Configurable-by-Construction Runtime Monitoring*

Author's version – December 18, 2022

Clemens Dubslaff[1] and Maximilian A. Köhl[2]

[1] Technische Universität Dresden, Dresden, Germany
clemens.dubslaff@tu-dresden.de
[2] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
koehl@cs.uni-saarland.de

Abstract. Most modern systems, be it cyber-physical or mere software systems, are highly configurable. The main challenge when dealing with such *configurable systems* stems from the usually huge number of system variants that can be exponential in the number of configuration options or features. Monitoring systems that react on observations, e.g., sensor data, varying across system configurations or being themselves configurable also face this challenge but have barely been considered in the literature. In this paper, we discuss new aspects for runtime monitoring with variability in the system being monitored as well as the monitor itself. As a first step towards a *configurable-by-construction* runtime monitoring approach, we introduce configurable monitors from an automata-theoretic and stream-based perspective. For this, we harvest existing work on featured transition systems and present a variability-aware variant of the stream-based specification language Lola.

1 Introduction

Modern cyber-physical systems such as cars, airplanes, or robots are highly configurable based on customer needs or through their inherent adaptivity to the environments in which they are operating. For example, cars may come with different driver assistance systems the customer payed for or robots adapt their behaviors depending on whether they operate in a machine-only or human-machine co-adaptive setting. Within software systems we also encounter configurability, e.g., in *software product lines* that can be configured through *features* as incremental or optional functionalities [32,2]. Often the configuration spaces are exponential in the number of configuration options or features, which renders the development, analysis, and explanation of such systems challenging tasks.

The manifold possibilities to configure these systems raise further challenges also in the runtime monitoring setting that have been barely addressed in the

* This work was partially supported by the DFG under the projects TRR 248 (see https://perspicuous-computing.science, project ID 389792660) and EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy).

existing literature. In this paper, we propose first steps towards a *configurable-by-construction* runtime monitoring approach that is inspired by automata-theoretic runtime verification [24,6], featured transition systems [11], and stream-based runtime monitoring [12,27]. Specifically, we (a) discuss challenges towards *configurable monitors* and variability-aware runtime verification in general, (b) present methods to synthesize concise *featured monitors* from specifications in a featured variant of linear temporal logic (LTL) [11], and (c) introduce a configurable variant of the stream-based specification language Lola [12] together with a family-based analysis for well-formedness and efficient monitorability that exploits commonalities across system variants. While there exist many different approaches to runtime monitoring and verification, automata-based and stream-based approaches are among the most prominent [24]. Hence, we concretize the idea of configurable-by-construction runtime monitoring for both of them.

Outline and contribution. This paper is structured as follows. We first discuss benefits and open challenges in a configurable runtime monitoring setting (Section 2). Then, we introduce an automata-based framework for configurable monitors and their synthesis from featured LTL specifications (Section 3) and extend the stream-based specification language Lola to the configurable setting (Section 4). These two orthogonal approaches are intentionally presented in self-contained sections to separate concerns and allow interested readers to choose the approach according to their needs. We close the paper by summarizing our findings and related work and provide an outlook on future work (Section 5).

2 The Quest of Configurable Runtime Monitoring

Considering monitors to be themselves configurable opens many new challenges. In this section, we first introduce a running example of a configurable system from the automotive systems domain before we discuss selected challenges and how they are addressed in this paper.

2.1 Feature-oriented Example

While conducting research on real-world runtime monitoring of vehicles [22,9], we encountered the need for monitor configurability. Inspired by this example, we here establish a running example on feature-oriented systems modeling.

Real driving emissions tests. It is well known that the real driving emissions (RDE) regulation, put in force by the European Union [29], can be cast into a runtime monitoring problem using existing stream-based runtime monitoring techniques [22]. An RDE test is supposed to evaluate the emissions of a vehicle under realistic conditions.[3] In recent work, we presented an Android application

[3] Similar test procedures exist for characteristics of electric vehicles, for instance, those issued by the United States Environmental Protection Agency [31].

[9] that enables laypersons to conduct lightweight RDE tests via the standard on-board diagnostic (OBD) interface – a component any modern car is required to be equipped with [28]. At its core, our application relies on the runtime monitoring framework RTLola [18]. As different cars usually have different fuel-consumption characteristics, are fully electric, or have different sensors, they provide different information via OBD. To this end, the RTLola specification needs to be adapted to each car based on the car's configuration. Currently, the required RTLola specifications for the different configurations are pieced together in an ad-hoc fashion [9].

For example, to compute the amount of emitted pollutants, such as nitrous oxide (NOx), the *exhaust mass flow* (EMF), i.e., the mass of exhaust emitted per time unit, must be known in addition to the relative concentration (in ppm) of the pollutant in the exhaust gas. While many modern diesel cars come equipped with sensors providing the relative concentration of nitrous oxide, the EMF is rarely provided directly via OBD due to the car not having of an EMF sensor. Fortunately, the EMF can be computed based on various other values such as the mass air flow (MAF) in combination with the fuel rate (FR) or the fuel-air equivalence (FAE) ratio.

Hence, the car is configurable by providing different sensors that can be used for the RDE test. For determining whether the RDE test itself passes, monitoring techniques might be used to establish a monitor observing the sensor data. However, in this case, both the system being monitored as well as the monitor itself should configurable as they depend on the cars sensor configuration.

Feature-oriented modeling. Variability in configurable systems can be abstractly described, e.g., using well-known concepts from feature-oriented system development [2]. Here, configuration options are encapsulated in *features* as optional or incremental units of functionality [32] where each set of features stands for a system *configuration*. Not all configurations are feasible in practice: For instance, any car without an EMF sensor or MAF sensor cannot be used for RDE tests since then, the EMF cannot be obtained by neither sensing nor computation. Such constraints on feasible system configurations are usually specified by *feature diagrams* [20] to describe the set of *valid configurations*. Feature diagrams are hierarchical diagrams over features where the parents of features have to be also included in the configuration containing the respective feature. A feature diagram for our RDE example is shown in Fig. 1. Here, if the FR feature is included in the system variant, this requires feature EMFc to be included as well.[4] Branching connectives in feature diagrams might also impose constraints on children, e.g., any configuration that has the RDE feature is also required to include at least one of the features EMF, NOxs, or COs. Differently, the EMFc feature requires both, the MAFs feature and the FR feature to be included towards a valid configuration. In total, there are $3 + 4{\cdot}(1 + 2{\cdot}3) = 31$ different

[4] For brevity, we shorten the feature names for computation and sensor features with tailing c and s, respectively.

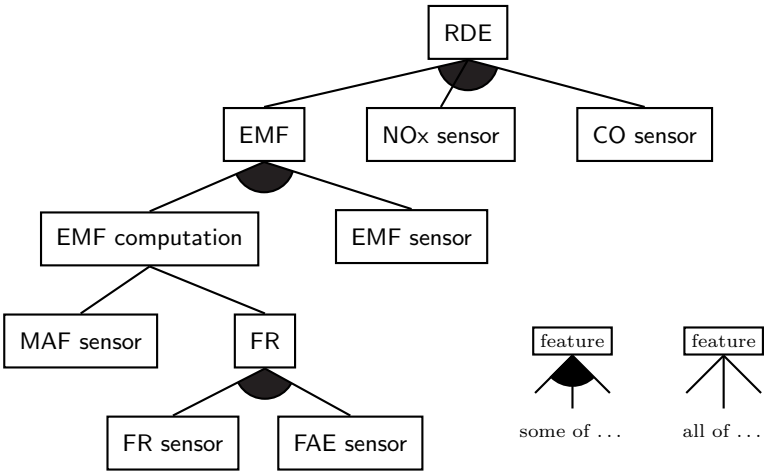

Fig. 1. Feature diagram of the configurable RDE system.

system variants, e.g., $\{\mathsf{RDE}, \mathsf{NOxs}\}$ and $\{\mathsf{RDE}, \mathsf{EMF}, \mathsf{EMFc}, \mathsf{MAFs}, \mathsf{FAEs}\}$, but not $\{\mathsf{RDE}, \mathsf{NOxs}, \mathsf{EMFc}\}$.

2.2 Challenges

While configurable system design and analysis is a well-established area, e.g., in feature-oriented software development, the link to runtime monitoring techniques is still missing. Indeed, the example of Section 2.1 demonstrates that there is a need for specification mechanisms that treat configurations as first-class citizens. It should be clear that similar considerations apply to other scenarios where variability plays a key role, e.g., monitoring of computer networks or microservice-based cloud applications where reconfiguration and scaling happens on a rather frequent basis.

Towards our configurable-by-construction approach to runtime monitoring that establishes the aforementioned link, we now identify five challenges that are to be addressed:

(1) Featured specification languages: As a first step, variability-aware specification languages for runtime monitoring and verification have to be developed or existing languages have to be adapted to account for variability. Such languages should yield *monitor families*, i.e., sets of monitors for different configurations, and may in addition offer support for more advanced scenarios such as online reconfiguration without disrupting any ongoing monitoring.

(2) Family-based monitoring: Given a featured specification, it has to be possible to decide properties of this specification over all possible configurations

without suffering from the exponential blowup in the number of features. In particular, in case a monitor has to be configured at runtime, as in Section 2.1, separate monitors for each valid configuration are often infeasible. Similar as for family-based analysis approaches, where an *all-in-one* analysis on a family model avoids the combinatorial blowup that arises within a *one-by-one* analysis of each configuration [30], a solution to this challenge could be to aim towards family monitors that are configurable and establish all-in-one family-based monitoring.

(3) Synthesis of monitors: For arriving at a monitor we have to be able to effectively synthesize such from featured specifications. This applies to monitors for single configurations but also to family monitors to enable a family-based monitoring approach as described in (2). It might be also possible that for certain configurations no monitor can be synthesized. The challenge is here to efficiently check *well-formedness* of a configuration, i.e., whether a monitor can be synthesized for the configuration, and to determine all well-formed configurations for which a family monitor can be synthesized. Another interesting question would be: Can we give an upper bound on the required memory and computation time of each single configuration monitor or family monitor?

(4) Matching monitor and system configurations: Configurable monitors can provide monitors with different functionalities depending on the contexts of the monitor or to enable reconfigurations in the case of family monitors. However, their primary use case might be monitoring systems that are themselves configurable. In Section 2.1 configurable monitors are required to adapt to the cars configuration of sensors on which the monitor bases its verdict of passing the RDE test. For such cases, mechanisms that match system and monitor configurations have to be developed.

(5) Suitable monitor configurations: Monitor configurations may also be more *suitable* than others [5]. Returning to Section 2.1, the EMF sensor might be preferred to EMF computation, as it is presumably more precise than a value computed based on other sensor data. Then, the configuration of monitors turns to an optimization problem: What is the most suitable monitor configuration given a requirement, e.g., the matching to system configurations as of (4)?

2.3 This paper

Exhaustively addressing all of the challenges raised above goes beyond the scope of this paper. In the following, we tackle some aspects towards configurable monitors by presenting an automata-based and a stream-based approach.

The automata-based approach is used to enhance standard LTL monitorability techniques [24] to the feature-oriented setting, presenting *featured monitors* as formalism to specify family monitors (addressing challenge (2)) and a synthesis algorithm towards monitors for properties in featured LTL [11] (3).

Our stream-based approach relies on the specification language Lola[5], for which we introduce a configurable variant (1). We present family-based analysis algorithms for well-formedness (3) [12] and efficient-monitorability of configurable Lola specifications that exploits commonalities between configurations and thus mitigates the exponential blowup in the number of features (2).

3 Configurable Automata-Based Monitoring

In this section, we tackle the problem of constructing *featured monitors* by means of an automata-theoretic approach to runtime monitors [7,24] that are configurable through features.

3.1 Preliminaries

For a finite set X we denote by X^* and X^ω the sets of finite and infinite sequences of elements in X, respectively, and by $\varepsilon \in X^*$ the empty sequence.

Propositional logic. By $\mathbb{B}(X)$ we denote the set of *Boolean expressions* over X, given by the grammar $\phi ::= \texttt{true} \mid x \mid \neg\phi \mid \phi \wedge \phi$ where variables x range over X. We use well-known Boolean connectives such as $\vee$, $\rightarrow$, etc. from which a Boolean expression can be easily obtained using standard syntactic transformations such as De Morgan's rule. Further, we define $\texttt{false} = \neg\texttt{true}$. The satisfaction relation $\vDash \subseteq \wp(X) \times \mathbb{B}(X)$ is defined in the usual way, where for $Y \subseteq X$ and $\phi \in \mathbb{B}(X)$ we have $Y \vDash \phi$ if ϕ evaluates to $\texttt{true}$ when all variables in Y are assumed to be $\texttt{true}$ and all variables in $X \setminus Y$ are $\texttt{false}$, respectively. We denote by $[\![\phi]\!] = \{Y \subseteq X \mid Y \vDash \phi\}$ the set of all ϕ-satisfying sets. With $\chi(\phi) \in \mathbb{B}(X)$ for a formula $\phi \in \mathbb{B}(X)$ we identify a *characteristic formula* that is unique modulo ϕ-satisfaction, i.e., for all $\phi, \phi' \in \mathbb{B}(X)$ with $[\![\phi]\!] = [\![\phi']\!]$ we have $\chi(\phi) = \chi(\phi')$.[6]

Linear temporal logic. To specify temporal properties, we rely on *linear temporal logic (LTL)* [26]. An *LTL formula* over X is given by the grammar

$$\varphi \quad ::= \quad \texttt{true} \mid x \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi$$

where x ranges over X. Basically, an LTL formula is a Boolean expression and we essentially use the same derived notations but with including the next operator X and the until operator U. The *size* $|\varphi|$ of an LTL formula φ is defined as the number of contained operators. An infinite sequence $\pi \in X^\omega$ with $\pi = \pi_0\pi_1\ldots$ *satisfies* an LTL formula φ, written $\pi \vDash \varphi$, if either $\varphi = \texttt{true}$, $\varphi = x$ and $\pi_0 = x$, $\varphi = \neg\psi$ and $\pi \nvDash \psi$, $\varphi = \psi \wedge \psi'$ and $\pi \vDash \psi$ as well as $\pi \vDash \psi'$, $\varphi = \mathsf{X}\psi$ and $\pi_1\pi_2\ldots \vDash \psi$, and $\varphi = \psi \mathsf{U} \psi'$ and there is a $k \in \mathbb{N}$ such that $\pi_k\pi_{k+1}\ldots \vDash \psi'$ and $\pi_i\pi_{i+1}\ldots \vDash \psi$ for all $i < k$.

[5] This applies to all variants of Lola, including the original variant [12], Lola 2.0 [17], and RTLola [18]. For simplicity, we use "Lola" as an umbrella term here.

[6] A canonical candidate for $\chi(\phi)$ would be the disjunctive normal form $\chi(\phi) = \bigvee_{Y \in [\![\phi]\!]} \left(\bigwedge_{x \in Y} x \wedge \bigwedge_{x \in X \setminus Y} \neg x\right)$, but also any other uniquely chosen formula, e.g., focusing on small lengths, would be suitable.

Monitors. We loosely follow definitions for automata-theoretic runtime monitoring according to Bauer et al. [7].

Definition 1 (Monitor). *A monitor is a tuple* $\mathcal{A} = (Q, \Sigma, \delta, \iota)$ *where* Q *is a finite set of control states containing* verdict states $\top, \bot \in Q$, Σ *is a finite input alphabet,* $\delta\colon Q \times \Sigma \to Q$ *is a transition function where* $\delta(p, \alpha) = p$ *for all* $p \in \{\top, \bot\}$ *and* $\alpha \in \Sigma$, *and* $\iota \in Q$ *is an initial state.*

Intuitively, a monitor is a deterministic finite state automaton that contains two accepting states $V = \{\top, \bot\}$ that are traps and formalize verdicts $\top$ and $\bot$. For $\pi \in \Sigma^*$ with $\pi = \pi_0\pi_1 \dots \pi_n$ we write $\delta(q, \pi)$ for $\delta(\delta(q, \pi_0), \pi_1 \dots \pi_n)$, i.e., the state reached after consuming π, where $\delta(q, \varepsilon) = q$.

Definition 2 (Monitor for LTL). *Let* φ *be an LTL formula over* Σ. *A monitor* $\mathcal{A} = (Q, \Sigma, \delta, \iota)$ *is a* monitor for φ *if for all* $\pi \in \Sigma^*$

$$\delta(\iota, \pi) = \top \;\; \textit{iff} \;\; \pi\rho \vDash \varphi \textit{ for all } \rho \in \Sigma^\omega \textit{ and}$$
$$\delta(\iota, \pi) = \bot \;\; \textit{iff} \;\; \pi\rho \nvDash \varphi \textit{ for all } \rho \in \Sigma^\omega.$$

Intuitively, a monitor for an LTL formula φ observes a sequence π and yields a verdict $\top$ or $\bot$ if φ is surely satisfied or not satisfied, respectively, w.r.t. to all extensions of π. Conversely, if no verdict state is reached via π, a final decision about satisfaction of φ cannot be drawn.

Theorem 1 ([7]). *For any LTL formula* φ *over* Σ *there is a monitor* $\mathcal{A}^\varphi$ *for* φ *with a state space of size* $\mathcal{O}(2^{2^{|\varphi|}})$.

Feature-oriented systems. We denote the set of abstract features by F and say that any set $C \subseteq F$ is a *configuration*. To formally describe behaviors of a configurable system family, *featured transition systems (FTSs)* [11] enhance transition systems by *feature guards*, i.e., Boolean expressions over F that are annotated to transitions. Formally, an *FTS* is a tuple $\mathcal{T} = (S, F, \Sigma, \Delta, I)$ where S, F, and Σ are finite sets of states, features, and actions, respectively, $\Delta \subseteq S \times \mathbb{B}(F) \times \Sigma \times S$ is a featured transition relation, and $I \subseteq \mathbb{B}(F) \times S$ is a set of featured initial states. The *projection* of $\mathcal{T}$ onto a configuration $C \subseteq F$ is the transition system $\mathcal{T}|_C = (S, \Sigma, \Delta|_C, I|_C)$ where $I|_C$ is the set of initial states ι for which there is $(\phi, \iota) \in I$ with $C \vDash \phi$ and $\Delta|_C \subseteq S \times \Sigma \times S$ is the smallest transition relation that satisfies

$$\frac{(s, \phi, \alpha, t) \in \Delta \quad C \vDash \phi}{(s, \alpha, t) \in \Delta|_C}$$

The semantics of an FTS $\mathcal{T}$ is thus a family of transition systems $\{\mathcal{T}|_C \mid C \subseteq F\}$.[7] To specify variability-aware properties, e.g., for FTSs, *featured linear temporal logic (fLTL)* has been defined as featured extension of LTL [11]. Formulas

[7] Note that behaviors for invalid feature configurations can be specified through non-satisfying feature guards on initial states, leading to empty initial state projections.

in fLTL over a feature domain F and an alphabet Σ are of the form $[\phi]\varphi$ where $\phi \in \mathbb{B}(F)$ is a feature guard over F and φ is an LTL formula over Σ. For a set Φ of fLTL formulas, we define $\Phi|_C = \bigwedge_{[\phi]\varphi \in \Phi, C \vDash \phi} \varphi$ as the LTL formula that is effective in a feature configuration $C \subseteq F$.

Example 1. Let us specify properties for the configurable RDE system example (see Section 2.1). In case the EMF sensor is present, a desirable property is that no computation of the EMF is triggered, as the EMF should be directly obtained from the EMF sensor. This could be expressed, e.g., by an fLTL formula

$$\psi_0 \;=\; [\mathsf{EMFs}]\neg\Diamond \mathit{comp\text{-}EMF}.$$

Another example would be the property that if the EMF computation feature is included and no EMF sensor is present, then after the MAF sensor has been read, in the next step the EMF must be computed. This is expressed by

$$\psi_1 \;=\; [\mathsf{EMFc} \wedge \neg\mathsf{EMFs}]\Box(\mathit{use\text{-}MAF\text{-}sensor} \rightarrow \mathsf{X}\mathit{comp\text{-}EMF}).$$

◇

3.2 Featured Monitors

Combining the concepts of monitors and FTSs, we define *featured monitors*:

Definition 3 (Featured Monitor). *A* featured monitor *is an FTS* $\mathcal{M} = (Q, F, \Sigma, \Delta, I)$ *where* Q *contains verdict states* V *and where for all* $C \subseteq F$*:*

(i) $(q, \mathtt{true}, \alpha, q) \in \Delta$ *for all* $q \in V$ *and* $\alpha \in \Sigma$
(ii) for all $p \in Q$ *and* $\alpha \in \Sigma$ *there is exactly one* $(p, \phi, \alpha, q) \in \Delta$ *with* $C \vDash \phi$
(iii) there is exactly one $(\phi, \iota) \in I$ *with* $C \vDash \phi$

Note that any projection $\mathcal{M}|_C$ of a featured monitor $\mathcal{M}$ onto a configuration $C \subseteq F$ is a monitor according to Definition 1 when interpreting $\Delta|_C$ as transition function and $I|_C$ as single state. This monitor is well-defined due to the uniqueness of C-satisfying transitions and initial states. Intuitively, a featured monitor $\mathcal{M}$ ensembles multiple monitors $\mathcal{M}|_C$ for feature configurations $C \subseteq F$.

To combine monitoring functionalities of featured monitors, we define a *featured monitor composition* by a product construction that adjusts feature guards and verdicts. For this, the *verdict composition* on some p and q is defined by

$$p \sqcap q \;=\; \begin{cases} \top & \text{if } p = \top \text{ and } q = \top \\ \bot & \text{if } p = \bot \text{ or } q = \bot \\ \langle p, q\rangle & \text{otherwise.} \end{cases}$$

Definition 4 (Featured Monitor Composition). *Let* $\mathcal{M}_i = (Q_i, F, \Sigma, \Delta_i, I_i)$ *for* $i \in \{0, 1\}$ *be two featured monitors over a common feature domain* F *and alphabet* Σ*. The* composition *of* $\mathcal{M}_0$ *and* $\mathcal{M}_1$ *is defined by* $\mathcal{M}_0 \sqcap \mathcal{M}_1 = (Q, F, \Sigma, \Delta, I)$ *where*

- $Q = V \cup (Q_0 \times Q_1)$
- $\Delta \subseteq Q \times \mathbb{B}(F) \times \Sigma \times Q$ *is the smallest featured transition relation such that*

$$\frac{(p_0, \phi_0, \alpha, q_0) \in \Delta_0 \quad (p_1, \phi_1, \alpha, q_1) \in \Delta_1 \quad p_0 \sqcap p_1 \notin V}{\big(p_0 \sqcap p_1, \chi(\phi_0 \wedge \phi_1), \alpha, q_0 \sqcap q_1\big) \in \Delta} \qquad \frac{p \in V \quad \alpha \in \Sigma}{(p, \mathtt{true}, \alpha, p) \in \Delta}$$

- $I = \big\{\big(\chi(\phi_0 \wedge \phi_1), \iota_0 \sqcap \iota_1\big) \mid (\phi_0, \iota_0) \in I_0, (\phi_1, \iota_1) \in I_1\big\}$.

Recall that $\chi(\phi)$ denotes a uniquely defined ϕ-equivalent Boolean expression and that $V = \{\top, \bot\}$ is the set of verdict states.

Lemma 1. *$\mathcal{M}_0 \sqcap \mathcal{M}_1$ is a featured monitor for any two featured monitors $\mathcal{M}_0$ and $\mathcal{M}_1$ over a common feature domain and alphabet.*

3.3 Synthesizing Featured Monitors

We use formulas in fLTL to specify variability-aware monitoring properties and describe how to synthesize featured monitors. Recall that for a set Φ of fLTL formulas, we define $\Phi|_C = \bigwedge_{[\phi]\varphi \in \Phi, C \vDash \phi} \varphi$ as the LTL formula that is effective in a feature configuration $C \subseteq F$.

Definition 5 (Featured Monitor for fLTL). *Let Φ be a finite set of fLTL properties over F and Σ. A* featured monitor for Φ *is a featured monitor $\mathcal{M}$ over F and Σ such that $\mathcal{M}|_C$ is a monitor for $\Phi|_C$ for all configurations $C \subseteq F$.*

Thus, a featured monitor for a set of fLTL formulas Φ provides a concise representation of a family of monitors that verdicts LTL properties for configurations to be jointly satisfied.

Let now $[\phi]\varphi$ be an fLTL formula over the feature domain F and an alphabet Σ. Further, let $\mathcal{A}^\varphi = (Q, \Sigma, \delta, \iota)$ be the monitor for φ according to Theorem 1. We then define the featured monitor $\mathcal{M}^\varphi_\phi = (Q, F, \Sigma, \Delta, I)$ by

$$\Delta \;=\; \big\{\; \big(q, \chi(\phi), \alpha, \delta(q, \alpha)\big), \big(q, \chi(\neg\phi), \alpha, q\big) \;\mid\; q \in Q, \alpha \in \Sigma \;\big\}$$

and $I = \{(\chi(\phi), \iota), (\chi(\neg\phi), \top)\}$. Note that $\mathcal{M}^\varphi_\phi$ is indeed a featured monitor for $[\phi]\varphi$ since for all configurations $C \subseteq F$ with $C \vDash \phi$ we have that $\mathcal{M}^\varphi_\phi|_C = \mathcal{A}^\varphi$ and $\mathcal{A}^\varphi$ is a monitor for φ.

Theorem 2. *For any set of fLTL formulas Φ, we have that $\mathcal{M}^\Phi = \bigsqcap_{[\phi]\varphi \in \Phi} \mathcal{M}^\varphi_\phi$ is a featured monitor for Φ.*[8]

Example 2. Let us describe how to synthesize a featured monitor from the set of fLTL formulas $\Phi = \{\psi_0, \psi_1\}$ from Example 1. First, we construct monitors for ψ_0 and ψ_1 according to Theorem 1 and, following the construction for $\mathcal{M}^\varphi_\phi$

[8] The corner case where $|\Phi| = 0$ is covered by the $\top$-verdicting featured monitor $\mathcal{M}^{\mathtt{true}}_{\mathtt{true}}$ that arises from $\mathcal{A}^{\mathtt{true}} = (V, \Sigma, \delta, \top)$ where $\delta(p, \alpha) = p$ for all $p \in V$ and $\alpha \in \Sigma$.

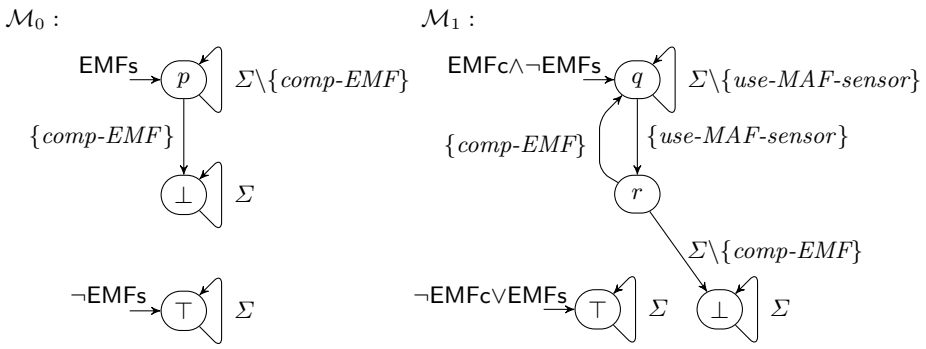

Fig. 2. Synthesized featured monitors $\mathcal{M}_0$ and $\mathcal{M}_1$ for ψ_0 and ψ_1, respectively

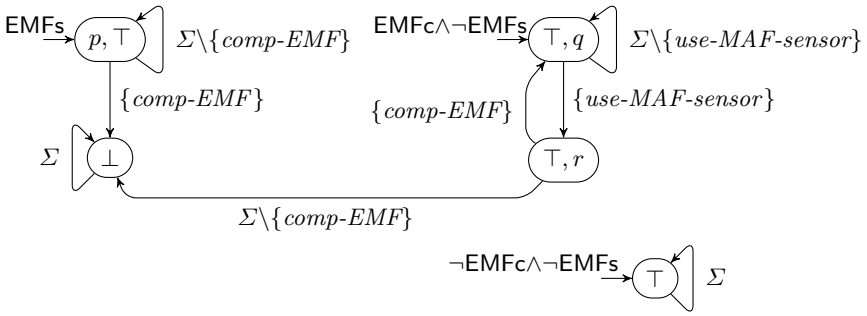

Fig. 3. Featured monitor composition $\mathcal{M}^{\Phi} = \mathcal{M}_0 \sqcap \mathcal{M}_1$ for $\Phi = \{\psi_0, \psi_1\}$

above, include feature guards towards $\mathcal{M}_0$ and $\mathcal{M}_1$. In Fig. 2 we depict both $\mathcal{M}_0$ and $\mathcal{M}_1$. For brevity, we only show the reachable parts where feature guards are satisfiable, annotated the transitions with the set of possible actions for the transition, and only indicated the feature guard once in initial states.

In Fig. 3 we show the featured monitor $\mathcal{M}^{\Phi} = \mathcal{M}_0 \sqcap \mathcal{M}_1$ obtained by the construction of Theorem 2. Note that there is also an initial state $(\texttt{false}, \langle p, q\rangle)$ not depicted in Fig. 3 due to the unsatisfiable feature guard. The featured monitor $\mathcal{M}^{\Phi}$ as of Fig. 3 provides a non-trivial $\bot$-verdict and only a trivial $\top$-verdict for configurations that do not have EMF computation nor sensor. This configuration can be considered as non-valid, since then, no RDE test is possible (see Section 2.1). However, there are also properties that have both, non-trivial $\bot$- and $\top$-verdicts: Consider the property of passing the NOx emission test in an RDE setup, i.e., that within 60 and 120 minutes, a valid RDE situation arises and the NOx emissions are below a threshold of 80 mg/km. To specify this property in fLTL, observe that time intervals can be expressed in LTL by chains of X-operators, e.g., $\Diamond^{[2,4]}\varphi$ stands for the formula $\mathsf{XX}(\varphi \vee \mathsf{X}(\varphi \vee \mathsf{X}\varphi))$ specifying φ to hold within 2 and 4 time units. With this at hand, the fLTL property for the

NOx passing RDE test could be specified by

$$\psi^{\text{NOx}}_{\text{passed}} \quad = \quad [\mathsf{NOxs}]\Diamond^{[60,120]}(\mathit{valid\text{-}RDE} \wedge \mathit{NOx\text{-}emission} < 80).$$

If the test is valid within the time interval and the emissions are below the threshold, then the featured monitor verdicts $\top$, while when passing the upper time bound of 120 without reaching a valid RDE state where the emissions are below the threshold, the featured monitor verdicts $\bot$. $\diamond$

3.4 Concise Featured Monitors

In the last sections, we introduced featured monitors and provided a method to synthesize such from fLTL specifications. To keep the presentation clean, we disregarded several technicalities that, however, would increase feasibility of the approach. First, our definitions always range over all possible configurations $C \subseteq F$ but could well be defined to range over valid configurations only. Second, our synthesis construction could be enhanced by reduction steps to obtain more concise featured monitors. For instance, we could join featured transitions (p, ϕ, α, q) and (p, ϕ', α, q) towards $(p, \chi(\phi \vee \phi'), \alpha, q)$ as well as featured initial states (ϕ, ι) and (ϕ', ι) towards $(\chi(\phi \vee \phi'), \iota)$. Transforming featured monitors into the related formalism of FTSs, we can directly apply dedicated bisimulation techniques [8] towards a compact featured monitor. For this, we label the verdict states $\top$ and $\bot$ by corresponding atomic propositions $\top$ and $\bot$, respectively, and keep all other states not labeled at all. Featured bisimulation is then achieved w.r.t. verdicts, preserving the featured monitor properties.

4 Configurable Stream-Based Monitoring

As discussed in Section 2, there is a need for featured specification languages for runtime monitoring. We now address this challenge by presenting an extension of the stream-based specification language Lola [12]. For simplicity, we will do so for the original variant of Lola [12]. Our extension is orthogonal to the extensions introduced by Lola 2.0 [17] and RTLola [18]. It should be straightforward to use the same techniques to introduce variability to those extensions.

We first restate the syntax and semantics of Lola as well as the central theorem necessary for constructing monitors out of Lola specifications. As our contribution, we then (a) introduce a notion of composability of Lola specifications, (b) use this notion to define *configurable Lola specifications*, and (c) present a family-based analysis for configurable Lola specifications ensuring that a monitor can be constructed for each configuration of interest and that the amount of memory it requires is independent of the length of its input.

4.1 Preliminaries

Let's start by introducing the stream-based specification language Lola.

Definition 6 (Lola Specification [12][9]). *Let $\mathcal{T}$ be a set of* data types *and $\mathbb{S}$ be a set of* typed stream variables*, i.e., each $s \in \mathbb{S}$ has an associated type $T_s \in \mathcal{T}$. A* Lola specification *over $\mathcal{T}$ and $\mathbb{S}$ is a partial function $f\colon \mathbb{S} \rightharpoonup \mathbb{E}$ assigning stream variables to* typed stream expressions*. The set of typed stream expressions $\mathbb{E}$ is defined inductively by:*

(i) Constants and stream variables of type T are expressions of type T.
(ii) Let $g\colon T_1 \times \cdots \times T_k \to T$ be a k-ary operator and $\eta_1, \ldots, \eta_k$ be expressions of type $T_1, \ldots, T_k$, then $g(\eta_1, \ldots, \eta_k)$ is an expression of type T.
(iii) Let η be a Boolean expression and η_1 and η_2 be expressions of some type T, then $\mathrm{ite}(\eta, \eta_1, \eta_2)$ is an expression of type T.
(iv) Let s be a stream variable of type T, c be a constant of type T, and $z \in \mathbb{Z}$, then $s[z, c]$ is a stream expression of type T.

A Lola specification f defines stream expressions for the set $\mathrm{dom}(f) \subseteq \mathbb{S}$ of stream variables. Those variables are coined *dependent* while the remaining variables are coined *independent*. The idea is that the stream expressions constrain the values the dependent variables can have at certain points in time.

The semantics of Lola is defined in terms of *evaluation models*. An evaluation model σ of length N assigns a sequence $\sigma(s) = \sigma_1(s) \cdots \sigma_N(s)$ of values of type T_s, coined a *stream*, to each stream variable $s \in \mathbb{S}$. Note that the individual streams all have the same length N. Given an evaluation model σ we inductively define an evaluation function $[\![\cdot]\!]_t$ for stream expressions at time t:

$$[\![c]\!]_t = c \qquad [\![s]\!]_t = \sigma_t(s) \qquad [\![g(\eta_1, \ldots, \eta_k)]\!]_t = g([\![\eta_1]\!]_t, \ldots, [\![\eta_k]\!]_t)$$

$$[\![\mathrm{ite}(\eta, \eta_1, \eta_2)]\!]_t = \begin{cases} [\![\eta_1]\!]_t & \text{if } [\![\eta]\!]_t = \texttt{true} \\ [\![\eta_2]\!]_t & \text{otherwise} \end{cases}$$

$$[\![s[z, c]]\!]_t = \begin{cases} \sigma_{t+z}(s) & \text{if } 1 \leq t + z \leq N \\ c & \text{otherwise} \end{cases}$$

An evaluation model σ *is consistent with* a Lola specification f, denoted by $\sigma \vDash f$, if and only if $[\![f(s)]\!]_t = \sigma_t(s)$ for all $s \in \mathrm{dom}(f)$ and $1 \leq t \leq N$. That is, the values of each dependent stream variable $s \in \mathrm{dom}(f)$ over time are consistent with the expressions specified by f.

Let us now have a look at a concrete example taken from Section 2.1. The RDE regulation stipulates that the acceleration a_t at time t shall be computed as follows [29, ANNEX IIIA, Appendix 7a, 3.1.2]:

$$a_t = (v_{t-1} + v_{t+1})/(2 \cdot 3.6)$$

[9] We slightly deviate from the original definition for notational convenience. In particular, we do not allow expressions of the form $\eta[z, c]$ where η is an arbitrary stream expression. It has been shown that those can be rewritten to $s'[z, c]$ by introducing an additional stream variable s' such that $f(s') = \eta$.

Hence, the acceleration at time t shall be computed using the velocity v at time $t-1$ and $t+1$. The translation into Lola is straightforward:

$$f(a) \;=\; \frac{v[-1,0] + v[+1,0]}{2 \cdot 3.6}$$

By using an offset expression of the form $s[z,c]$ we can access the value of the stream variable v in the past as well as the future. The default value c in an offset expression is used at time t if and only if the time $t+z$ lies outside of the streams provided by the evaluation model. Offset expressions are arguably the most innovative feature of Lola and responsible for its great expressive power. We refer to the original Lola paper for a detailed discussion [12].

Online monitoring and well-formedness. An online monitor for a Lola specification incrementally computes values for the dependent stream variables based on incrementally observed independent stream variables. In the following, we refer to the streams for the independent variables as *input streams* and to the streams for the dependent variables as *output streams*.

Using this terminology, an online monitor for a specification f computes output streams from input streams such that those streams together form an evaluation model consistent with the given specification. Being able to construct such a monitor is the primary purpose of a Lola specification.

In case of our example, assume that v is an independent variable, i.e., the velocity is provided as an input to the monitor, the monitor will then compute the stream for the acceleration a as defined by the RDE regulation.

Unfortunately, with the provided definitions, it might be impossible to compute output streams from input streams as no evaluation model may exist or there might be multiple such models for a given set of input streams leading to ambiguity [12]. To address this issue, a notion of well-definedness is introduced: A Lola specification is *well-defined* if and only if for any set of appropriately typed input streams, all of the same length, it has exactly one consistent evaluation model [12]. This restriction ensures that the monitor is well-defined.

From a practical perspective, however, well-definedness is difficult to deal with. Instead, the original paper [12] introduces a purely syntactic criterion called *well-formedness* such that the following central theorem holds:

Theorem 3 ([12]). *If a specification is well-formed, then it is well-defined.*

Now, well-formedness is defined by means of a *dependency graph*:

Definition 7 (Dependency Graph). *Let f be a Lola specification over the stream variables $\mathbb{S}$. The* dependency graph *for f is a directed and weighted multi-graph $G = \langle \mathbb{S}, E \rangle$ where E is the set of* edges*. An* edge *is a triple $\langle s_x, s_y, z \rangle$ where $s_x, s_y \in \mathbb{S}$ and $z \in \mathbb{Z}$. The set E of edges contains an edge $\langle s_x, s_y, z \rangle$ if and only if $s_x \in \mathrm{dom}(f)$ and the expression $f(s_x)$ contains an offset expression $s_y[z,c]$ for some constant c.*

Intuitively, the existence of an edge $\langle s_x, s_y, z\rangle$ in E records the fact that the stream for s_x depends on the stream for s_y with an offset of z. If there exists a cycle whose weights z sum up to zero, i.e., a *zero-weight cycle*, then the value at a given time circularly depends on the very same value leading to ambiguity issues. Well-formedness forbids precisely such viscous cycles.

Definition 8 (Well-Formedness). *A specification is* well-formed *if and only if its dependency graph does not contain any zero-weight cycle.*

By checking well-formedness of a specification, we make sure that a monitor for it is uniquely defined. It is this property of well-formedness that has to be checked in order to ensure that a monitor can be constructed using the techniques presented in the original Lola paper [12]. We refer to this paper for further details regarding the monitor construction and Theorem 3. Note that the *zero-weight cycle problem*, i.e., the problem of deciding whether a zero-weight cycle exists, is known to be NP-complete, which can be shown by a simple reduction from the NP-complete subset-sum problem [3, Theorem 3.12].

4.2 Configurable Lola

We now introduce an extension of Lola that yields monitor families and accounts for variability. To this end, we first define a notion of composition enabling the combination of multiple Lola specifications into one.

Definition 9 (Composition). *Two Lola specifications f_1 and f_2 are* composable *if and only if their dependent variables are disjoint. Formally that is $\mathrm{dom}(f_1) \cap \mathrm{dom}(f_2) = \emptyset$. For two composable specifications f_1 and f_2, we define their composition $f_1 \parallel f_2 : \mathrm{dom}(f_1) \cup \mathrm{dom}(f_2) \to \mathbb{E}$ as follows:*

$$(f_1 \parallel f_2)(s) := \begin{cases} f_1(s) & \text{if } s \in \mathrm{dom}(f_1) \\ f_2(s) & \text{if } s \in \mathrm{dom}(f_2) \end{cases}$$

Note that the composition $f_1 \parallel f_2$ is itself a Lola specification. The set of dependent variables of $f_1 \parallel f_2$ is $\mathrm{dom}(f_1) \cup \mathrm{dom}(f_2)$.

Without the restriction to composable specifications, the composition of two specifications would be problematic since the sets of dependent variables may overlap, thus, containing multiple potentially different equations for the same variable thereby leading to ambiguity.

Note that the composition operator $\parallel$ is associative and commutative. For that reason, the order of composition does not matter and any set of pairwise composable Lola specifications has a unique composition.

Leveraging this notion of composability, we now have all the tools to formally introduce *configurable Lola specifications*:

Definition 10 (Configurable Lola Specification). *A* configurable Lola specification *is a set $F = \{f_1, \ldots, f_m\}$ of Lola specifications, i.e., f_i is a Lola specification for each $1 \leq i \leq m$. We call the individual specifications f_i* features *of*

EMF sensor feature (EMFs):

```
input emf_sensor: Float64; // g/s
output emf = emf_sensor;
```

EMF computation feature (EMFc):

```
input maf_sensor: Float64; // g/s
input fuel_rate: Float64; // g/s
output emf = maf_sensor + fuel_rate;
```

FR sensor feature (FRs):

```
input fuel_rate_sensor: Float64; // g/s
output fuel_rate = fuel_rate_sensor;
```

FR computation feature (FRc):

```
input maf_sensor: Float64; // g/s
input fuel_air_equivalence_sensor: Float64; // ratio
output fuel_rate = maf_sensor / (14.5 * fuel_air_equivalence_sensor)
```

Fig. 4. Features extracted from the different RDE Lola specifications [9,22].

F. A configuration *C of F is a subset of F such that all features included in the subset are pairwise composable.*

By composition as defined in Definition 9, every configuration gives rise to a uniquely defined composite specification. In the following, we make use of this fact and simply treat configurations as if they are Lola specifications. In particular, we apply the concepts of well-formedness and efficient monitorability directly to configurations.

Featured Lola specifications make variability a first-class concept enabling the specification of configurable-by-construction runtime monitors. Returning to Section 2.1, we can now use different features for the different ways the EMF and fuel rate can be determined (see also Fig. 1). Fig. 4 shows four features, two for obtaining the EMF and two for computing the fuel rate. It is also a natural consequence of Definition 10 that the different ways to compute the EMF or fuel rate are mutually exclusive because those features overlap in their dependent stream variables (defined using the `output` keyword), i.e., they are not composable. Using a configurable Lola specification, those features can now systematically be combined as required.

Well-formedness of configurations. As with ordinary Lola specifications, we have to ensure that any configuration is well-formed or at least determine the configurations which are not. In particular, when reconfiguration at runtime is required, we have to ensure that any configuration of interest gives rise to a well-formed composite specification. Otherwise, at runtime, it might turn out impossible to construct a monitor for a certain configuration.

In case a feature is already not well-formed itself, we can conclude that any configuration containing this feature will also be not well-formed:

Lemma 2. *If any of the individual features is not well-formed, then any configuration containing the respective feature is not well-formed.*

This is easy to see as the dependency graph of the feature is a subgraph of the dependency graph of the configuration. Hence, any zero-weight cycle will also exist in the dependency graph of the configuration thereby rendering the configuration itself not well-formed.

Unfortunately, the reverse is not true, i.e., even if the individual features are all well-formed, this does not imply that any configuration is also well-formed. The dependency graph of a configuration combines those of the features and may thereby introduce new zero-weight cycles that are not present in the dependency graphs of any of the features when considered in isolation.

An obvious way to decide whether all configurations are well-formed would be to construct the specification for each configuration and then determine whether it is well-formed. Considering each configuration of a configurable Lola specification in isolation and checking its well-formedness is, in general, exponential in the number of features of that specification, thus rendering it infeasible as soon as the amount of features grows large. As we will see, using a family-based analysis for checking well-formedness prevents this blowup by exploiting commonalities between different configurations.

4.3 Family-Based Specification Analysis

For checking well-formedness and efficient monitorability for all configurations without an exponential blowup, we present a family-based analysis that exploits commonalities between configurations. Instead of using a dependency graph for each configuration in isolation, we construct a *family dependency graph*:

Definition 11 (Family Dependency Graph). *Let* $F = \{ f_1, \ldots, f_m \}$ *be a configurable Lola specification with* m *features over the stream variables* $\mathbb{S}$*. The* family dependency graph *for* F *is a directed, weighted, and feature-labeled multi-graph* $G = \langle \mathbb{S}, E \rangle$ *where* E *is the set of* edges*. An* edge *is a quadruple* $\langle s_x, s_y, z, i \rangle$ *where* $s_x, s_y \in \mathbb{S}$*,* $z \in \mathbb{Z}$*, and* $1 \leq i \leq m$*. The set* E *of edges contains an edge* $\langle s_x, s_y, z, i \rangle$ *if and only if* $s_x \in \mathrm{dom}(f_i)$ *and the expression* $f_i(s_x)$ *contains an expression* $s_y[z, c]$ *for some constant* c*.*

Intuitively, the existence of an edge $\langle s_x, s_y, z, i \rangle$ in E records the fact that the stream for s_x depends on the stream for s_y with an offset of z when activating the feature f_i. The family dependency graph is a superimposition of the dependency graphs of each individual feature with additional edge labels for the features they belong to. Hence, the following criterion is easily established:

Lemma 3. *If the family dependency graph of a configurable Lola specification does not contain a zero-weight cycle, then every configuration is well-formed.*

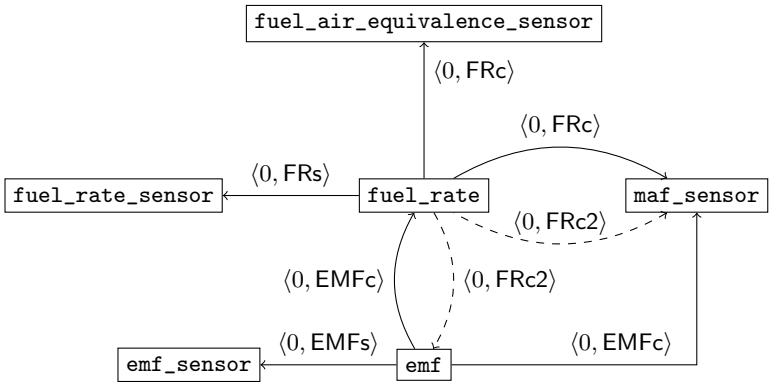

Fig. 5. Family dependency graph (without the dashed lines) for the configurable Lola specification in Fig. 4. The arrows are to be read as "depends on" with the given offset and feature. The dashed lines are with the additional FRc2 feature.

Clearly, the dependency graph of every configuration is a subgraph of the family dependency graph. Therefore, if the family dependency graph does not contain a zero-weight cycle then the dependency graphs of any individual configuration cannot contain such a cycle either.

Fig. 5 shows the family dependency graph (without the dashed lines) for the configurable Lola specification in Fig. 4. It does not contain any zero-weight cycles, in fact, it does not contain any cycles at all. Hence, all configurations for the configurable Lola specification are well-formed.

Lemma 3 gives us a sufficient criterion for well-formedness of every configuration. This criterion can be checked with the same algorithms and techniques as checking well-formedness of an individual specification. In contrast to the naive approach, which would consider each configuration individually, these algorithms can now exploit commonalities between the dependency graphs of the different configurations thereby mitigating the exponential blowup due to the often exponential number of configurations.

A Necessary Criterion. While Lemma 3 gives us a sufficient criterion for well-formedness of all configurations, this is actually not a necessary criterion. Intuitively, only those zero-weight cycles pose a problem that can actually arise from a configuration, i.e., a set of pairwise composable features. By adding this additional condition for cycles, we obtain the following theorem:

Theorem 4. *Every configuration is well-formed if and only if every zero-weight cycle in the family dependency graph contains at least two edges labeled with features which are not composable.*

The dependency graph of an individual configuration can be obtained from the family dependency graph by removing all edges corresponding to features which are not enabled. Now, if a zero-weight cycle in the family dependency

graph contains at least two edges labeled with features which are not composable, this cycle will not be included in a dependency graph of any of the configurations as a configuration can only contain features which are pairwise composable. If this is the case for all zero-weight cycles of the family dependency graph, then none of these cycles will be included in the dependency graph of any of the configurations. As a result, all configurations are well-formed. Conversely, if a zero-weight cycle exists which does not contain two edges that are labeled with non-composable features, then the respective features on this cycle can be composed to form a specification which is not well-formed.

Consider an extension of the configurable Lola specification in Fig. 4 with the following additional feature (FRc2) for computing the fuel rate:

```
input maf_sensor: Float64; // g/s
input emf: Float64; // g/s
output fuel_rate = emf - maf_sensor;
```

This introduces additional edges in the dependency graph (dashed edges in Fig. 5). With this feature, the family dependency graph now contains one elementary zero-weight cycle between `fuel_rate` and `emf`. Indeed, a configuration with both the FRc2 and EMFc feature is not well-formed. Enabling both features would mean that the `fuel_rate` should be computed based on the `emf` but at the same time the `emf` should be computed based on the `fuel_rate`. This cycle yields that a monitor is no longer well-defined. Note that FRc2 and EMFc are composable because they contain no overlapping definitions. With the family-based analysis this can be detected solely relying on the family dependency graph and without considering all $2^5 = 32$ configurations one-by-one.

Complexity. As the zero-weight cycle problem, the problem of finding a zero-weight cycle not containing two edges that are labeled with non-composable features is also NP-complete. It is more general than the zero-weight cycle problem because it contains an additional condition on cycles. It also lies in NP because it is easy to verify in polynomial time that a cycle is zero-weight and does not contain two edges that are labeled with non-composable features.

For practical purposes, it makes sense to collapse all edges $\langle s_x, s_y, z, i\rangle$ between the same vertices s_x and s_y that have the same weight z into a single edge $\langle s_x, s_y, z, I\rangle$ where I is the set of all feature labels found on any of these edges. This can drastically reduce the number of edges and thereby the number of potential zero-weight cycles to be considered.

Efficient monitorability. Recall that an additional property of Lola specifications that is of practical interest is *efficient monitorability*. Efficiently monitorable specifications are guaranteed to be monitorable with a bounded amount of memory independent of the length of the involved streams. A Lola specification is efficiently monitorable if and only if its dependency graph does not have positive cycles [12]. Our family-based analysis and Theorem 4 is easily extended to the question whether all configurations are efficiently monitorable:

Theorem 5. *Every configuration is efficiently monitorable if and only if every positive-weight cycle in the feature dependency graph contains at least two edges labeled with features which are not composable.*

Practical impact. Checking the family dependency graph instead of the dependency graph of each configuration in isolation can mitigate the exponential blowup in the number of features. At the same time, it enables analyzing a configurable Lola specification and makes sure that any configurations that appears in practice will indeed give rise to a well-formed and efficiently monitorable specification. This, in turn, means that a monitor can be synthesized and that its memory consumption will be bounded. Based on found zero-weight cycles, configurations that would not lead a well-formed specification can be identified ahead-of-time, providing a static guarantee for configurations at runtime.

We want to point out that Lola specifications can be parametrized and that parametrization can also be used for configurable monitors. However, while emulating features as we considered them with parameters and ite expressions is possible to some extent, the traditional well-formedness analysis will not understand that the different cases of "ite" are mutually exclusive. Thus, it would essentially correspond to a coarse-grained analysis according to Lemma 3. Instead, the analysis we propose here is more fine-grained. In addition, when emulating features using parameters, the independent variables of all features would be merged with no explicit distinction about which actually have to be provided and which are merely an encoding artifact. Thus, our work complements parameters offering a more fine-grained analysis and explicit treatment of features and independent variables. Together, parameters and features as we considered them make Lola a perfect fit for configurable-by-construction runtime verification.

5 Concluding Remarks

We presented initial concepts towards a *configurable-by-construction* approach for runtime monitoring, introducing *featured monitors* to serve the automata-theoretic view on runtime monitoring, and *configurable Lola specifications* that take on a stream-oriented view.

Related work. While we addressed some of the challenges for configurable-by-construction runtime verification, this is still a largely uncharted field. In the feature-oriented systems domain, Kim et al. [21] lift static analysis techniques to determine those feature configurations where safety properties could be violated and hence should be monitored during runtime. They address different problems than we consider in this paper, not considering variability in the monitor itself.

In addition to the Lola family [12,17,18] there are also other stream-based specification languages, in particular, TeSSLa [23] and Striver [19]. We expect our insights on configurability to at least partially carry over to them. In addition, there are also stream-based specification mechanisms based on automata theory [1] that provide the opportunity to consider the interplay between our notion of featured monitors and configurable Lola specifications.

Future work. Two notable dimensions that we did not consider yet in this paper concerns the question of uninterrupted online reconfiguration and the systematic matching of monitor and system configurations. For this future work, we could harvest results on reconfigurable systems analysis [14,13]. Monitors that take causality information on the system into account [4] could be also considered in the feature-oriented setting [16] and could trigger preemptive reconfigurations of features. Feature-oriented concepts can well be used to describe context-dependent systems [25,15] or role-based systems [10]. Thus, our runtime monitoring approach also enables monitoring systems to verdict contexts and roles entities play, e.g., whether a network device has the role of a server, client, or relay in different contexts.
To enhance the expressiveness of featured monitor specifications, our synthesis algorithm to obtain featured monitors from sets of fLTL formulas could surely be extended to arbitrary Boolean expressions over fLTL formulas. This requires the definition of further composition operators on verdicts and featured monitors, for which we solely defined the conjunctive counterpart $\sqcap$ in this paper.

References

1. Alur, R., Mamouras, K., Stanford, C.: Automata-based stream processing. In: Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP'17). Schloss Dagstuhl, Leibniz-Zentrum für Informatik (2017)
2. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines. Concepts and Implementation. Springer (2013)
3. Baier, C., Bertrand, N., Dubslaff, C., Gburek, D., Sankur, O.: Stochastic shortest paths and weight-bounded properties in Markov decision processes. In: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'18). pp. 86–94. ACM, New York, NY, USA (2018)
4. Baier, C., Dubslaff, C., Funke, F., Jantsch, S., Majumdar, R., Piribauer, J., Ziemek, R.: From verification to causality-based explications. In: Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP'21). LIPIcs, vol. 198, pp. 1:1–1:20. Leibniz-Zentrum für Informatik (2021)
5. Baier, C., Dubslaff, C., Hermanns, H., Klauck, M., Klüppelholz, S., Köhl, M.A.: Components in probabilistic systems: Suitable by construction. In: Proceedings of the 9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles (ISoLA'20). pp. 240–261. LNCS, Springer International Publishing, Cham (2020)
6. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. In: Lectures on Runtime Verification – Introductory and Advanced Topics, LNCS, vol. 10457, pp. 1–33. Springer (2018)
7. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Proceedings of the 26th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06). pp. 260–272. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
8. Belder, T., ter Beek, M.H., de Vink, E.P.: Coherent branching feature bisimulation. In: Proceedings 6th Workshop on Formal Methods and Analysis in SPL Engineering (FMSPLE@ETAPS'15). EPTCS, vol. 182, pp. 14–30 (2015)

9. Biewer, S., Finkbeiner, B., Hermanns, H., Köhl, M.A., Schnitzer, Y., Schwenger, M.: Rtlola on board: Testing real driving emissions on your phone. In: Proceedings of the 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'21). pp. 365–372. Springer International Publishing, Cham (2021)
10. Chrszon, P., Baier, C., Dubslaff, C., Klüppelholz, S.: From features to roles. In: Proceedings of the 24th ACM International Systems and Software Product Line Conference (SPLC'20). pp. 19:1–19:11. ACM (2020)
11. Classen, A., Cordy, M., Schobbens, P.Y., Heymans, P., Legay, A., Raskin, J.F.: Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. Transactions on Software Engineering **39**, 1069–1089 (2013)
12. 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: Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME'05). pp. 166–174. IEEE Computer Society Press (June 2005)
13. Dubslaff, C.: Quantitative Analysis of Configurable and Reconfigurable Systems. Ph.D. thesis, TU Dresden, Institute for Theoretical Computer Science (2021)
14. Dubslaff, C., Baier, C., Klüppelholz, S.: Probabilistic model checking for feature-oriented systems. Transactions on Aspect-Oriented Software Development **12**, 180–220 (2015)
15. Dubslaff, C., Koopmann, P., Turhan, A.Y.: Ontology-mediated probabilistic model checking. In: Proceedings of the 15th Conference on integrated Formal Methods (iFM'19). vol. LNCS:11918, pp. 194–211. Springer (2019)
16. Dubslaff, C., Weis, K., Baier, C., Apel, S.: Causality in configurable software systems. In: Proceedings of the 44th International Conference on Software Engineering (ICSE'22) (2022)
17. Faymonville, P., Finkbeiner, B., Schirmer, S., Torfah, H.: A stream-based specification language for network monitoring. In: Proceedings of the 16th International Conference on Runtime Verification (RV'16). LNCS, vol. 10012, pp. 152–168. Springer (2016)
18. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: Streamlab: Stream-based monitoring of cyber-physical systems. In: Proceedings of the 31st International Conference on Computer Aided Verification (CAV'19). LNCS, vol. 11561, pp. 421–431. Springer (2019)
19. Gorostiaga, F., Sánchez, C.: Striver: Stream runtime verification for real-time event-streams. In: Proceedings of the 18th International Conference on Runtime Verification (RV'18). pp. 282–298. Springer International Publishing, Cham (2018)
20. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon University Software Engineering Institute (1990)
21. Kim, C.H.P., Bodden, E., Batory, D., Khurshid, S.: Reducing configurations to monitor in a software product line. In: Proceedings of the 10th International Conference on Runtime Verification (RV'10). pp. 285–299. Springer, Berlin, Heidelberg (2010)
22. Köhl, M.A., Hermanns, H., Biewer, S.: Efficient monitoring of real driving emissions. In: Proceedings of the 18th International Conference on Runtime Verification (RV'18). pp. 299–315. Springer International Publishing, Cham (2018)

23. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: Tessla: Runtime verification of non-synchronized real-time streams. In: Proceedings of the 33rd ACM Symposium on Applied Computing (SAC'18). ACM, ACM, France (04/2018 2018)
24. Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming **78**(5), 293–303 (2009)
25. Mauro, J., Nieke, M., Seidl, C., Yu, I.C.: Context aware reconfiguration in software product lines. In: Proceedings of the 10th Workshop on Variability Modelling of Software-intensive Systems (VaMoS'16). pp. 41–48. ACM (2016)
26. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Symposium on Foundations of Computer Science (SFCS'77). pp. 46–57. IEEE (1977)
27. Sánchez, C.: Online and offline stream runtime verification of synchronous systems. In: Proceedings of the 18th International Conference on Runtime Verification (RV'18). pp. 138–163. Springer International Publishing, Cham (2018)
28. 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
29. 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
30. Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: A classification and survey of analysis strategies for software product lines. ACM Comput. Surv. **47**(1s), 6:1–6:45 (2014)
31. United States Environmental Protection Agency: https://www.epa.gov/greenvehicles/explaining-electric-plug-hybrid-electric-vehicles
32. Zave, P.: Feature-oriented description, formal methods, and dfc. In: Proceedings of the Workshop on Language Constructs for Describing Features. pp. 11–26. Springer (2001)