded are often significantly higher than allowed. Generally speaking, the phenomena described above are considered as incorrect software behaviour by one party, but as intended software behaviour by the other party (usually the manufacturer). In the literature, such phenomena are called software doping [3,10].

The difference between software doping and bugs is threefold: (1) There is a disagreement of intentions about what the software should do. (2) While a bug is most often a small coding error, software doping can be present in a considerable portion of the implementation. (3) Bugs can potentially be detected during production by the manufacturer, whereas software doping needs to be uncovered after production, by the other party facing the final product. Embedded software is typically proprietary, so (unless one finds a way to breach into the intellectual property [9]) it is only possible to detect software doping by observation of the behaviour of the product, i.e., by black-box testing.

This paper develops the foundations for black-box testing approaches geared towards uncovering doped software in concrete cases. We will start off from an established formal notion of robust cleanness (which is the negation of software doping) [10]. Essentially, the idea of robust cleanness is based on a succinct specification (called a "contract") that all involved parties agree on and which captures the intended behaviour of a system with respect to all inputs to the system. Inputs are considered to be user inputs or environmental inputs given by sensors. The contract is defined by input and output distances on standard system trajectories supplemented by input and output thresholds. Simply put, the input distance and threshold induce a tube around the standard inputs, and similar for outputs. For any input in the tube around some standard input the system must be able to react with an output that is in the tube around the output possible according to the standard.

Example 1. For a diesel car the standard trajectory is the behaviour exhibited during the official emissions test cycle. The input distance measures the deviation in car speed from the standard. The input threshold is a small number larger than the acceptable error tolerance of the cycle limiting the inputs considered of interest. The output distance then is the difference between (the total amount of) NO_x extruded by the car facing inputs of interest and that extruded if on the standard test cycle. For cars with an active defeat device we expect to see a violation of the contract even for relatively large output thresholds.

A cyber-physical system (CPS) is influenced by physical or chemical dynamics. Some of this can be observed by the sensors the CPS is equipped with, but some portion might remain unknown, making proper analysis difficult. Nondeterminism is a powerful way of representing such uncertainty faithfully, and indeed the notion of robust cleanness supports non-deterministic reactive systems [10]. Furthermore, the analysis needs to consider (at least) two trajectories simultaneously, namely the standard trajectory and another that stays within the input tube. In the presence of nondeterminism it might even become necessary to consider infinitely many trajectories at the same time. Properties over multiple traces are called hyperproperties [8]. In this respect, expressing robust cleanness as a hyperproperty needs both $\forall$ and $\exists$ trajectory quantifiers. Formulas containing only one type of quantifier can be analysed efficiently, e.g., using model-checking techniques, but checking properties with alternating quantifiers is known to be very complex [7,16]. Even more, testing of such problems is in general not possible. Assume, for example, a property requiring for a (nondeterministic) system that for every input i, there exists the output $o = i$, i.e., one of the system's possible behaviours computes the identity function. For black-box systems with infinite input and output domains the property can neither be verified nor falsified through testing. In order to verify the property, it is necessary to iterate over the infinite input set. For falsification one must show that for some i the system can not produce i as output. However, not observing an output in finitely many steps does not rule out that this output can be generated. As a result, there is no prior work (we are aware of) that targets the automatic generation of test cases for hyperproperties, let alone robust cleanness.

The contribution of this paper is three-fold. (1) We observe that standard behaviour, in particular when derived by common standardisation procedures, can be represented by finite models, and we identify under which conditions the resulting contracts are (un)satisfiable. (2) For a given satisfiable contract we construct the largest non-deterministic model that is robustly clean w.r.t. this contract. We integrate this model into a model-based testing theory, which can provide a non-deterministic algorithm to derive sound test suites. (3) We develop a testing algorithm for bounded-length tests and discretised input/output values. We present test cases for the diesel emissions scandal and execute these tests with a real car on a chassis dynamometer.

2 Software Doping on Reactive Programs

Embedded software is reactive, it reacts to inputs received from sensors by producing outputs that are meant to control the device functionality. We consider a reactive program as a function $P : \mathsf{In}^\omega \to 2^{(\mathsf{Out}^\omega)}$ on infinite sequences of inputs so that the program reacts to the k-th input in the input sequence by producing non-deterministically the k-th output in each respective output sequence. Thus, the program can be seen, for instance, as a (non-deterministic) Mealy or Moore machine. Moreover, we consider an equivalence relation $\approx \subseteq \mathsf{In}^\omega \times \mathsf{In}^\omega$ that equates sequences of inputs. To illustrate this, think of the program embedded in a printer. Here $\approx$ would for instance equate input sequences that agree with respect to submitting the same documents regardless of the cartridge brand, the level of the toner (as long as there is sufficient), etc. We furthermore consider the set $\mathsf{StdIn} \subseteq \mathsf{In}^\omega$ of inputs of interest or *standard inputs*. In the previous example, StdIn contains all the input sequences with compatible cartridges and printable documents. The definitions given below are simple adaptations of those given in [10] (but where parameters are instead treated as parts of the inputs).

Definition 1. *A reactive program P is* clean *if for all inputs $\mathsf{i}, \mathsf{i}' \in \mathsf{StdIn}$ such that $\mathsf{i} \approx \mathsf{i}'$, $P(\mathsf{i}) = P(\mathsf{i}')$. Otherwise it is* doped.

This definition states that a program is *clean* if its execution exhibits the same visible sequence of output when supplied with two equivalent inputs, provided such inputs comply with the given standard StdIn. Notice that the behaviour outside StdIn is deemed immediately clean since it is of no interest.

In the context of the printer example, a program that would fail to print a document when provided with an ink cartridge from a third-party manufacturer, but would otherwise succeed to print would be considered doped, since this difference in output behaviour is captured by the above definition. For this, the inputs (being pairs of document and printer cartridge) must be considered equivalent (not identical), which comes down to ink cartridges being compatible.

However, the above definition is not very helpful for cases that need to preserve certain intended behaviour *outside* of the standard inputs StdIn. This is clearly the case in the diesel emissions scandal where the standard inputs are given precisely by the emissions test, but the behaviour observed there is assumed to generalise beyond the singularity of this test setup. It is meant to ensure that the amount of NO_2 and NO (abbreviated as NO_x) in the car exhaust gas does not deviate considerably *in general*, and comes with a legal prohibition of defeat mechanisms that simply turn off the cleaning mechanism. This legal framework is obviously a bit short sighted, since it can be circumvented by mechanisms that alter the behaviour gradually in a continuous manner, but in effect drastically. In a nutshell, one expects that if the input values observed by the electronic control unit (ECU) of a diesel vehicle deviate within "reasonable distance" from the *standard* input values provided during the lab emission test, the amount of NO_x found in the exhaust gas is still within the regulated threshold, or at least it does not exceed it more than a "reasonable amount".

This motivates the need to introduce the notion of distances on inputs and outputs. More precisely, we consider distances on finite traces: $d_{\mathsf{In}} : (\mathsf{In}^* \times \mathsf{In}^*) \to \mathbb{R}_{\geq 0}$ and $d_{\mathsf{Out}} : (\mathsf{Out}^* \times \mathsf{Out}^*) \to \mathbb{R}_{\geq 0}$. Such distances are required to be pseudometrics. (d is a pseudometric if $d(x,x) = 0$, $d(x,y) = d(y,x)$ and $d(x,y) \leq d(x,z) + d(z,y)$ for all x, y, and z.) With this, D'Argenio et al. [10] provide a definition of robust cleanness that considers two parameters: parameter κ_{i} refers to the acceptable distance an input may deviate from the norm to be still considered, and parameter κ_{o} that tells how far apart outputs are allowed to be in case their respective inputs are within κ_{i} distance (Definition 2 spells out the Hausdorff distance used in [10]).

Definition 2. *Let $\sigma[..k]$ denote the k-th prefix of the sequence σ. A reactive program P is* robustly clean *if for all input sequences $\mathsf{i}, \mathsf{i}' \in \mathsf{In}^\omega$ with $\mathsf{i} \in \mathsf{StdIn}$, it holds for arbitrary $k \geq 0$ that whenever $d_{\mathsf{In}}(\mathsf{i}[..j], \mathsf{i}'[..j]) \leq \kappa_{\mathsf{i}}$ for all $j \leq k$, then*

1. *for all $\mathsf{o} \in P(\mathsf{i})$ there exists $\mathsf{o}' \in P(\mathsf{i}')$ such that $d_{\mathsf{Out}}(\mathsf{o}[..k], \mathsf{o}'[..k]) \leq \kappa_{\mathsf{o}}$, and*
2. *for all $\mathsf{o}' \in P(\mathsf{i}')$ there exists $\mathsf{o} \in P(\mathsf{i})$ such that $d_{\mathsf{Out}}(\mathsf{o}[..k], \mathsf{o}'[..k]) \leq \kappa_{\mathsf{o}}$.*

Notice that this is what we actually need for the non-deterministic case: each possible output generated along one of the executions of the program should be matched within "reasonable distance" by some output generated by the other execution of the program. Also notice that i' does not need to satisfy StdIn, but it will be considered as long as it is within κ_{i} distance of any input satisfying StdIn. In such a case, outputs generated by $P(\mathsf{i}')$ will be requested to be within κ_{o} distance of some output generated by the respective execution induced by a standard input.

We remark that Definition 2 entails the existence of a *contract* which defines the set of standard inputs StdIn, the tolerance parameters κ_{i} and κ_{o} as well as the distances d_{In} and d_{Out}. In the context of diesel engines, one might imagine that the values to be considered, especially the tolerance parameters κ_{i} and κ_{o} for a particular car model are made publicly available (or are even advertised by the car manufacturer), so as to enable potential customers to discriminate between different car models according to the robustness they reach in being clean. It is also imaginable that the tolerances and distances are fixed by the legal authorities as part of environmental regulations.

3 Robustly Clean Labelled Transition Systems

This section develops the framework needed for an effective theory of black-box doping tests based on the above concepts. In this, the standard behaviour (e.g. as defined by the emission tests) and the robust cleanness definitions together will induce a set of reference behaviours that then serve as a model in a model-based conformance testing approach. To set the stage for this, we recall the definitions of labelled transition systems (LTS) and input-output transitions systems (IOTS) together with Tretmans' notion on model-based conformance testing [25]. We then recast the characterisation of robust cleanness (Definition 2) in terms of LTS.

Definition 3. *A* labelled transition system (LTS) *with inputs and outputs is a tuple* $\langle Q, \mathsf{In}, \mathsf{Out}, \rightarrow, q_0 \rangle$ *where (i)* Q *is a (possibly uncountable) non-empty set of states; (ii)* $L = \mathsf{In} \uplus \mathsf{Out}$ *is a (possibly uncountable) set of labels; (iii)* $\rightarrow \subseteq Q \times L \times Q$ *is the transition relation; (iv)* $q_0 \in Q$ *is the initial state. We say that a LTS is an* input-output transition system (IOTS) *if it is input-enabled in any state, i.e., for all* $s \in Q$ *and* $a \in \mathsf{In}$ *there is some* $s' \in Q$ *such that* $s \xrightarrow{a} s'$.

For ease of presentation, we do not consider internal transitions. The following definitions will be used throughout the paper. A *finite path* p in an LTS $\mathcal{L}$ is a sequence $s_1 a_1 s_2 a_2 \dots a_{n-1} s_n$ with $s_i \xrightarrow{a_i} s_{i+1}$ for all $1 \leq i < n$. Similarly, an *infinite path* p in $\mathcal{L}$ is a sequence $s_1 a_1 s_2 a_2 \dots$ with $s_i \xrightarrow{a_i} s_{i+1}$ for all $i \in \mathbb{N}$. Let $\mathsf{paths}_*(\mathcal{L})$ and $\mathsf{paths}_\omega(\mathcal{L})$ be the sets of all finite and infinite paths of $\mathcal{L}$ beginning in the initial states, respectively. The sequence $a_1 a_2 \cdots a_n$ is a *finite trace* of $\mathcal{L}$ if there is a finite path $s_1 a_1 s_2 a_2 \dots a_n s_{n+1} \in \mathsf{paths}_*(\mathcal{L})$, and $a_1 a_2 \cdots$ is an *infinite trace* if there is an infinite path $s_1 a_1 s_2 a_2 \dots \in \mathsf{paths}_\omega(\mathcal{L})$. If p is a path, we let $\mathsf{trace}(p)$ denote the trace defined by p. Let $\mathsf{traces}_*(\mathcal{L})$ and $\mathsf{traces}_\omega(\mathcal{L})$ be the sets of all finite and infinite traces of $\mathcal{L}$, respectively. We will use $\mathcal{L}_1 \subseteq \mathcal{L}_2$ to denote that $\mathsf{traces}_\omega(\mathcal{L}_1) \subseteq \mathsf{traces}_\omega(\mathcal{L}_2)$.

Model-Based Conformance Tests. In the following we recall the basic notions of *input-output conformance* (**ioco**) testing [25–27], and refer to the mentioned literature for more details. In this setting, it is assumed that the implemented system under test (IUT) $\mathcal{I}$ can be modelled as an IOTS while the specification of the required behaviour is given in terms of a LTS *Spec*. The idea of whether the IUT $\mathcal{I}$ *conforms to* the specification *Spec* is formalized by means of the **ioco** relation which we define in the following.

We first need to identify the *quiescent* (or *suspended*) states. A state is quiescent whenever it cannot proceed autonomously, i.e., it cannot produce an output. We will make each such state identifiable by adding a quiescence transition to it, in the form of a loop with the distinct label δ.

Definition 4. *Let* $\mathcal{L} = \langle Q, \mathsf{In}, \mathsf{Out}, \rightarrow, q_0 \rangle$ *be an LTS. The* quiescence closure *(or* δ-closure*) of* $\mathcal{L}$ *is the LTS* $\mathcal{L}_\delta := \langle Q, \mathsf{In}, \mathsf{Out} \cup \{\delta\}, \rightarrow_\delta, q_0 \rangle$ *with* $\rightarrow_\delta := \rightarrow \cup \{s \xrightarrow{\delta}_\delta s \mid \forall o \in \mathsf{Out}, t \in Q : s \not\xrightarrow{o} t\}$. *Using this we define the* suspension traces *of* $\mathcal{L}$ *by* $\mathsf{traces}_*(\mathcal{L}_\delta)$.

Let $\mathcal{L}$ be an LTS with initial state q_0 and $\sigma = a_1 a_2 \dots a_n \in \mathsf{traces}_*(\mathcal{L})$. We define $\mathcal{L}$ after σ as the set $\{q_n \mid q_0 a_1 q_1 a_2 \dots a_n q_n \in \mathsf{paths}_*(\mathcal{L})\}$. For a state q, let $\mathsf{out}(q) = \{o \in \mathsf{Out} \cup \{\delta\} \mid \exists q' : q \xrightarrow{o} q'\}$ and for a set of states $Q' \subseteq Q$, let $\mathsf{out}(Q') = \bigcup_{q \in Q'} \mathsf{out}(q)$.

The idea behind the **ioco** relation is that any output produced by the IUT must have been foreseen by its specification, and moreover, any input in the IUT not foreseen in the specification may introduce new functionality. **ioco** captures this by harvesting concepts from refusal testing. As a result, $\mathcal{I}$ **ioco** *Spec* is defined to hold whenever $\mathsf{out}(\mathcal{I}_\delta \ \mathsf{after} \ \sigma) \subseteq \mathsf{out}(\mathit{Spec}_\delta \ \mathsf{after} \ \sigma)$ for all $\sigma \in \mathsf{traces}_*(\mathit{Spec}_\delta)$.

The base principle of *conformance testing* now is to assess by means of testing whether the IUT conforms to its specification w.r.t. **ioco**. An algorithm to derive a corresponding test suite T_{Spec} is available [26,27], so that for any IUT $\mathcal{I}$, $\mathcal{I}$ **ioco** *Spec* iff $\mathcal{I}$ passes all tests in T_{Spec}.

It is important to remark that the specification in the setting considered here is missing. Instead, we need to construct the specification from the standard inputs and the respective observed outputs, together with the distances and the thresholds given by the contract. Furthermore, this needs to respect the $\forall - \exists$ interaction required by the cleanness property (Definition 2).

Software Doping on LTS. To capture the notion of software doping in the context of LTS, we provide two projections of a trace, projecting to a sequence of the appearing inputs, respectively outputs. To do this, we extend the set of labels by adding the input $-_i$, that indicates that in the respective step some output (or quiescence) was produced (but masking the precise output), and the output $-_o$ that indicates that in this step some (masked) input was given.

The *projection on inputs* $\downarrow_i : L^\omega \to (\mathsf{In} \cup \{-_i\})^\omega$ and the *projection on outputs* $\downarrow_o : L^\omega \to (\mathsf{Out} \cup \{-_o\})^\omega$ are defined for all traces σ and $k \in \mathbb{N}$ as follows: $\sigma\downarrow_i[k] :=$ **if** $\sigma[k] \in \mathsf{In}$ **then** $\sigma[k]$ **else** $-_i$ and $\sigma\downarrow_o[k] :=$ **if** $\sigma[k] \in \mathsf{Out}$ **then** $\sigma[k]$ **else** $-_o$. They are lifted to sets of traces in the usual elementwise way.

Definition 5. *A LTS $\mathcal{S}$ is a* standard *for a LTS $\mathcal{L}$, if* $\mathsf{traces}_\omega(\mathcal{S}_\delta) \subseteq \mathsf{traces}_\omega(\mathcal{L}_\delta)$.

The above definition provides an interpretation of the notion of StdIn for a given program P modelled in terms of LTS $\mathcal{L}$. This interpretation relaxes the original definition of StdIn, because it requires to fix only a subset of the behaviour that $\mathcal{L}$ exhibits when executed with standard inputs. This corresponds to a testing context, in which recordings of the system executing standard inputs are the baseline for testing. StdIn can then be considered as implicitly determined as the input sequences $\mathsf{traces}_\omega(\mathcal{S})\downarrow_i$ occurring in $\mathcal{S}$. If instead $\mathcal{L}$ and $\mathsf{StdIn} \subseteq (\mathsf{In} \cup -_i)^\omega$ are given, we denote by $\mathcal{S}^{(\mathcal{L},\mathsf{StdIn})}$ a standard LTS which is maximal w.r.t. StdIn and $\mathcal{L}$, i.e., for all $\sigma \in \mathsf{traces}_\omega({\mathcal{S}_\delta}^{(\mathcal{L},\mathsf{StdIn})})$ iff $\sigma\downarrow_i \in \mathsf{StdIn}$ and $\sigma \in \mathsf{traces}_\omega(\mathcal{L}_\delta)$.

In this new setting, we assume that the distance functions d_{In} and d_{Out} run on traces containing labels $-_i$ and $-_o$, i.e. they are pseudometrics in $(\mathsf{In} \cup \{-_i\})^* \times (\mathsf{In} \cup \{-_i\})^* \to \mathbb{R}_{\geq 0}$ and $(\mathsf{Out} \cup \{-_o\})^* \times (\mathsf{Out} \cup \{-_o\})^* \to \mathbb{R}_{\geq 0}$, respectively. We will denote a contract explicitly by a 5-tuple $\mathcal{C} = \langle \mathcal{S}, d_{\mathsf{In}}, d_{\mathsf{Out}}, \kappa_{\mathsf{i}}, \kappa_{\mathsf{o}} \rangle$, which contains a LTS $\mathcal{S}$ representing some standard behaviour, the distances and thresholds (the domains In and Out are captured implicitly as the domains of d_{In}, respectively d_{Out}). With this, robust cleanness can be restated in terms of LTS as follows.

Definition 6. *Let $\mathcal{L}$ be an IOTS and $\mathcal{C} = \langle \mathcal{S}, d_{\mathsf{In}}, d_{\mathsf{Out}}, \kappa_{\mathsf{i}}, \kappa_{\mathsf{o}} \rangle$ a contract so that $\mathcal{S}$ is standard for $\mathcal{L}$. This $\mathcal{L}$ is* robustly clean w.r.t. *$\mathcal{C}$ if for all $\sigma \in \mathsf{traces}_\omega(\mathcal{S}_\delta)$ and $\sigma' \in \mathsf{traces}_\omega(\mathcal{L}_\delta)$ it holds for arbitrary $k \geq 0$ that whenever $d_{\mathsf{In}}(\sigma[..j]\downarrow_i, \sigma'[..j]\downarrow_i) \leq \kappa_{\mathsf{i}}$ for all $j \leq k$ then*

1. *there exists $\sigma'' \in \mathsf{traces}_\omega(\mathcal{L}_\delta)$ s.t. $\sigma'\downarrow_i = \sigma''\downarrow_i$ and $d_{\mathsf{Out}_\delta}(\sigma[..k]\downarrow_o, \sigma''[..k]\downarrow_o) \leq \kappa_{\mathsf{o}}$,*
2. *there exists $\sigma'' \in \mathsf{traces}_\omega(\mathcal{S}_\delta)$ s.t. $\sigma\downarrow_i = \sigma''\downarrow_i$ and $d_{\mathsf{Out}_\delta}(\sigma'[..k]\downarrow_o, \sigma''[..k]\downarrow_o) \leq \kappa_{\mathsf{o}}$.*

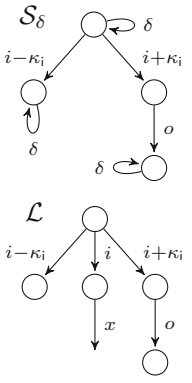
In the spirit of model-based testing with **ioco**, Definition 6 takes specific care of quiescence in a system. In order to properly integrate quiescence into the context of robust cleanness it must be considered as a unique output. As a consequence, in the presence of a contract $\mathcal{C} = \langle \mathcal{S}, d_{\mathsf{In}}, d_{\mathsf{Out}}, \kappa_{\mathsf{i}}, \kappa_{\mathsf{o}} \rangle$, we use – instead of $\mathcal{S}$, Out and d_{Out} – the quiescence closure $\mathcal{S}_\delta$ of $\mathcal{S}$, $\mathsf{Out}_\delta = \mathsf{Out} \cup \{\delta\}$ and an extended output distance defined as $d_{\mathsf{Out}_\delta}(\sigma_1, \sigma_2) \coloneqq d_{\mathsf{Out}}(\sigma_{1\setminus\delta}, \sigma_{2\setminus\delta})$ if $\sigma_1[i] = \delta \Leftrightarrow \sigma_2[i] = \delta$ for all i, and $d_{\mathsf{Out}_\delta}(\sigma_1, \sigma_2) \coloneqq \infty$ otherwise, where $\sigma_{\setminus\delta}$ is the same as σ with all δ removed.

For the maximal standard LTS $\mathcal{S}^{(\mathcal{L},\mathsf{StdIn})}$, Definition 6 echoes the semantics of the HyperLTL interpretation appearing in Proposition 19 of [10] restricted to programs with no parameters. Thus, the proof showing that Definition 6 is the correct interpretation of Definition 2 in terms of LTS, can be obtained in a way similar to that of Proposition 19 in [10].

In the sequel, we will at some places need to refer to Definition 6 only considering the second condition (but not the first one). We denote this as Definition 6.2.

4 Reference Implementation for Contracts

As mentioned before, doping tests need to be based on a contract $\mathcal{C}$, which we assume given. $\mathcal{C}$ specifies the domains In, Out, a standard LTS $\mathcal{S}$, the distances d_{In} and d_{Out} and the bounds κ_{i} and κ_{o}. We intuitively expect the contract to be satisfiable in the sense that it never enforces a single input sequence of the implementation to keep outputs close enough to two different executions of the specification while their outputs stretch too far apart. We show such a problematic case in the following example.



Example 2. On the right a quiescence-closed standard LTS $\mathcal{S}_\delta$ for an implementation $\mathcal{L}$ (shown below) is depicted. For simplicity some input transitions are omitted. Assume $\mathsf{Out} = \{o\}$ and $\mathsf{In} = \{i, i-\kappa_{\mathsf{i}}, i+\kappa_{\mathsf{i}}\}$. Consider the transition labelled x of $\mathcal{L}$. This must be one of either o or δ, but we will see that either choice leads to a contradiction w.r.t. the output distances induced. The input projection of the middle path in $\mathcal{L}$ is $i\ {-_i}$ and the input distance to $(i-\kappa_{\mathsf{i}})\ {-_i}$ and $(i+\kappa_{\mathsf{i}})\ {-_i}$ is exactly κ_{i}, so both branches $(i+\kappa_{\mathsf{i}})\ o$ and $(i-\kappa_{\mathsf{i}})\ \delta$ of $\mathcal{S}_\delta$ must be considered to determine x. For $x = o$, the output distance of ${-_o}\ x$ to ${-_o}\ o$ in the right branch of $\mathcal{S}_\delta$ is 0, i.e. less than κ_{o}. However, $d_{\mathsf{Out}_\delta}({-_o}\ \delta, {-_o}\ o) = \infty > \kappa_{\mathsf{o}}$. Thus the output distance to the left branch of $\mathcal{S}_\delta$ is too high if picking o. Instead picking $x = \delta$ does not work either, for the symmetric reasons, the problem switches sides. Thus, neither picking o nor δ for x satisfies robust cleanness here. Indeed, no implementation satisfying robust cleanness exists for the given contract.

We would expect that a correct implementation fully entails the standard behaviour. So, to satisfy a contract, the standard behaviour itself must be robustly clean. This and the need for satisfiability of particular inputs lead to Definition 7.

Definition 7 (Satisfiable Contract). *Let* $\mathcal{C} = \langle \mathcal{S}, d_{\mathsf{In}}, d_{\mathsf{Out}}, \kappa_{\mathsf{i}}, \kappa_{\mathsf{o}} \rangle$ *be a contract. Let* input $\sigma_i \in (\mathsf{In} \cup \{-_i\})^\omega$ *be the input projection of some trace.* σ_i *is* satisfiable *for* $\mathcal{C}$ *if and only if for every standard trace* $\sigma_S \in \mathsf{traces}_\omega(\mathcal{S}_\delta)$ *and* $k > 0$ *such that for all* $j \leq k$ $d_{\mathsf{In}}(\sigma_i[..j], \sigma_S[..j]{\downarrow_i}) \leq \kappa_{\mathsf{i}}$ *there is some implementation* $\mathcal{L}$ *that satisfies Definition 6.2 w.r.t.* $\mathcal{C}$ *and has some trace* $\sigma \in \mathsf{traces}_\omega(\mathcal{L}_\delta)$ *with* $\sigma{\downarrow_i} = \sigma_i$ *and* $d_{\mathsf{Out}_\delta}(\sigma[..k]{\downarrow_o}, \sigma_S[..k]{\downarrow_o}) \leq \kappa_{\mathsf{o}}$.
$\mathcal{C}$ *is* satisfiable *if and only if all inputs* $\sigma_i \in (\mathsf{In} \cup \{-_i\})^\omega$ *are satisfiable for* $\mathcal{C}$ *and if* $\mathcal{S}$ *is robustly clean w.r.t.* $\mathcal{C}$*. A contract that is not satisfiable is called* unsatisfiable.

Given a satisfiable contract it is always possible to construct an implementation that is robustly clean w.r.t. to this contract. Furthermore, for every contract there is exactly one implementation (modulo trace equivalence) that contains all possible outputs that satisfy robust cleanness. Such an implementation is called the *largest implementation*.

Definition 8 (Largest Implementation). *Let* $\mathcal{C}$ *be a contract and* $\mathcal{L}$ *an implementation that is robustly clean w.r.t.* $\mathcal{C}$*.* $\mathcal{L}$ *is the* largest implementation within $\mathcal{C}$ *if and only if for every* $\mathcal{L}'$ *that is robustly clean w.r.t.* $\mathcal{C}$ *it holds that* $\mathsf{traces}_\omega(\mathcal{L}'_\delta) \subseteq \mathsf{traces}_\omega(\mathcal{L}_\delta)$.

In the following, we will focus on the fragment of satisfiable contracts with standard behaviour defined by finite LTS. For unsatisfiable contracts, testing is not necessary, because every implementation is not robustly clean w.r.t. to $\mathcal{C}$. Finiteness of $\mathcal{S}$ will be necessary to make testing feasible in practice. For simplicity we will further assume *past-forgetful* output distance functions. That is, $d_{\mathsf{Out}}(\sigma_1, \sigma_2) = d_{\mathsf{Out}}(\sigma'_1, \sigma'_2)$ whenever $\mathsf{last}(\sigma_1) = \mathsf{last}(\sigma'_1)$ and $\mathsf{last}(\sigma_2) = \mathsf{last}(\sigma'_2)$ (where $\mathsf{last}(a_1\ a_2 \ldots a_n) = a_n$.) Thus, we simply assume that $d_{\mathsf{Out}} : (\mathsf{Out} \cup \{-_o\} \times \mathsf{Out} \cup \{-_o\}) \to \mathbb{R}_{\geq 0}$, i.e., the output distances are determined by the last output only. We remark that $d_{\mathsf{Out}_\delta}(\delta, o) = \infty$ for all $o \neq \delta$.

We will now show how to construct the largest implementation for any contract (of the fragment we consider), which we name *reference implementation* $\mathcal{R}$. It is derived from $\mathcal{S}_\delta$ by adding inputs and outputs in such a way that whenever the input sequence leading to a particular state is within κ_{i} distance of an input sequence σ_i of $\mathcal{S}_\delta$, then the outputs possible in such a state should be at most κ_{o} distant from those outputs possible in the unique state on $\mathcal{S}_\delta$ reached through σ_i. This ensures that $\mathcal{R}$ will satisfy condition (2) in Definition 6.

Reference Implementation. To construct the reference implementation $\mathcal{R}$ we decide to model the quiescence transitions explicitly instead of using the quiescence closure. We preserve the property, that in each state of the LTS it is possible to do an output or a quiescence transition. The construction of $\mathcal{R}$ proceeds by adding all transitions that satisfy the second condition of Definition 6.

Definition 9. *Let* $\mathcal{C} = \langle \mathcal{S}, d_{\mathsf{In}}, d_{\mathsf{Out}}, \kappa_{\mathsf{i}}, \kappa_{\mathsf{o}} \rangle$ *be a contract. The* reference implementation $\mathcal{R}$ *for* $\mathcal{C}$ *is the LTS* $\langle (\mathsf{In} \cup \mathsf{Out})^*, \mathsf{In}, \mathsf{Out}, \to_{\mathcal{R}}, \epsilon \rangle$ *where* $\to_{\mathcal{R}}$ *is defined by*

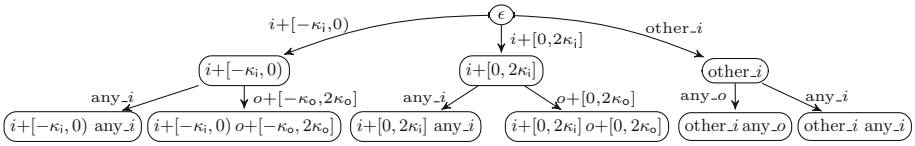
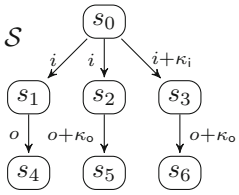


Fig. 1. The reference implementation $\mathcal{R}$ of $\mathcal{S}$ in Example 3.

$$\frac{\begin{array}{l}\forall \sigma_i \in \mathsf{traces}_\omega(\mathcal{S}_\delta)\downarrow_i : \\ \quad (\forall j \leq |\sigma| + 1 : d_{\mathsf{In}}((\sigma \cdot a)\downarrow_i[..j], \sigma_i[..j]) \leq \kappa_{\mathsf{i}}) \\ \qquad \Rightarrow \exists \sigma_S \in \mathsf{traces}_\omega(\mathcal{S}_\delta) : \sigma_S\downarrow_i = \sigma_i \wedge d_{\mathsf{Out}_\delta}(a\downarrow_o, \sigma_S[|\sigma| + 1]\downarrow_o) \leq \kappa_{\mathsf{o}}\end{array}}{\sigma \xrightarrow{a}_{\mathcal{R}} \sigma \cdot a}$$

Notably, $\mathcal{R}$ is deterministic, since only transitions of the form $\sigma \xrightarrow{a}_{\mathcal{R}} \sigma \cdot a$ are added. As a consequence of this determinism, outputs and quiescence may coexist as options in a state, i.e. they are not mutually exclusive.

Example 3. Fig. 1 gives a schematic representation of the reference implementation $\mathcal{R}$ for the LTS $\mathcal{S}$ on the right. Input (output) actions are denoted with letter i (o, respectively), quiescence transitions are omitted. We use the absolute difference of the values, so that $d_{\mathsf{In}}(i, i') \coloneqq |i - i'|$ and $d_{\mathsf{Out}}(o, o') \coloneqq |o - o'|$. For this example, the quiescence closure $\mathcal{S}_\delta$ looks like $\mathcal{S}$ but with δ-loops in states s_0, s_4, s_5, and s_6. Label $r+[a, b]$ should be interpreted as any value $r' \in [a + r, b + r]$ and similarly $r+[a, b)$ and $r+(a, b]$, appropriately considering closed and open boundaries; "other_i" represents any other input not explicitly considered leaving the same state; and "any_i" and "any_o" represent any possible input and output (including δ), respectively. In any case $-_i$ and $-_o$ are not considered since they are not part of the alphabet of the LTS. Also, we note that any possible sequence of inputs becomes enabled in the last states (omitted in the picture).



Robust Cleanness of Reference Implementation. In the following, the aim is to show that $\mathcal{R}$ is robustly clean. By construction, each state in $\mathcal{R}$ equals the trace that leads to that state. In other words, $\mathsf{last}(p) = \mathsf{trace}(p)$ for any $p \in \mathsf{paths}_*(\mathcal{R})$ can be shown by induction. As a consequence, a path in $\mathcal{R}$ can be completely identified by the trace it defines. The following lemma states that $\mathcal{R}$ preserves all traces of the standard $\mathcal{S}_\delta$ it is constructed from. This can be proven by using that $\mathcal{S}_\delta$ is robustly clean w.r.t. the (satisfiable) contract $\mathcal{C}$ (see Definition 7).

Lemma 1. *Let $\mathcal{R}$ be the reference implementation for $\mathcal{C} = \langle \mathcal{S}, d_{\mathsf{In}}, d_{\mathsf{Out}}, \kappa_{\mathsf{i}}, \kappa_{\mathsf{o}} \rangle$. Then $\mathcal{S}$ is standard for $\mathcal{R}$.*

The following theorem states that the reference implementation $\mathcal{R}$ is robustly clean w.r.t. the contract it was constructed from.

Theorem 1. *Let $\mathcal{R}$ be the reference implementation for some contract $\mathcal{C}$. Then $\mathcal{R}$ is robustly clean w.r.t. $\mathcal{C}$.*

Furthermore, it is not difficult to show that $\mathcal{R}$ is indeed the largest implementation within the contract it was constructed from.

Theorem 2. *Let $\mathcal{R}$ be the reference implementation for some contract $\mathcal{C}$. Then $\mathcal{R}$ is the largest implementation within $\mathcal{C}$.*

5 Model-Based Doping Tests

Following the conceptual ideas behind **ioco**, we need to construct a specification that is compatible with our notion of robust cleanness in such a way that a test suite can be derived. Intuitively, such a specification must be able to foresee every behaviour of the system that is allowed by the contract. We will take the reference implementation from the previous section as this specification. Indeed we claim that $\mathcal{R}$ is constructed in such a way that whenever an IUT $\mathcal{I}$ is robustly clean, $\mathcal{I}$ **ioco** $\mathcal{R}$ holds. The latter translates to

$$\forall \sigma \in \mathsf{traces}_*(\mathcal{R}_\delta) : \mathsf{out}(\mathcal{I}_\delta \text{ after } \sigma) \subseteq \mathsf{out}(\mathcal{R}_\delta \text{ after } \sigma). \tag{1}$$

Theorem 3. *Let $\mathcal{C}$ be a contract with standard $\mathcal{S}$ and let IOTS $\mathcal{I}$ be robustly clean w.r.t. $\mathcal{C}$. If $\mathcal{R}$ is the reference implementation for $\mathcal{C}$, then $\mathcal{I}$* **ioco** *$\mathcal{R}$.*

The key observations to prove this theorem are: (i) the reference implementation is the largest implementation within the contract, i.e. if the IUT is robustly clean, then all its traces are covered by $\mathcal{R}$, and (ii) by construction of $\mathcal{R}$ and satisfiability of $\mathcal{C}$, the suspension traces of $\mathcal{R}$ are exactly its finite traces.

Test Algorithm. An important element of the model-based testing theory is a non-deterministic algorithm to generate test cases. It is, however, not guaranteed that this algorithm, even if existing, is implementable, a problem which we will tackle in this section. A set of test cases is called a *test suite*. It is shown elsewhere [27], that there is an algorithm that can produce a (possibly infinitely large) test suite T, for which a system $\mathcal{I}$ passes T if $\mathcal{I}$ is correct w.r.t. **ioco** and, conversely, $\mathcal{I}$ is correct w.r.t. **ioco** if $\mathcal{I}$ passes T. The former property is called *soundness* and the latter is called *exhaustiveness*. Algorithm 1 shows a tail-recursive algorithm to test for robust cleanness. This DT algorithm takes as an argument the history h of the test currently running. Every doping test is initialized by $\mathsf{DT}(\epsilon)$. Several runs of the algorithm constitute a test suite. Each test can either **pass** or **fail**, which is reported by the output of the algorithm. In each call DT picks one of three choices: (i) it either terminates the test by returning **pass** (line 3), (ii) if there is no pending output that has to be read from the system under test, the algorithm may pick a new input and pass it to the system (lines 5–6), or (iii) DT reads and checks the next output (or quiescence) that the system produces (lines 9–10). Quiescence can be recognized by using a timeout mechanism that returns δ if no output has been received in a given amount of time. In the original algorithm, the case and the next input are determined non-deterministically. Our algorithm is parameterized by Ω_{case}

Algorithm 1. Doping Test (DT)

Input: history $h \in (\mathsf{In} \cup \mathsf{Out} \cup \{\delta\})^*$
Output: **pass** or **fail**

```
1  c ← Ω_case(h)    /* Pick from one of three cases */
2  if c = 1 then
3     return pass    /* Finish test generation */
4  else if c = 2 and no output from I is available then
5     i ← Ω_In(h)    /* Pick next input */
6     i ↠ I    /* Forward input to IUT */
7     return DT(h · i)    /* Continue with next step */
8  else if c = 3 or output from I is available then
9     o ↞ I    /* Receive output from IUT */
10    if o ∈ acc(h) then
11       return DT(h · o)  /* If o is foreseen by oracle continue with next step */
12    else
13       return fail    /* Otherwise, report test failure */
14    end if
15 end if
```

and Ω_{In}, which can be instantiated by either non-determinism or some optimized test-case selection. Until further notice we assume non-deterministic selection. An output or quiescence that has been produced by the IUT is checked by means of an oracle acc (line 10). The oracle reflects the reference implementation $\mathcal{R}$, that is used as the specification for the **ioco** relation and is defined in Eq. (2).

$$\begin{aligned} \mathsf{acc}(h) \coloneqq \{o \in \mathsf{Out}_\delta \mid & \\ \forall \sigma_i \in \mathsf{traces}_\omega(\mathcal{S}_\delta)\downarrow_i : (\forall j \leq |h|+1 : d_{\mathsf{In}}(\sigma_i[..j]\downarrow_i, (h \cdot o)[..j]\downarrow_i) \leq \kappa_{\mathsf{i}}\}) & \\ \Rightarrow \exists \sigma \in \mathsf{traces}_\omega(\mathcal{S}_\delta) : \sigma\downarrow_i = \sigma_i\downarrow_i \wedge d_{\mathsf{Out}_\delta}(o, \sigma[|h|+1]\downarrow_o) \leq \kappa_{\mathsf{o}} & \end{aligned} \tag{2}$$

Given a finite execution, acc returns the set of acceptable outputs (after such an execution) which corresponds exactly to the set of outputs in $\mathcal{R}$ (after such an execution). Thus $\mathsf{acc}(h)$ is precisely the set of outputs that satisfies the premise in the definition of $\mathcal{R}$ after the trace h, as stipulated in Definition 9.

We refer to acc as an oracle, because it cannot be computed in general due to the infinite traces of $\mathcal{S}_\delta$ in the definition. However, we get the following theorem stating that the algorithm is sound and exhaustive with respect to **ioco** (and we present a computable algorithm in the next section). The theorem follows from the soundness and exhaustiveness of the original test generation algorithm for model-based testing and Definition 9.

Theorem 4. *Let $\mathcal{C}$ be a contract with standard $\mathcal{S}$. Let $\mathcal{I}$ be an implementation with $\mathcal{S}_\delta \subseteq \mathcal{I}_\delta$ and let $\mathcal{R}$ be the largest implementation within $\mathcal{C}$. Then, $\mathcal{I}$* **ioco** *$\mathcal{R}$ if and only if for every test execution $t = \mathsf{DT}(\epsilon)$ it holds that $\mathcal{I}$* **passes** *t.*

Corollary 1. *Let $\mathcal{C}$ be a contract with standard $\mathcal{S}$. Let $\mathcal{I}$ be an implementation with $\mathcal{S}_\delta \subseteq \mathcal{I}_\delta$. If $\mathcal{I}$ is robustly clean w.r.t. $\mathcal{C}$, then for every test execution $t = \mathsf{DT}(\epsilon)$ it holds that $\mathcal{I}$* **passes** *t.*

This corollary is derived from Theorem 3 and the satisfiability of $\mathcal{C}$. It is worth noting that in this corollary we do not get that $\mathcal{I}$ is robustly clean if $\mathcal{I}$ always passes DT. This is due the intricacies of genuine hyperproperties. By testing, we will never be able to verify the first condition of Definition 6, because this needs a simultaneous view on all possible execution traces of $\mathcal{I}$. During testing, however, we always can observe only one trace.

Finite Doping Tests. As mentioned before, the execution of DT is not possible, because the oracle acc is not computable. There is, however, a computable version acc_b of acc for executions up to some test length b for bounded and discretised In and Out. Even for infinite executions, b can be seen as a limit of interest and testing is still sound. acc_b is shown in Eq. (3). The only variation w.r.t. acc lies in the use of the set $\mathsf{traces}_b(\mathcal{S}_\delta)$, instead of $\mathsf{traces}_\omega(\mathcal{S}_\delta)$, so as to return all traces of $\mathcal{S}_\delta$ whose length is exactly b. Since $\mathcal{S}_\delta$ is finite, function acc_b can be implemented.

$$\begin{aligned}\mathsf{acc}_b(h) \coloneqq \{o \in \mathsf{Out}_\delta \mid \\ \forall \sigma_i \in \mathsf{traces}_b(\mathcal{S}_\delta)\downarrow_i : (\forall j \leq |h|{+}1 : d_{\mathsf{In}}(\sigma_i[..j]\downarrow_i, (h \cdot o)[..j]\downarrow_i) \leq \kappa_{\mathsf{i}}) \\ \Rightarrow \exists \sigma \in \mathsf{traces}_b(\mathcal{S}_\delta) : \ \sigma\downarrow_i = \sigma_i\downarrow_i \wedge d_{\mathsf{Out}_\delta}(o, \sigma[|h|{+}1]\downarrow_o) \leq \kappa_{\mathsf{o}}\}\end{aligned} \tag{3}$$

Now we get a new algorithm DT_b by replacing acc by acc_b in DT and by forcing case 1 when and only when $|h| = b$. We get a similar soundness theorem for DT_b as in Corollary 1.

Theorem 5. *Let $\mathcal{C}$ be a contract with standard $\mathcal{S}$. Let $\mathcal{I}$ be an implementation with $\mathcal{S}_\delta \subseteq \mathcal{I}_\delta$. If $\mathcal{I}$ is robustly clean w.r.t. $\mathcal{C}$, then for every boundary b and every test execution $t = \mathsf{DT}_b(\epsilon)$ it holds that $\mathcal{I}$* **passes** *t.*

Since $\mathcal{I}$ **passes** $\mathsf{DT}_b(\epsilon)$ implies $\mathcal{I}$ **passes** $\mathsf{DT}_a(\epsilon)$ for any $a \leq b$, we have in summary arrived at an on-the-fly algorithm DT_b that for sufficiently large b (corresponding to the length of the test) will be able to conduct a "convicting" doping test for any IUT $\mathcal{I}$ that is not robustly clean w.r.t. a given contract $\mathcal{C}$. The bounded-depth algorithm effectively circumvents the fact that, except for $\mathcal{S}$ and $\mathcal{S}_\delta$, all other objects we need to deal with are countably or uncountably infinite and that the property we check is a hyperproperty.

We implemented a prototype of a testing framework using the bounded-depth algorithm. The specification of distances, value domains and test case selection are parameters of the algorithm that can be set specific for a concrete test scenario. This flexibility enables us to use the framework in a two-step approach for cyber-physical systems not equipped with a digital interface to forward the inputs to: first, the tool can generate test inputs, that are executed by a human or a robot on the CPS under test. The actual inputs (possibly deviating from the generated inputs) and outputs from the system are recorded so that in the second step our tool determines if the (actual) test is passed or failed.

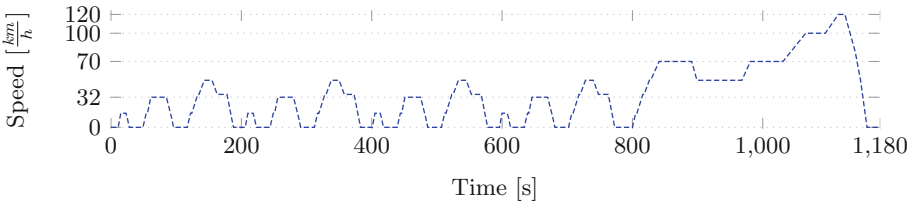


Fig. 2. NEDC speed profile.

6 Evaluation

The normed emission test NEDC (New European Driving Cycle) (see Fig. 2) is the legally binding framework in Europe [28] (at the time the scandal surfaced). It is to be carried out on a chassis dynamometer and all relevant parameters are fixed by the norm, including for instance the outside temperature at which it is run.

For a given car model, the normed test induces a standard LTS $\mathcal{S}$ as follows. The input dimensions of $\mathcal{S}$ are spanned by the sensors the car model is equipped with (including e.g. temperature of the exhaust, outside temperature, vertical and lateral acceleration, throttle position, time after engine start, engine rpm, possibly height above ground level etc.) which are accessible via the standardized OBD-2 interface [24]. The output is the amount of NO_x per kilometre that has been extruded since engine start. Inputs are sampled at equidistant times (once per second). The standard LTS $\mathcal{S}$ is obtained from the trace representing the observations of running NEDC on the chassis dynamometer, say $\sigma_S := i_1 \cdots i_{1180}\, o_S\, \delta\, \delta\, \delta \cdots$ with inputs $i_1, \cdots i_{1180}$ given by the NEDC over its 20 min (1180 s) duration, and o_S is the amount of NO_x gases accumulated during the test procedure. This σ_S is the only standard trace of our experiments. The trace ends with an infinite suffix δ^ω of quiescence steps.

The input space, In is a vector space spanned by all possible input parameter dimensions. For $\boldsymbol{a} \in \mathsf{In}$ we distinguish the speed dimension as $v(\boldsymbol{a}) \in \mathbb{R}$ (measured in km/h). We can use past-forgetful distances with $d_{\mathsf{In}}(\boldsymbol{a}, \boldsymbol{b}) := |v(\boldsymbol{a}) - v(\boldsymbol{b})|$ if $\boldsymbol{a}, \boldsymbol{b} \in \mathsf{In}$, $d_{\mathsf{In}}(-_i, -_i) = 0$ and $d_{\mathsf{In}}(a, b) = \infty$ otherwise. The speed is the decisive quantity defined to vary along the NEDC (cf. Fig. 2). Hence $d_{\mathsf{In}}(\boldsymbol{a}, \boldsymbol{b}) = 0$ if $v(\boldsymbol{a}) = v(\boldsymbol{b})$ regardless of the values of other parameters. We also take $\mathsf{Out} = \mathbb{R}$ for the average amount of NO_x gases per kilometre since engine start (in mg/km). We define $d_{\mathsf{Out}}(a, b) = |a - b|$ if $a, b \in \mathsf{Out}$, and $d_{\mathsf{Out}}(a, b) = \infty$ otherwise.

Doping Tests in Practice. For the purpose of practically exercising doping tests, we picked a Renault 1.5 dci (110*hp*) (Diesel) engine. This engine runs, among others, inside a Nissan NV200 Evalia which is classified as a Euro 6 car. The test cycle used in the original type approval of the car was NEDC (which corresponds to Euro 6b). Emissions are cleaned using *exhaust gas recirculation* (EGR). The technical core of EGR is a valve between the exhaust and intake pipe, controlled by a software. EGR is known to possibly cause performance losses, especially at

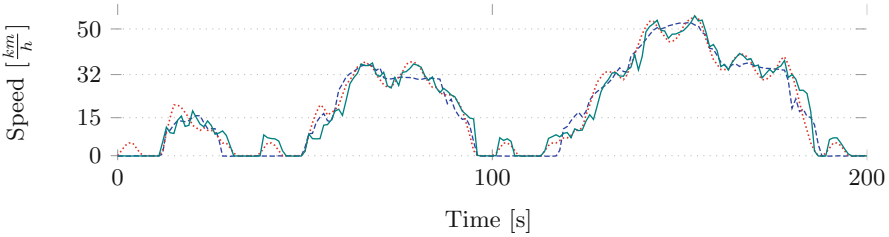


Fig. 3. Initial 200 s of a SineNEDC (red, dotted), its test drive (green) and the NEDC driven (blue, dashed). (Color figure online)

higher speed. Car manufacturers might be tempted to optimize EGR usage for engine performance unless facing a known test cycle such as the NEDC.

We fixed a contract with $\kappa_{\mathsf{i}} = 15\,\text{km/h}$, $\kappa_{\mathsf{o}} = 180\,\text{mg/km}$. We report here on two of the tests we executed apart from the NEDC reference: *(i)* PowerNEDC is a variation of the NEDC, where acceleration is increased from $0.94\frac{m}{s^2}$ to $1.5\frac{m}{s^2}$ in phase 6 of the NEDC elementary urban cycle (i.e. after 56 s, 251 s, 446 s and 641 s) and *(ii)* SineNEDC defines the speed at time t to be the speed of the NEDC at time t plus $5 \cdot sin(0.5t)$ (but capped at 0). Both can be generated by $\mathsf{DT}_{1181}(\epsilon)$ for specific deterministic Ω_{case} and Ω_{In}. For instance, SineNEDC is given below. Fig. 3 shows the initial 200 s of SineNEDC (red, dotted).

$$\Omega_{\text{case}}(h) = \begin{cases} 2 & \text{, if } |h| \leq 1179 \\ 3 & \text{, if } |h| = 1180 \end{cases} \qquad \Omega_{\mathsf{In}}(h) = \max \left\{ \begin{array}{l} 0, \\ \text{NEDC}(|h|) + 5 \cdot \sin(0.5|h|)) \end{array} \right\}$$

The car was fixed on a *Maha LPS 2000* dynamometer and attached to an *AVL M.O.V.E iS* portable emissions measurement system (PEMS, see Fig. 4) with speed data sampling at a rate of 20 Hz, averaged to match the 1 Hz rate of the NEDC. The human driver effectuated the NEDC with a deviation of at most 9 km/h relative to the reference (notably, the result obtained for NEDC are not consistent with the car data sheet, likely caused by lacking calibration and absence of any further manufacturer-side optimisations).

Fig. 4. Nissan NV200 Evalia on a dynamometer

Table 1. Dynamometer measurements (sample rate: 1 Hz)

	NEDC	Power	Sine
Distance [m]	11,029	11,081	11,171
Avg. Speed $\left[\frac{km}{h}\right]$	33	29	34
CO_2 $\left[\frac{g}{km}\right]$	189	186	182
NO_x $\left[\frac{mg}{km}\right]$	180	204	**584**

The PowerNEDC test drive differed by less than 15 km/h and the SineNEDC by less than 14 km/h from the NEDC test drive, so both inputs deviate by less than κ_{i}. The green line in Fig. 3 shows SineNEDC driven. The test outcomes are summarised in Table 1. They show that the amount of CO_2 for the two tests is lower than for the NEDC driven. The NO_x emissions of PowerNEDC deviate

by around 24 mg/km, which is clearly below κ_{o}. But the SineNEDC produces about 3.24 times the amount of NO_x, that is 404 mg/km more than what we measured for the NEDC, which is a violation of the contract. This result can be verified with our algorithm a posteriori, namely by using Ω_{In} to replay the actually executed test inputs (which are different from the test inputs generated upfront due to human driving imprecisions) and by feeding the outputs recorded by the PEMS into the algorithm. As to be expected, this makes the recording of the PowerNEDC return **pass** and the recording of SineNEDC return **fail**.

Our algorithm is powerful enough to detect other kinds of defeat devices like those uncovered in investigations of the Volkswagen or the Audi case. Due to lack of space, we cannot present the concrete Ω_{case} and Ω_{In} for these examples.

7 Discussion

Related Work. The present work complements white-box approaches to software doping, like model-checking [10] or static code analysis [9] by a black-box testing approach, for which the specification is given implicitly by a contract, and usable for on-the-fly testing. Existing test frameworks like TGV [18] or TorX [29] provide support for the last step, however they fall short on scenarios where (i) the specification is not at hand and, among others, (ii) the test input is distorted in the testing process, e.g., by a human driving a car under test.

Our work is based on the definition of robust cleanness [10] which has conceptual similarities to continuity properties [6,17] of programs. However, continuity itself does not provide a reasonably good guarantee of cleanness. This is because physical outputs (e.g. the amount of NO_x gas in the exhaust) usually do change continuously. For instance, a doped car may alter its emission cleaning in a discrete way, but that induces a (rapid but) continuous change of NO_x gas concentrations. Established notions of stability and robustness [13,19,21,23] differ from robust cleanness in that the former assure the outputs (of a white-box system model) to stabilize despite transient input disturbances. Robust cleanness does not consider perturbations but (intentionally) different inputs, and needs a hyperproperty formulation.

Concluding Remarks. This work lays the theoretical foundations for black-box testing approaches geared towards uncovering doped software. As in the diesel emissions scandal – where manufacturers were forced to pay excessive fines [22] and where executive managers are facing lawsuits or indeed went to prison [5,14] – doped behaviour is typically strongly related to illegal behaviour.

As we have discussed, software doping analysis comes with several challenges. It can be performed (i) only after production time on the final embedded or cyber-physical product, (ii) notoriously without support by the manufacturer, and (iii) the property belongs to the class of hyperproperties with alternating quantifiers. (iv) Non-determinism and imprecision caused by a human in-the-loop complicate doping analysis of CPS even further.

Conceptually central to the approach is a contract that is assumed to be explicitly offered by the manufacturer. The contract itself is defined by very few

parameters making it easy to form an opinion about a concrete contract. And even if a manufacturer is not willing to provide such contractual guarantees, instead a contract with very generous parameters can provide convincing evidence of doping if a test uncovers the contract violation. We showed this in a real automotive example demonstrating how a legally binding reference behaviour and a contract altogether induce a finite state LTS enabling to harvest input-output conformance testing for doping tests. We developed an algorithm that can be attached directly to a system under test or in a three-step process, first generating a valid test case, afterwards used to guide a human interacting with the system, possibly adding distortions, followed by an a-posteriori validation of the recorded trajectory. For more effective test case selection [11,15] we are exploring different guiding techniques [1,2,12] for cyber-physical systems.

Acknowledgements. We gratefully acknowledge Thomas Heinze, Michael Fries, and Peter Birtel (Automotive Powertrain Institute of HTW Saar) for sharing their automotive engineering expertise with us, and for providing the automotive test infrastructure. This work is partly supported by the ERC Grant 695614 (POWVER), by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant 389792660 as part of TRR 248, see https://perspicuous-computing.science, by the Saarbrücken Graduate School of Computer Science, by the Sino-German CDZ project 1023 (CAP), by ANPCyT PICT-2017-3894 (RAFTSys), and by SeCyT-UNC 33620180100354CB (ARES).

References

1. Adimoolam, A., Dang, T., Donzé, A., Kapinski, J., Jin, X.: Classification and coverage-based falsification for embedded control systems. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 483–503. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_24
2. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19835-9_21
3. Barthe, G., D'Argenio, P.R., Finkbeiner, B., Hermanns, H.: Facets of software doping. In: Margaria and Steffen [20], pp. 601–608. https://doi.org/10.1007/978-3-319-47169-3_46
4. Baum, K.: What the hack is wrong with software doping? In: Margaria and Steffen [20], pp. 633–647. https://doi.org/10.1007/978-3-319-47169-3_49
5. BBC: Audi chief Rupert Stadler arrested in diesel emissions probe. BBC (2018). https://www.bbc.com/news/business-44517753. Accessed 28 Jan 2019
6. Chaudhuri, S., Gulwani, S., Lublinerman, R.: Continuity analysis of programs. In: Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, 17–23 January 2010, pp. 57–70. ACM (2010). http://doi.acm.org/10.1145/1706299.1706308
7. 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, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54792-8_15

8. Clarkson, M.R., Schneider, F.B.: Hyperproperties. In: CSF 2008, pp. 51–65 (2008). http://dx.doi.org/10.1109/CSF.2008.7
9. Contag, M., et al.: How they did it: an analysis of emission defeat devices in modern automobiles. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, 22–26 May 2017, pp. 231–250. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.66
10. D'Argenio, P.R., Barthe, G., Biewer, S., Finkbeiner, B., Hermanns, H.: Is your software on dope? In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 83–110. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_4
11. de Vries, R.: Towards formal test purposes. In: Formal Approaches to Testing of Software 2001 (FATES 2001). BRICS Notes Series, No. NS-01-4, pp. 61–76. BRICS, University of Aarhus, August 2001
12. Deshmukh, J., Jin, X., Kapinski, J., Maler, O.: Stochastic local search for falsification of hybrid systems. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 500–517. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24953-7_35
13. Doyen, L., Henzinger, T.A., Legay, A., Nickovic, D.: Robustness of sequential circuits. In: 10th International Conference on Application of Concurrency to System Design, ACSD 2010, Braga, Portugal, 21–25 June 2010, pp. 77–84. IEEE Computer Society (2010). https://doi.org/10.1109/ACSD.2010.26
14. Ewing, J.: Ex-Volkswagen C.E.O. Charged With Fraud Over Diesel Emissions. New York Times (2018). https://www.nytimes.com/2018/05/03/business/volkswagen-ceo-diesel-fraud.html. Accessed 28 Jan 2019
15. Feijs, L.M.G., Goga, N., Mauw, S., Tretmans, J.: Test selection, trace distance and heuristics. In: Testing of Communicating Systems XIV, Applications to Internet Technologies and Services, Proceedings of the IFIP 14th International Conference on Testing Communicating Systems - TestCom 2002, Berlin, Germany, 19–22 March 2002. IFIP Conference Proceedings, vol. 210, pp. 267–282. Kluwer (2002)
16. Finkbeiner, B., Rabe, M.N., Sánchez, C.: Algorithms for model checking HyperLTL and HyperCTL*. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 30–48. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_3
17. Hamlet, D.: Continuity in sofware systems. In: Proceedings of the International Symposium on Software Testing and Analysis, ISSTA 2002, Roma, Italy, 22–24 July 2002, pp. 196–200. ACM (2002). https://doi.org/10.1145/566172.566203
18. Jard, C., Jéron, T.: TGV: theory, principles and algorithms. STTT **7**(4), 297–315 (2005)
19. Majumdar, R., Saha, I.: Symbolic robustness analysis. In: Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1–4 December 2009, pp. 355–363. IEEE Computer Society (2009). https://doi.org/10.1109/RTSS.2009.17
20. Margaria, T., Steffen, B. (eds.): ISoLA 2016, Part II. LNCS, vol. 9953. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47169-3
21. Pettersson, S., Lennartson, B.: Stability and robustness for hybrid systems. In: Proceedings of 35th IEEE Conference on Decision and Control, vol. 2, pp. 1202–1207, December 1996
22. Riley, C.: Volkswagen's diesel scandal costs hit $30 billion. CNN Business (2018). https://money.cnn.com/2017/09/29/investing/volkswagen-diesel-cost-30-billion/index.html. Accessed 28 Jan 2019

23. Tabuada, P., Balkan, A., Caliskan, S.Y., Shoukry, Y., Majumdar, R.: Input-output robustness for discrete systems. In: Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2012, Part of the Eighth Embedded Systems Week, ESWeek 2012, Tampere, Finland, 7–12 October 2012, pp. 217–226. ACM (2012). http://doi.acm.org/10.1145/2380356.2380396
24. The European Parliament and the Council of the European Union: Directive 98/69/ec of the european parliament and of the council. Official Journal of the European Communities (1998). http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31998L0069:EN:HTML
25. Tretmans, J.: A formal approach to conformance testing. Ph.D. thesis, University of Twente, Enschede, Netherlands (1992). http://purl.utwente.nl/publications/58114
26. Tretmans, J.: Conformance testing with labelled transition systems: implementation relations and test generation. Comput. Netw. ISDN Syst. **29**(1), 49–79 (1996). https://doi.org/10.1016/S0169-7552(96)00017-7
27. Tretmans, J.: Model based testing with labelled transition systems. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) Formal Methods and Testing. LNCS, vol. 4949, pp. 1–38. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78917-8_1
28. 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
29. de Vries, R.G., Tretmans, J.: On-the-fly conformance testing using SPIN. STTT **2**(4), 382–393 (2000). https://doi.org/10.1007/s100090050044