2018 3rd Workshop on Monitoring and Testing of Cyber-Physical Systems

Cyber-Physical Doping Tests

Sebastian Biewer, Holger Hermanns
Saarland University – Computer Science
Saarland Informatics Campus
Saarbrücken, Germany

Pedro R. D'Argenio
Universidad Nacional de Córdoba – FaMAF
CONICET
Córdoba, Argentina

***Abstract*—We are confronted with a growing number of cases where device manufacturers equip their products with embedded software that includes functionalities that are not in the owner's interest. Examples include customer lock-in strategies in inkjet printers and as a prominent case the diesel emissions scandal in the automotive industry. This *software doping* phenomenon is turning more widespread as software is embedded in ever more devices of daily use.**

In this work we present a formal characterization which can distinguish *clean* and *doped* reactive programs, based on a contract that is assumed to exist between the end user of a cyber physical device and the manufacturer of the control software embedded therein. We further discuss our current work on combining this characterization with the theory of model-based testing, so as to arrive at a formal basis upon which it will be possible to perform efficient doping tests in practice.

I. Motivation

Program verification and testing are methods for software manufacturers to check if their products satisfy certain objectives. Classically, these objectives agree with those of the users or the general interest. However, we observe a trend where the interests of the manufacturers diverge from the general interest, in particular in the context of embedded and cyber-physical systems. If the software includes functionality that is in the mere interest of the manufacturer, we call this software being *doped*.

Examples of software doping include customer lock-in strategies as found for instance in inkjet printers [1] that refuse to work when supplied with a toner or ink cartridge of a third party manufacturer despite technical compatibility, and in laptops that refuse to charge if connected to a third-party battery charger [2]. Such functionalities are clearly in the interest of the manufacturer, because they boost demand for original manufacturer accessories or replacement parts instead of (compatible) third-party parts. They are not in the interest of the user because of the often excessive pricing of OEM parts.

The diesel emission scandal received a lot of attention and is another example of software doping. Modern cars need to comply to a range of environmental regulations limiting the level of emissions for various toxic substances, greenhouse gases, and particles. The prime approach to assure compliance with these regulations is black-box testing carried out in a controlled environment: Emission tests are carried out on a chassis dynamometer where the car is fixed but tires can rotate freely. During the test, emissions are measured at the exhaust pipe while the vehicle is made to follow a precisely defined profile meant to imitate real driving conditions. The conditions of the test, including speed profile and other details such as outside temperature, are both standardized and public, ensuring that the testing can be carried out in a reproducible way by an independent party, treating the car itself as a black box.

However, the singularity of the conditions on the chassis dynamometer makes it possible to infer when a car is undergoing an emission test and to intentionally adjust the car behaviour so as to comply with emission standards, while exceeding them during normal driving in favour of more economic resource usage. This surreptitious alteration of functionality is at the heart of the *diesel emissions scandal*. It has taken place in millions of cars equipped with diesel engines in a broad spectrum of vehicle models originating from various manufacturers. A detailled account of how this was achieved in the case of Volkswagen and of Fiat-Chrysler has been given [3], illustrating a variety of embedded control mechanisms with surreptitious functionality inside the cars we drive.

Common to all these examples is that the software user has little or no control over its execution, and that the functionality in question is against the interests of user or of society. At the core of this functionality are proprietary software artefacts and for the case of the diesel scandal the manufacturers promise to remove the undesired functionality by an update of their proprietary software.

Many more examples of software doping exist [4] and it is turning more widespread as software is embedded in ever more devices of daily use. The future will thus likely see many more facets of software doping pestering us.

II. Clean and doped software

There is thus a critical research agenda gaining momentum which focusses on means to identify, understand and eventually remedy the various facets of software doping. As a pivotal step in this endeavour, we need to be able to tell apart *doped* software from *clean* software, be it to certify regulatory compliance in a cyberphysical environment, be it to prevent customer lock-in or planned obsolescence by software, be it to empower trust of device users. The problem of software doping has been recognized in the literature [5], [4], [6], [7], [3], and a hierarchy of simple but solid formal definitions has lately been proposed [8]. It is based on a model of the embedded software behaviour which in turn is derived from

978-1-5386-6748-4/18/$31.00 ©2018 IEEE
DOI 10.1109/MT-CPS.2018.00016

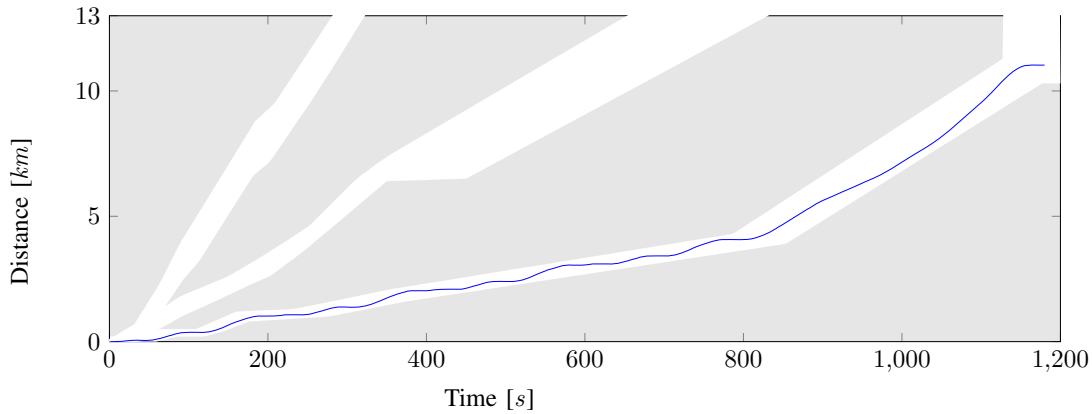

Figure 1: Time-distance profiles of 2013 Sharan and NEDC

a contract that is assumed to be explicitly offered by the manufacturer. The contract includes the identities of all sensor inputs relevant to the embedded control problem at hand, a number of reference tests, and a pair of values identifying in how far variations in input readings may possibly induce variations in output behaviour. In this contract, the reference tests echo the nowadays common practice of public authorities to check compliance with respect to legal requirements by performing a few (black-box) tests under lab conditions that are known publicly, so as to assure reproducibility. The need for wrapping these reference tests into a broader contract stems from the fact that otherwise software that alters its behaviour if outside the singularities of the lab conditions would be considered clean, not doped.

III. Doped emission cleaning

Fig. 1 illustrates the problem in the context of the diesel emission system used in a Volkswagen Sharan model from 2013. The engine control unit (ECU) of this car contains several pairs of piecewise-linear time-distance functions which define the gray and white areas in Fig. 1. If the current trip (i.e., the time-distance profile of the vehicle after engine start) stays within the white area, the ECU performs the best possible emission cleaning. However, once it moves outside of the white area emission cleaning is turned off [3]. The blue curve in the lower white area is the time-distance profile of the New European Driving Cycle (NEDC), which is the only test diesel cars had to pass for admission in EU in 2013. Noticeably, the boundaries of the white area tightly enclose this curve.

In order to capture in how far a clean software is allowed to deviate in emissions if the driving profile deviates from the standard behaviour (e.g. as it materializes on a chassis dynamometer when driving the NEDC) the definition put forward in [8] assumes a contract to exist between software manufacturer and user. Intuitively, this contract is meant to ensure that if the input to the control software changes only slightly (e.g., if the distance in Fig. 1 crosses the gray area briefly), then the output of the program (e.g. the number of NO_x particles in the exhaust gas) is to deviate only in a reasonably bounded way, too. This is not the case for the Sharan, since emission cleaning is turned off once hitting the gray area whence the number of particles in the exhaust gas increases significantly.

IV. From contract to test

The above mentioned contractual formulation provides a basis to distinguish whether a program is clean or doped, and is amenable to model-checking style verification [8]. This, however, requires a white-box setting, i.e., a setting where the embedded control program is known to the analysis in all details. But since commercial embedded control software usually is of a proprietary nature, a black-box approach to software doping is needed instead. Indeed, the contract definitions can be twisted so as to enable the generation of model-based test cases tailored for discovering the existence of software doping. This approach is being developed into a full-fledged methodology as ongoing work.

Acknowledgment

This work is partly supported by ERC Advanced Grant 695614 (POWVER), by the Saarbrücken Graduate School of Computer Science, by the Sino-German CDZ project 1023 (CAP), and by SeCyT-UNC 05/BP12 and 05/B497.

References

[1] J. C. Dvorak, "The secret printer companies are keeping from you," PC Mag UK, http://uk.pcmag.com/printers/60628/opinion/the-secret-printer-companies-are-keeping-from-you, 2012, Online; accessed: 2018-03-23.

[2] Tritech Computer Solutions, "Dell laptops reject third-party batteries and AC adapters/chargers. Hardware vendor lock-in?" https://nctritech.wordpress.com/2010/01/26/dell-laptops-reject-third-party-batteries-and-ac-adapterschargers-hardware-vendor-lock-in/, 2010, Online; accessed: 2018-03-23.

[3] M. Contag, G. Li, A. Pawlowski, F. Domke, K. Levchenko, T. Holz, and S. Savage, "How they did it: An analysis of emission defeat devices in modern automobiles," in *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017, pp. 231–250.

[4] G. Barthe, P. R. D'Argenio, B. Finkbeiner, and H. Hermanns, "Facets of software doping," in *7th International Symposium on Leveraging Applications of Formal Methods, ISOLA 2016, Part II*, ser. LNCS, vol. 9953. Springer, 2016, pp. 601–608.

[5] L. Hatton and M. van Genuchten, "When software crosses a line," *IEEE Software*, vol. 33, no. 1, pp. 29–31, 2016.

[6] M. Huisman, H. Bos, S. Brinkkemper, A. van Deursen, J. F. Groote, P. Lago, J. van de Pol, and E. Visser, "Software that meets its intent," in *7th International Symposium on Leveraging Applications of Formal Methods, ISOLA 2016, Part II*. Springer, 2016, pp. 609–625.

[7] K. Baum, "What the hack is wrong with software doping?" in *7th International Symposium on Leveraging Applications of Formal Methods, ISOLA 2016, Part II*, ser. LNCS, vol. 9953. Springer, 2016, pp. 633–647.

[8] P. R. D'Argenio, G. Barthe, S. Biewer, B. Finkbeiner, and H. Hermanns, "Is your software on dope? - Formal analysis of surreptitiously "enhanced" programs," in *26th European Symposium on Programming, ESOP 2017*, ser. LNCS, vol. 10201. Springer, 2017, pp. 83–110.