POWVER

Technical Report 2018-11

Title: **Efficient Monitoring of Real Driving Emissions**

Authors: Maximilian A. Köhl, Holger Hermanns, Sebastian Biewer

Report Number: 2018-11

ERC Project: Power to the People. Verified.

ERC Project ID: 695614

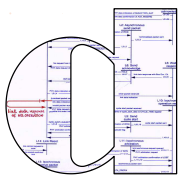
Funded Under: H2020-EU.1.1. – EXCELLENT SCIENCE

Host Institution: Universität des Saarlandes, Dependable Systems and Software
Saarland Informatics Campus

Published In: RV 2018

Please cite this publication as follows:

Maximilian A. Köhl, Holger Hermanns, Sebastian Biewer.
Efficient Monitoring of Real Driving Emissions.
Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings. Lecture Notes in Computer Science 11237, Springer 2018, ISBN 978-3-030-03768-0. 299-315.



POWER TO THE PEOPLE.
VERIFIED.
erc

Efficient Monitoring of Real Driving Emissions*

Maximilian A. Köhl, Holger Hermanns, and Sebastian Biewer

Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
{mkoehl,hermanns}@cs.uni-saarland.de
biewer@depend.uni-saarland.de

Abstract. The diesel emissions scandal has demonstrated that real-world behavior of systems can deviate excessively from the behavior shown under certification conditions. In response to the massive revelation of fraudulent behavior programmed inside diesel cars across Europe, the European Union has defined a procedure to test for *Real Driving Emissions* (RDE) [22]. This is gradually being put into force since September 2017 [23]. To avoid misinterpretation, the RDE regulation comes with an informal but relatively precise specification that spells out in how far a real trip, i.e., a trajectory driven with a car, constitutes an RDE test, or not. This paper presents a formalization of the RDE test procedure which is used to monitor for RDE violations at runtime and thereby fosters perspicuity. To this end, we extend the stream-based specification language Lola [5,10] with sliding aggregation windows. We evaluate the approach experimentally using data from real trips and further present a low-cost variant of the RDE test which can be conducted without expensive test equipment solely with on-board sensors.

Keywords: Automotive Testing · Runtime Monitoring · Specification Languages · Software Doping · Perspicious Systems.

1 Introduction

The recent diesel emissions scandal has put the problem of doped software [6] in the spotlight: proprietary embedded control software may decide to exploit functionality offered by a device against the best interest of the device owner or of society, in favor of interests of the manufacturer. Concretely, the controllers embedded in many diesel-powered cars are programmed in ways that induce substantial environmental pollution, in violation of many emission regulations around the world. This escapes detection through official test procedures because the behavior is programmed to surreptitiously change whenever the car is deemed to be in a test setting. This is easily possible, since, at least so far, emission test procedures were carried out in a precisely defined environment, and were following a precisely defined driving profile, with the car under test fixed on a chassis dynamometer. This precision is needed so as to ensure reproducibility of

*This research was supported in part by the Saarbrücken Graduate School of Computer Science and by ERC Advanced Grant 695614 (POWVER).

the tests and to enable comparisons of exhaust footprint and fuel consumption across different car models.

For about a decade, the binding standard to be used during type approval of a new car model has been the *New European Driving Cycle* (NEDC), which has recently been replaced by the *Worldwide harmonized Light vehicles Test Procedure* (WLTP) [23]. The latter is considered to be more realistic, but it still shares the problematic characteristics of the NEDC in that the WLTP driving profile is very much a singularity, and therefore easy to identify by control software doped by the manufacturer. To overcome this conceptual problem, the European Union has lately defined a new procedure to test for *Real Driving Emissions* (RDE). This comes with broad certification conditions for tests which are to be conducted under real-world conditions, on public roads and during working days. The RDE is gradually being put into force since September 2017. The RDE complements the WLTP—while the RDE is intended to measure emissions under real-world conditions the WLTP is intended to measure fuel consumption in a reproducible manner enabling comparability.

To avoid misinterpretation, the RDE regulation comes with an informal but relatively precise specification document [23] that spells out in how far a road trip, i.e., a trajectory driven with a car in-the-wild, constitutes an RDE test, or not. This specification contains constraints on the route, allowed altitude and speed, and on the dynamics of the driving profile, that make use of percentiles. The specification also accounts for dynamic conditions like the weather. Conducting an RDE test requires a PEMS, a *Portable Emissions Measurement System*. This is a device that measures the emissions at the tailpipe of a vehicle and is small and light enough to be carried inside or moved with the vehicle during the test drive. The unit price of a PEMS is in the order of $250,000. Commercial software such as "AVL Concerto for PEMS" is used to effectuate the measurement, collect the relevant data, and to decide if the test performed is indeed an RDE or not [1]. As usual for proprietary software, the source code of AVL Concerto and similar programs is not available, so there is no direct check available to reassure the verdict of the program after a test drive.

This paper phrases the question of RDE compliance as a runtime monitoring problem. Its central contribution is a formalization of the RDE regulation. For this, we extend the stream-based specification language Lola [10,5] with sliding aggregation windows enabling the efficient computation of percentiles and moving averaging windows as needed by the RDE regulation.

We exploit this formalization in a low-cost variant of the RDE test procedure for NO_x emissions which only uses on-board sensors instead of an expensive PEMS. The hardware cost of our system is in the order of $100. With our openly available formalization[1] at its core, the system implements the blueprint of an independent emission control and compliance system which we use to empirically monitor real vehicles under real driving conditions. The empirical results we can report are very encouraging. We discuss the pros and cons of our solution. In

[1] All details of the monitor, including the source code of the Lola specification and RDE trip data, can be found at `https://www.powver.org/real-driving-emissions/`.

addition, we show that our extension of Lola can be compiled into plain Lola, albeit at the price of losing succinctness.

The contributions of this paper are an extended version of the stream-based specification language Lola with sliding aggregation windows, an elaborate study on the formalization of the RDE test procedure using this extended version of Lola, and the presentation of experimental results which make use of a low-cost version of the RDE without expensive equipment.

Organization of the paper. In Section 3 we briefly introduce Lola [10] and the RDE regulation [23]. In Section 4 we extend Lola with sliding aggregation windows. In Section 5 we present the RDE test procedure and its formalization displaying the capabilities of the freshly introduced sliding aggregation windows. In Section 6 we evaluate the RDE formalization and the thereof constructed monitor experimentally using data from real trips. Furthermore, we show how a runtime monitor can be used to continuously supervise cars in use.

2 Related Work

We implement our RDE monitor using an extended version of the stream-based specification language Lola 2.0 [5,10]. Lola can express complex temporal properties referring to the past and the future. It can be used for checking if single traces satisfy given properties and generating statistical measures. Lola allows the computation of output streams, which are instances of *templates* specifying when and how streams are computed. *Triggers* are used to define boolean properties based on input and output streams. Lola 2.0 supports instance aggregation functions (e.g. exists, forall, count, ...) for output streams enabling reasoning about all active instances of a certain stream template.

Recently, Lola 2.0 has been extended to RTLola, which supports real-time properties [11] and is especially useful when data does not arrive with a fixed frequency. The RDE, however, is based on test parameters provided with a fixed sampling frequency. Statistical measurements for real-time data have been realized by using *sliding aggregation windows* over discrete real-time intervals, and due to the real-time semantics, arbitrary many sample points. Currently, no implementation of RTLola is available. In our work, we follow a similar approach by extending Lola with sliding aggregation windows over a fixed number of data points. In Lola 2.0, aggregation functions can only be used to aggregate data of several instances at the *current time*, whereas sliding aggregation windows aggregate data from an interval.

Temporal logic [15,16] has been extended for real-time systems in MITL [2]. Signal Temporal Logic (STL) [13,14] introduces real-valued signals to MITL. The logic can specify past or future behavior. However, unlike Lola, it cannot relate values of the stream at different points in time. Further, it is not possible to generate the statistical measures required to validate an RDE test. Hence, it is not suitable for encoding the RDE regulation. An approach to extend STL for properties, that need a global view on the data, has already been proposed [8].

	Urban	Rural	Motorway
Ratio Range [%]	$[29, 44]$	$[23, 43]$	$[23, 43]$
Speed Range [km/h]	$[0, 60]$	$]60, 90]$	$]90, 160]$
Distance [km]	≥ 16	≥ 16	≥ 16
Additional Constraints	stop percentage between 6% and 30% of urban time; average velocity in range $[15, 40]$km/h		$>$ 100km/h for at least 5mins
Temperature [K]	moderate: $[273, 303]$; extended: $[266, 273[$ or $]303, 308]$		
Altitude [m]	moderate: < 700; extended: $]700, 1300]$		
Speed Limit [km/h]	145 ($]145{,}160]$ for at most 3% of motorway time)		

Table 1. Some constraints for the urban, rural and motorway phase of RDE tests.

The semantics of temporal logics has been extended from boolean satisfaction of a formula to robustness values [9,7,17]. Positive numbers indicate that the property is satisfied, a negative value shows the opposite. Falsification techniques for reactive systems try to make such a value smaller to eventually make it negative and disprove the property [3].

Efficient algorithms for the incremental computation of sliding window aggregations have been extensively studied [12]. This work allows us to make use of these algorithms within Lola using a standardized interface.

Currently, RDE tests are conducted using Portable Emissions Measurement Systems, e.g., [1]. The commercial software that is delivered with these systems includes ready-to-use RDE monitors.

3 Preliminaries

3.1 Real Driving Emissions

Although the RDE regulation specifies broad testing conditions, there are still some constraints which should guarantee that a trip is close to real-driving conditions making it neither too easy nor too hard for the vehicle to pass the test. For this, valid RDE drives take 90 to 120 minutes and traverse three phases: urban, rural, and motorway. The regulation defines minimum and maximum permissible ratios of each phase w.r.t. the whole test distance. Moreover, the regulation defines speed constraints, minimum distances, ambient conditions, and more. Table 1 shows some of those constraints which we will formalize in Section 5.

3.2 Lola 2.0: An Introduction

Lola 2.0 [10] is a stream-based specification language based on its predecessor Lola [5]. This paper uses Lola 2.0, however, for readability reasons we draw

this distinction explicitly only where it is relevant and otherwise refer to LOLA 2.0 simply as LOLA. This section aims to give a brief introduction to LOLA, for further details we refer to the original publications [5,10].

LOLA provides an evaluation model based on synchronous streams where output streams are computed based on input streams. To this end, a LOLA 2.0 specification comprises a declaration of N typed input stream variables t_i and M typed parameterized output stream variables s_j that get assigned stream expressions which specify how the respective output streams are computed from values of the input streams, output streams, and parameters.

Input stream variables are declared by

`input` T_i t_i

---

where T_i is the type of input stream variable t_i.

Parameterized *output stream variables* are declared by *templates* of the form

---

`output` T_j $s_j\langle p_1 : T_1^j, \ldots, p_k : T_k^j\rangle$ `:` `inv:` s_{inv}`;` `ext:` s_{ext}`;` `ter:` s_{ter}
`:=` $e(t_1, \ldots, t_N, s_1, \ldots, s_M, p_1, \ldots, p_k)$

---

where T_j is the type of output stream variable s_j. Concrete streams $s_j(\alpha)$, i.e., *instances* of s_j, are identified by *parameter valuations* $\alpha \in T_1^j \times \ldots \times T_k^j$. Each instance has a local clock. Instances $s_j(\alpha)$ are invoked with local time 0 whenever a tuple α appears on the *invocation stream* s_{inv} for which there does not already exist an instance. The local time of an instance advances with an increment of 1 whenever `true` appears on the boolean *extension stream* s_{ext} which is required to have the same *parameter signature* $\vec{P} = \langle p_1 : T_1^j, \ldots, p_k : T_k^j\rangle$ as the template for s_j. Whenever the extension stream is `true`, a new value is computed by the *stream expression* e over stream variables and parameters which is then appended to the stream. Otherwise, the previous value remains valid and the local time does not advance. If `true` appears on the boolean *termination stream* s_{ter}, again with signature $\vec{P}$, the instance is terminated. Intuitively, the invocation of a new instance creates a new output stream that produces values with each tick of its local clock and which ends on termination of the instance. An instance is *alive* starting with its invocation until its termination. Input streams are alive from the beginning until termination of the monitor.

In addition to the input and output streams, there is a stream producing the constant empty tuple in every global step. If streams without parameters are defined, then this stream is used as the invocation stream which thus is omitted in the template. Additionally, if s_{ext} is omitted instances are extended with every global step and if s_{ter} is omitted instances are never terminated.

Stream expressions are defined inductively. Constants c and *parameter variables* p_i are *atomic stream expressions*. Let $e_1, \ldots, e_k$ be stream expressions of types $T_1, \ldots, T_k$. If f is a k-ary function of type $T_1 \times T_2 \times \ldots \times T_k \to T$, then $f(e_1, \ldots, e_k)$ is a stream expression of type T. If b is a boolean stream expression and $T_1 = T_2$, then $ite(b, e_1, e_2)$[2] is a stream expression of type T_1. Let

[2] *ite* is short for if-then-else

$k \in \mathbb{Z}$, d be a constant of type T_1, and s be an output stream variable. Then $s(\vec{p})[k, d]$ is a stream expression of type T_1 where $\vec{p}$ are *parameters* comprised of atomic stream expressions matching the signature of s. Further, for an *instance aggregation operator* O, $O(s)$ is an expression.

Semantically, constants, parameter variables, functions, and if-then-else constructs are defined as usual. The semantics of *offset expressions* $s(\vec{p})[k, d]$ are defined as follows: if the parameter tuple $\vec{p}$ evaluates to α, then $s(\vec{p})[k, d]$ is the value of the instance $s(\alpha)$ at local time $t + k$ where t is the local time of instance $s(\alpha)$ at the global time step that the expression is evaluated. If the instance $s(\alpha)$ is not alive at the global time the expression is evaluated or if the local time $t + k$ refers to a point before the invocation or after the termination, the value of the offset expression is the default value d. Instance aggregation operators compute properties about all instances of a template. For example, COUNT(S) returns the number of active instances of output stream template s.

Notice that input stream variables can be used wherever output stream variables are expected by copying the input to an output stream.

LOLA allows the declaration of triggers,

```
trigger φ
```

where φ is a boolean stream expression. The trigger is activated if φ becomes true. Triggers usually indicate the violation of a property.

4 Sliding Aggregation Windows

In the previous section, we briefly introduced the RDE regulation and the LOLA specification language in which we aim to formalize the regulation. The RDE regulation assesses the overall driving dynamics of a trip in terms of the 95% percentile of speed times positive acceleration. In this section, we extend LOLA 2.0 with *sliding aggregation windows* which enable the efficient computation and perspicuous specification of percentiles.

In [11] LOLA 2.0 has been extended with *real-time* sliding aggregation windows which allow for efficient aggregation over windows comprising an unbounded amount of values specified in terms of real-time intervals. Our extension of LOLA 2.0 complements this extension by allowing the computation of aggregated values over windows of fixed width in terms of values taken into account.

For an introductory example, see the definition of stream $\widetilde{va_{95}}$

```
output float va95~ := percentile95(va[-n:0 | a_is_positive])
```

which computes the 95% percentile of speed times positive acceleration over the last `n` samples. In general, an aggregation window comprises an *aggregation function* (`percentile95`) and a *window expression* which is composed of a parameterized stream variable (`va`), a *window specifier* (`-n:0`), and an optional condition (`a_is_positive`).

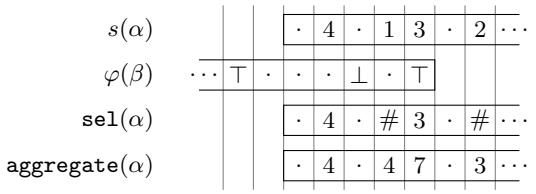


Fig. 1. Example of an Unrolled Aggregation Window: $sum(s(\alpha)[-3:0 \mid \varphi(\beta)])$

Intuitively a sliding aggregation window aggregates over those values within the bounds of the window specifier for which the condition is satisfied by applying the aggregation function to the sequence of these values.

Formally, we extend the syntax of Lola 2.0's stream expressions with sliding aggregation window expressions of the form

$$f\left(s(\vec{p_s})\left[i:j \mid \varphi(\vec{p_\varphi})\right]\right) \tag{1}$$

where $f : T^* \to T'$ is an aggregation function mapping possibly empty sequences of values of type T to a value of type T', $s(\vec{p_s})$ is a stream variable of type T with parameters $\vec{p_s}$, $i, j \in \mathbb{Z}$ with $j \geq i$ define the window boundaries, and $\varphi(\vec{p_\varphi})$ is a boolean stream variable with parameters $\vec{p_\varphi}$. We further require that the parameters $\vec{p_\varphi}$ only contain parameter variables also occurring in $\vec{p_s}$ or constants, which relates exactly one φ instance to each s instance. The type of the whole expression (1) is the target set T' of the aggregation function.

As we will show in the following, technically the introduction of sliding aggregation windows does not allow us to express any new properties, because aggregation windows can be rewritten to ordinary Lola 2.0 syntax. Nevertheless, they have the advantage that they are much more succinct making them more intuitive and easier to write than their rewritten equivalents. Thereby they foster perspicuity of the specification—for which we strive.

4.1 Explicit Unrolling Semantics

We start with a naive rewriting rule for an explicit unrolling of the sliding aggregation window into the syntax of Lola 2.0.

To unroll an aggregation window expression we define a new n-ary function f' where n is the window width, i.e., $n = j - i$, and manually hand over every value of the respective s instance to f' for each position in the window where the condition holds using an offset. To this end, we introduce a new unique value # to the type of s and define an auxiliary stream template `sel` whose instances run in tandem with the respective instances of s. Whenever a new value for an s instance becomes available, we extend the respective `sel` instance with this value if the condition holds, otherwise with #. See Fig. 1 for an example.

`output` $T_s^{\#}$ `sel`$\langle \vec{P} \rangle$ `:` `inv:` s_{inv}`;` `ext:` s_{ext}`;` `ter:` s_{ter} `:=`
  $ite(\varphi(\vec{p_\varphi}')$`[0,false],` $s(\vec{p})$`[0,#],` `#)`

Functions	Space	Update Cost
sum, *avg*, *width*	$\mathcal{O}(n)$	worst-case $\mathcal{O}(1)$
count, *any*, *all*	$\mathcal{O}(n)$	worst-case $\mathcal{O}(1)$
max, *min*	$\mathcal{O}(n)$	amortized $\mathcal{O}(1)$
median, *percentile*	$\mathcal{O}(n)$	worst-case $\mathcal{O}(\log n)$

Table 2. Aggregation Function Costs as a Function of Window Size [12]

The invocation, extension, and termination streams are the same as for the stream template s we aggregate over, i.e., the instances $s(\alpha)$ and $\texttt{sel}(\alpha)$ for parameter valuation α are invoked, terminated, and extended together. $\vec{P}$ denotes the parameter signature of s and $\vec{p}$ passes those parameters on to s to select the corresponding instance of s. By requiring that $\vec{p_\varphi}$ contains only parameters also appearing in $\vec{p_s}$ or constants, we can reconstruct the parameters for φ using only the parameters passed to `sel`. This reconstruction is denoted by $\vec{p_\varphi}'$.

The result are streams that only contain those values of the respective s instances for which the condition holds. Notice that, although we specify a default value for $s(\vec{p})$ in the consequence of the conditional, $s(\vec{p})$ will never be undefined. It remains to pass those values with explicit offsets to the function f'.

`output` T' `aggregate`$\langle\vec{P}\rangle$ `: inv:` s_{inv}`; ext:` s_{ext}`; ter:` s_{ter} `:=`
  f'`(sel(`$\vec{p}$`)[i+1,#], sel(`$\vec{p}$`)[i+2,#], ..., sel(`$\vec{p}$`)[j,#])`

The function f' computes the value of the aggregation using f by constructing a sequence from its parameters in parameter order ignoring those that are #. Now, the aggregation window can be rewritten as follows

$$f(s(\vec{p_s})[i:j \mid \varphi(\vec{p_\varphi})]) \rightsquigarrow \texttt{aggregate}(\vec{p_s})[0, f'(\#^n)]$$

where $f'(\#^n) = f(\epsilon)$ is the aggregation function's default value in case there is no s instance for the respective parameters—which implies, that there also is no instance of `aggregate` for the respective parameters.

4.2 Efficient Aggregation Windows

With explicit unrolling, we can rewrite sliding aggregation windows to the usual syntax of Lola and hence use the Lola monitoring algorithm as is. However, the standard monitoring algorithm of Lola will recompute the aggregation functions from scratch for every window change, although many aggregation functions like summation or average allow for a much more efficient incremental updating strategy [12,11]. To this end, we present a more sophisticated rewrite rule which utilizes incremental updates and thereby allows us to compute aggregation functions much more efficiently. Table 2 provides an overview of selected aggregation functions as well as their space and update costs.

We follow [12] and define an abstract interface to aggregation algorithms that reuse intermediate results. For an aggregation function $f : T^* \to T'$ let D be an intermediate aggregation domain and $\epsilon \in D$ a unique initial value. We define three operations on intermediate aggregation values, $\mathit{insert} : D \times T \to D$ adds a new value of type T to an intermediate aggregate, $\mathit{evict} : D \times T \to D$ evicts an old value from an intermediate aggregate, and $\mathit{lower} : D \to T'$ lowers an intermediate aggregate into an aggregated value of type T'. Insertions and evictions happen in FIFO order, which will be guaranteed by our translation. For each aggregation function, we choose an aggregation algorithm.

Computing aggregation windows over streams with the defined interface is then straightforward. For each window one constructs the `sel` stream template as given in Section 4.1. Instead of computing the value with `aggregate` from scratch for every change, one stores an intermediate aggregation, inserts new values $v \neq \#$ whenever they become available on the corresponding `sel` instance and evicts old values $v \neq \#$ when they shift out of the window. This algorithm can be directly implemented within Lola 2.0 itself:

```
output D ins⟨P⃗⟩ : inv: s_inv; ext: s_ext; ter: s_ter :=
  ite(sel[j,#] ≠ #, insert(agg[-1,ϵ], sel[j,#]), agg[-1,ϵ])
output D agg⟨P⃗⟩ : inv: s_inv; ext: s_ext; ter: s_ter :=
  ite(sel[i,#] ≠ #, evict(ins[0,ϵ], sel[i,#]), ins[0,ϵ])
```

The stream template `agg` produces instances running in tandem with s instances and stores the intermediate results. First, the stream `ins` inserts new values appearing on the respective `sel` instance. Then, if a value is sliding out of the window, it is evicted by the `agg` stream. With this, rewriting aggregation windows is possible as follows:

$$f(s(\vec{p_s})[i : j \mid \varphi(\vec{p_\varphi})]) \rightsquigarrow \mathit{lower}(\mathtt{agg}(\vec{p_s})[0, \epsilon])$$

Rewriting the specification instead of extending the monitoring algorithm has the advantage that the core of Lola 2.0 stays small and is, therefore, easier to implement and reduces the chance for bugs. Additionally, our extension can be used directly with existing implementations of and optimizations for Lola 2.0. Using the standardized interface suggested in [12] we utilize existing research on sliding aggregation windows.

5 Formalizing Real Driving Emissions

We now have everything needed to formalize the RDE test procedure. The regulation is a contract imposing emission limits whenever a trip is a valid RDE test. We split our formalization into two main parts where the first part decides whether a trip qualifies as a valid test and the other assesses whether the emission limits are violated.

We use a trigger that indicates an RDE violation

```
trigger is_valid_test & emission_limits_exceeded
```

when the trip is a valid test but the emissions are exceeded. It remains to define boolean streams indicating a valid test and exceeded emissions.

As we will see our extension of Lola provides a very intuitive and natural way of formalizing the RDE test procedure. The specification is structured as follows: We first declare input streams for all test parameters, we then formalize the various preconditions of the regulation to determine the validity of a trip. Finally, we formalize the computation of the distance specific emissions and whether the respective emission limits are exceeded, or not. We describe selected and interesting parts of the formalization[1] in this section.

5.1 Test Parameters

As the test parameters are provided as synchronous streams with a fixed sampling frequency f prescribed by the regulation, we can directly declare them as inputs to our monitor. To asses whether a trip meets basic requirements regarding its route and ambient conditions, we need the speed `v` of the vehicle, the `altitude`, and the ambient `temperature`. To calculate the emissions, we further need the concentration of the various regulated emission gasses and the *Exhaust Mass Flow* (EMF), i.e., the weight of the exhaust emitted per second. For this, we declare an input stream *gas*`_ppm` for each regulated emission gas and another input stream `exhaust_mass_flow` for the EMF. Given the gas concentration and the EMF we can compute the mass flow of the respective gas. Usually, all inputs come from the *On-Board Diagnostics II* (OBD-II) [19] interface and the PEMS which includes a GPS tracker. Given an appropriately equipped vehicle, the NO_x concentration can be obtained via OBD-II. Since the early 2000s all new U.S. and European cars are equipped with an OBD-II port [18]. However, an NO_x sensor is not mandatory yet.

5.2 Preconditions

The preconditions a trip shall satisfy to qualify as a valid test are divided into *trip requirements*, stipulating basic requirements regarding, for instance, the route and the velocity, *ambient conditions*, stating acceptable temperature and altitude ranges, *overall trip dynamics*, encompassing the driving behavior, and *dynamic conditions*, accounting for road grade, weather, and other dynamic factors. While the trip requirements and ambient conditions are relatively straightforward to specify, the overall trip dynamics and dynamic conditions are more of a challenge.

Trip Requirements After declaring the input streams we formalize the trip requirements as specified in Section 6 of ANNEX IIIA of the RDE regulation. Compare Table 1 in Section 3 for an overview of the constraints and our full formalization for further details. To formalize the trip requirements, we first compute useful auxiliary streams e.g.

```
output bool  is_urban := v <= 60 // 6.3
output float u_avg_v  := avg(v[-N:0 | is_urban])
```

which we use to comprehensively assert the trip requirements. According to the regulation, values need to be binned according to the current speed in one of three bins, *urban*, *rural*, or *motorway*. The stream `u_avg_v` computes the average velocity in the urban speed bin using a sliding aggregation window. `N` is a constant denoting the maximal number of samples an RDE trip could have. An RDE trip must not last longer than 2h which is $\texttt{N} = 7200f$ samples. Computing the average only over that sampling interval instead of the whole data allows us to specify a monitor considering only the temporally maximal suffix of a trip. A trip with more than `N` samples is not a valid RDE trip in any case.

We use the auxiliary streams to compute a boolean stream which is the conjunction of all trip requirements, for instance, for the average velocity of the urban segment: `15 <= u_avg_v <= 40`.

Ambient Conditions The RDE regulation specifies the ambient conditions in terms of temperature (in Kelvin), e.g., `273 <= temperature <= 303`, and altitude ranges which can be directly translated to boolean formulae.

Overall Trip Dynamics The overall trip dynamics asses the drivers driving behavior. They require that the driver neither drives too aggressive nor too restrained. To this end, they require to compute the 95% percentile of speed times acceleration for acceleration values at least 0.1 for each speed bin. We show this exemplary for the urban speed bin:

```
output float a  := (v[+1,0] - v[-1,0]) / (2 * 3.6)
output float va := (v * a / 3.6)
output bool u_a_ge_01 := a >= 0.1 & is_urban
output float u_va_pct :=
  percentile95(va[-N:0 | u_a_ge_01])
```

These values can be efficiently computed with an update cost of $\mathcal{O}(\log n)$ and storage cost of $\mathcal{O}(n)$. Although, in general specifications with future references are not efficiently monitorable [10] this does not hold here, as `v` is extended in every step and cannot delay the computation indefinitely. We again use these values as part of boolean equations as specified in the RDE regulation, e.g.

```
u_va_pct > (0.136*u_avg_v + 14.44)
```

which invalidates the trip if less than 95% of the `va` values are below the given threshold, i.e., if the driving was too aggressive.

Dynamic Conditions The dynamic conditions encompass road grade, weather, and other factors that may influence the performance of the vehicle under test, but are out of control of the driver. They serve as built-in plausibility checks based on the reproducible CO_2 measurements of WLTP. To validate the dynamic conditions one considers variable width windows where a new window is

instantiated with each sample point. Owed to the parameterized stream templates of Lola 2.0 this can be expressed nicely by:

```
output bool win_completed(start: int) :
  inv: sample; ter: win_completed
  := total_co2_mass - win_start_co2(start) >= MCO2REF

output float win_v(start: int) :
  inv: sample; ter: win_completed := v

output float win_avg_v(start: int) :
  inv: sample; ter: win_completed
  := avg(win_v(start)[-N:0])
```

With each `sample` a new window is invoked. The values of these windows are extended in each step, and a window is completed in the step where the CO_2 emissions so far generated are at least a reference value determined by the WLTP test results. The RDE requires to compute the distance specific emissions for each window as well as the average speed. The earlier introduced sliding aggregation windows can be used to compute these values. Each window is then checked for normality by comparing the distance specific CO_2 emissions given the average speed to a reference curve. If at least 50% of windows are normal, the test is considered valid.

In the above, we are formalizing the computation of the values to demonstrate that we are indeed able to cover dynamic conditions faithfully with our formalization. However, our actual experiments do not include the checks regarding these conditions, which merely serve as plausibility check. The practical reason is that the needed WLTP values for our test vehicles were unavailable to us.

Calculating Emissions As already stated above the emissions are calculated using the exhaust mass flow and the gas concentrations. The emissions are then accumulated and based on the distance of the trip the distance specific emissions are calculated which are compared to the respective threshold [21], e.g.:

```
output bool nox_exceeded :=
  ite(d > 0, sum(D_nox_mass[-N:0]) / d, 0) > 0.08
```

5.3 RDE Violation

Given the boolean formulae of the preconditions, we compute whether the trip is indeed valid. Given the boolean formulae indicating whether the various regulated emission gases exceed the thresholds, we compute whether the emissions are exceeded. This gives us the boolean streams needed for our trigger.

Monitoring for RDE violations allows us to asses whether a car is compliant or not. In addition, one can ask the question of how difficult is it to detect

a running test as early as possible. Some of the preconditions, e.g., the total duration and the maximal velocity are monotonic and cannot be satisfied once violated. To build an RDE defeat device one needs to be able to tell whether the current trip prefix could still become a valid RDE trip or not.

6 Experimental Evaluation

We show that the formalization and the thereof constructed monitor are not only useful for certification purposes but can further be used by a layperson without expensive equipment to get an insight into exhaust emissions and whether her car does indeed adhere to the RDE regulation, or not.

Usually, the test parameters are obtained with a Portable Emissions Measurement System. We present two use cases for our monitor—one that requires a quite expensive PEMS and another that does not.

Case 1: Genuine RDE The first use case of the constructed RDE monitor is a genuine RDE test performed with a PEMS, for instance as part of an official certification process. The input streams for our monitor are directly generated by the PEMS and the control unit of the car.

Case 2: Low-Cost RDE The more interesting use case, however, is a *low-cost* RDE test without a PEMS. At best such a test can be conducted by a layperson without expensive equipment and expertise how to use it. The key challenge of a low-cost RDE is obtaining the test parameters.

A PEMS is a whole emission measurement laboratory in a box and therefore costs a significant amount of money. In addition, its setup procedure is rather complicated and usually requires an expert. Therefore, genuine RDE tests cannot be conducted by a layperson. As we will show, a low-cost variant of the RDE can be performed solely based on on-board sensors for a fraction of the cost with an easy to use monitor plugged into a standardized debug port.

The key challenge here is to obtain the test parameters—especially the concentrations of the regulated emission gasses. While the vehicle speed and altitude can be determined via GPS with an ordinary smartphone, measuring emissions requires specific sensors. Fortunately, many modern cars with a Selective Catalytic Reduction (SCR) system are already equipped with NO_x sensors measuring the NO_x concentration in the after-treatment exhaust stream, i.e., the stream of exhaust after it ran through the cleaning process as it leaves the tailpipe. Further, the CO_2 emissions can be approximated using data obtained from the engine control unit. Thanks to the standardized OBD-II interface [19] the required values can be obtained using a standard debug port.

We conducted a low-cost RDE with an Audi A7 3.0 TDI 200kW which is known to contain a defeat device which chokes the injected amount of urea shortly before it runs out of urea [24]. Urea is used as part of the SCR system to lower the NO_x emissions. We assume that the car indeed conforms with the EURO 6 emission limits in case the urea tank contains enough urea. If this were not the case, this would have been likely unrevealed by now based on the

extensive testing that was necessary to detect the defeat device in the first place. In order to convince ourselves that the RDE specification and monitor we provide is correct, we first checked the input validation by correctly validating recorded data of genuine RDE tests. For trips that obviously are not RDE, the monitor complained as expected. In a second step, we drove a valid RDE with the Audi A7 mentioned above with full urea tank. We briefly discuss the test setup, main obstacles, and the result of this test in the rest of this section.

While our test vehicle provides the NO_x concentration in the exhaust stream, it does not directly provide the exhaust mass flow, which is needed to calculate the emissions. We thus approximate the exhaust mass flow as follows: [23] describes a procedure to compute the exhaust mass flow based on the mass air flow and the mass fuel flow, i.e., the rate of mass of air and fuel used in the combustion process. We approximate the fuel mass flow based on the rate of fuel consumption in liters and the fuel density, which is approximately 0.835 kg/l [20] for Diesel. In Lola this is then calculated by:

```
output float exhaust_mass_flow :=  // in [kg/h]
  mass_air_flow + fuel_rate * 0.835
```

CO_2 emissions can be calculated based on the fuel rate and an oxidation factor specifying how much of the carbon is fully oxidized to CO_2. Cars do emit CO which is a regulated emission gas. Thus the oxidation factor has to be less than 1. For our tests, we assumed an oxidation factor of 0.99 [25].

Besides acquiring the test parameters, there is yet another challenge for a successful low-cost RDE conducted by a layperson—she needs to drive a valid RDE test trip. To assist with that, we augmented our specification with additional streams and triggers computing the urban, rural, and motorway distances which still need to be driven and other indicators, e.g., emitting a warning when the stop percentage comes close to the allowed boundaries.

With that assistance system in place, it was relatively easy to drive a valid test trip. See Figure 2 for the speed profile of the trip. Our monitor computed a value of 68 mg/km NO_x which is within the EURO 6 emission limits of 80 mg/km [21] and almost matches the value of the data-sheet [4], which is 67 mg/km. For CO_2 the monitor computed 151 g/km which is a deviation of +9% from the value of the datasheet, but CO_2 is only used to check plausibility in any case. This shows that a low-cost RDE test conducted by a layperson using inexpensive equipment can provide very good results.

Knowing that the vehicle chokes the urea injected into the exhaust stream whenever it runs low on urea, we tried to repeat our experiment with a close to empty urea tank. Unfortunately, we were unable to drain the urea from the tank of the car which is prevented by the construction of the vehicle.

To conclude the experimental evaluation and given those results, we envisage that in the future cars are equipped with more sophisticated emission measurement systems such that a low-cost RDE eventually becomes possible not only for NO_x but also for other emission gasses.

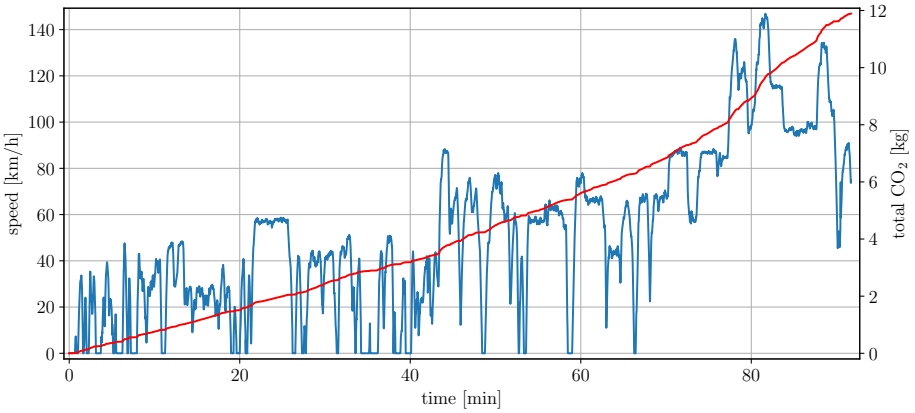


Fig. 2. Profile of a Low-Cost RDE with an Audi A7 3.0 TDI 200kW

7 Conclusion

We presented an extension of the stream-based specification language LOLA 2.0 with sliding aggregation windows and showcased its application with a formalization of the Real Driving Emissions (RDE) regulation. The constructed monitor has been successfully used as a basis for a low-cost, easy-to-use, and fully transparent tool, which can be plugged into the standardized OBD-II port with which every modern car is equipped. This enables laypersons to perform RDE tests for a fraction of the cost of a genuine RDE test. These measurements then rely on the on-board NO_x sensors. Research about their precision is still in progress. However, the tests we conducted suggest high precision measurements. Nevertheless, it should be mentioned that the sensors and their driver software are shipped with the car, so it is imaginable that the sensors are doped by the manufacturer, thereby invalidating the test results. We are indeed looking into the option to instead hook a separate sensor to the exhaust pipe.

The formalization of regulations is an essential step towards precise and succinct perspicuity enablers. The existing formalization already captures the heart of the RDE regulation. There are some corner cases and details for regions with peculiar geographic conditions that we did not implement yet, but we are planning to add. Our monitor is working *online*, i.e., the car's driving information is passed to the monitor in real-time. However, the decision whether the trip satisfies the RDE requirements, or not, does not consider possible continuations of the trip. Thus far, it does not detect that a current trip cannot be prolonged to a valid RDE drive anymore. We are working on this. Additionally, our goal is to enable RDE checks during normal usage of the car. To this end, we plan to integrate a detection algorithm identifying all intervals of a trip (longer than 90 minutes) satisfying the RDE conditions together with compliance checks concerning the emission thresholds. Such a monitor could, for instance, be integrated into an easy-to-use smartphone app, possibly paired with a wireless OBD-II dongle, or as a simple means to crowdsource an empirical answer to the question of how much of actual road traffic is covered by the RDE.

References

1. AVL M.O.V.E Real Driving Emission Testing, https://www.avl.com/emission-measurement/-/asset_publisher/gYjUpY19vEA8/content/avl-m-o-v-e-real-driving-emission-testing, accessed: 2018-07-06
2. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM **43**(1), 116–146 (1996)
3. Annpureddy, Y., Liu, C., Fainekos, G.E., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6605, pp. 254–257. Springer (2011)
4. Audi: Technische Daten - Audi A7 - 3.0 TDI 200 kW s-tronic quattro EU6W (March 2015)
5. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: Lola: Runtime monitoring of synchronous systems. In: 12th International Symposium on Temporal Representation and Reasoning (TIME'05). pp. 166–174. IEEE Computer Society Press (June 2005)
6. D'Argenio, P.R., Barthe, G., Biewer, S., Finkbeiner, B., Hermanns, H.: Is your software on dope? In: Yang, H. (ed.) Programming Languages and Systems. pp. 83–110. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)
7. 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)
8. Donzé, A., Maler, O., Bartocci, E., Nickovic, D., Grosu, R., Smolka, S.A.: On temporal logic and signal processing. In: Chakraborty, S., Mukund, M. (eds.) Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7561, pp. 92–106. Springer (2012)
9. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theor. Comput. Sci. **410**(42), 4262–4291 (2009)
10. Faymonville, P., Finkbeiner, B., Schirmer, S., Torfah, H.: A stream-based specification language for network monitoring. In: Falcone, Y., Sánchez, C. (eds.) Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10012, pp. 152–168. Springer (2016)
11. Faymonville, P., Finkbeiner, B., Schwenger, M., Torfah, H.: Real-time stream-based monitoring. CoRR **abs/1711.03829** (2017), http://arxiv.org/abs/1711.03829
12. Hirzel, M., Schneider, S., Tangwongsan, K.: Sliding-window aggregation algorithms: Tutorial. In: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems. pp. 11–14. DEBS '17, ACM, New York, NY, USA (2017)
13. 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)
14. Maler, O., Nickovic, D., Pnueli, A.: Checking temporal properties of discrete, timed and continuous behaviors. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday. Lecture Notes in Computer Science, vol. 4800, pp. 475–505. Springer (2008)
15. 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), http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4567914
16. Pnueli, A.: The temporal semantics of concurrent programs. Theor. Comput. Sci. **13**, 45–60 (1981)
17. Rizk, A., Batt, G., Fages, F., Soliman, S.: On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In: Heiner, M., Uhrmacher, A.M. (eds.) Computational Methods in Systems Biology, 6th International Conference, CMSB 2008, Rostock, Germany, October 12-15, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5307, pp. 251–268. Springer (2008)
18. The European Parliament and the Council of the European Union: 98/69/EC (October 1998), http://data.europa.eu/eli/dir/1998/69/oj
19. 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
20. The European Parliament and the Council of the European Union: Directive 2005/55/EC (September 2005), http://data.europa.eu/eli/dir/2005/55/oj
21. The European Parliament and the Council of the European Union: Commission Regulation (EU) 2007/715 (June 2007), http://data.europa.eu/eli/reg/2007/715/oj
22. The European Parliament and the Council of the European Union: Commission Regulation (EU) 2016/427 (March 2016), http://data.europa.eu/eli/reg/2016/427/oj
23. 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
24. Traufetter, G.: Audi manipulierte beliebtes Dienstwagenmodell - Produktion gestoppt (May 2018), http://www.spiegel.de/auto/aktuell/audi-manipulierte-beliebtes-dienstwagenmodell-a-1206722.html
25. U.S. Environmental Protection Agency: Emission Facts: Average Carbon Dioxide Emissions Resulting from Gasoline and Diesel Fuel (February 2005), https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=P1001YTF.TXT