# Safety enforcement for autonomous driving on a simulated highway using `Asmeta` models@run.time

Andrea Bombarda[1][0000−0003−4244−9319], Silvia Bonfanti[1][0000−0001−9679−4551], Angelo Gargantini[1][0000−0002−4035−0131], Nico Pellegrinelli[1][0009−0000−4944−6845], and Patrizia Scandurra[1][0000−0002−9209−3624]

University of Bergamo, Bergamo, Italy `{andrea.bombarda, silvia.bonfanti, angelo.gargantini, nico.pellegrinelli, patrizia.scandurra}@unibg.it`

**Abstract.** Mission-critical systems, such as autonomous vehicles, operate in dynamic environments where unexpected events should be managed while guaranteeing safe behavior. Ensuring the safety of these complex systems is a major open challenge and requires robust mechanisms to enforce correct behavior during runtime. This paper illustrates a runtime safety enforcement framework for the *output sanitization* of an autonomous driving agent on a highway. The enforcement mechanism is based on a (formally validated and verified) `Asmeta` model representing the enforcement rules and used at run-time to eventually steer the driving agent to behave safely and avoid collisions. We demonstrate both efficacy and efficiency of the proposed enforcement approach by conducting an experimental evaluation. We connected our safety enforcer with the highway simulation environment and co-executed it with the pre-trained (unsafe) AI agents as provided by the ABZ 2025 case study. We consider the single-lane case with the safety requirement and one scenario of the multi-lane case about preferring the right-most lane.

**Keywords:** ASMs · `Asmeta`· Runtime Safety Enforcement · Safety shield

## 1 Introduction

Mission-critical self-adaptive systems, like autonomous vehicles (AVs), are capable of adapting their behavior at runtime in complex environments with little to no human intervention. Given the critical roles these systems play, they are expected to safely adapt to changes in their execution environment. Runtime adaptation mechanisms of these systems can leverage formal models, referred to as *formal models@run.time* [10]. The applicability of formal analysis techniques can be extended to runtime environment to support adaptation decision-making and applying safety assurance approaches in adaptive systems [2, 24].

Leveraging our previous approach in [7], in this paper, we present a safety enforcement framework for the sanitization of the output control action of an AI-based driving agent in a simulated highway [17]. The proposed enforcement mechanism works as a *safety controller* (or *safety shield*): it observes the environmental changes and the action decided by the controlled AV (referred to as the *ego vehicle* or the *controlled vehicle* throughout the text) that might lead to potential safety violation (i.e., a collision), and then proactively elaborates and actuates a *proper answer* to replace the AV control action (*output sanitization*). The enforcement strategies are formally specified and served at runtime by an *Abstract State Machine* (ASM) [8, 9], developed using the `Asmeta` tool set [6].

Precisely, we validate and verify the correctness of the enforcement strategies, as specified in the `Asmeta` *enforcement model*, against the safety requirement (see Section 3). We keep this pre-analysis separated as an *offline phase*, where we can execute demanding analysis activities (including model checking for the verification of behavioral properties) without interfering with the system operation. For the runtime execution of this `Asmeta` model within the enforcement framework and its connection with the simulated highway environment and driving agents (*online phase*), we use the simulation engine `AsmetaS@run.time`. The latter was recently adapted for its use at run-time and for providing *simulation-as-a-service* exposed via a REST API [2] of `Asmeta` models. We connect our safety enforcer software with the pre-trained (unsafe) AI agents to work as a safety shield. We consider the single-lane case, where agents can accelerate and brake and the enforcement strategy is to *maintain a safe distance without slowing down too much*, and one scenario of the multi-lane case where the enforcement goal is to promote virtuous driving behavior by *preferring the rightmost lane* as required in many countries. To demonstrate the efficacy and efficiency of our approach, we conducted an experimental campaign using the highway simulation environment as provided by the ABZ 2025 case study [18].

The main contributions of this paper are the following:

- The application of the Runtime Safety Enforcement (RSE) approach [7] to the ABZ 2025 case study *Safety Controller for Autonomous Driving* [18];
- A RSE framework implemented using the Python programming language, and causally connected and co-executing with the simulated highway and driving AI-based agents;
- The evaluation of the RSE framework and of the `Asmeta` enforcement model in terms of soundness and computational overhead.

This paper is organized as follows. Section 2 provides some preliminary concepts. Section 3 illustrates the safety enforcement approach for AVs (including addressed requirements and assumptions, details of the RSE framework and of the `Asmeta` enforcement models). Section 4 reports the results of our evaluation experiments, conducted both offline and online. Section 5 highlights related work, and Section 6 concludes the paper.

## 2 Background

This section briefly introduces a mathematical model for AV safety, the `Asmeta` tool set, and the runtime safety enforcement approach.

### 2.1 Responsibility-Sensitive Safety for autonomous driving

*Responsibility-Sensitive Safety* (RSS) by Shalev-Shwartz et al. [21] is a mathematical model suggested by the specification document of the ABZ 2025 case study [18]. The RSS formula for calculating the (longitudinal) safety distance is:

$$d_{RSS} = \left[ v_r \cdot \rho + \frac{1}{2} \cdot a_{max} \cdot \rho^2 + \frac{(v_r + \rho \cdot a_{max})^2}{2 \cdot \beta_{min}} - \frac{v_f^2}{2 \cdot \beta_{max}} \right] +$$

where $\rho$ is the response time, $v_r$ is the speed of the rear (the ego) vehicle, $a_{max}$ is the maximum acceleration of the rear vehicle before braking, $\beta_{max}$ is the maximum braking acceleration of the front vehicle, $\beta_{min}$ is the braking acceleration of the rear vehicle (reaction to braking of front vehicle), and the notation $[x]+$ denotes $\max\{x, 0\}$. Collisions can be avoided by maintaining this safety distance.

### 2.2 ASMs and the `Asmeta` tool set

In this work, we adopt Abstract State Machines (ASMs) to devise the system. They are an extension of Finite State Machines (FSMs) that replaces unstructured control states with states capable of handling arbitrarily complex data represented with dynamic functions. The basic transition rules of ASMs consist of guarded parallel function updates. Specifically, we leverage the features provided by the `Asmeta` framework [2,6], which supports a comprehensive specification and analysis process encompassing the system life-cycle with three main stages: design, development, and operation. Each stage is supported by a variety of tools.

This work combines the design and operation phases, by exemplifying the applicability of `Asmeta` models produced at design time to the runtime environment [2]. In particular, we use the simulator `AsmetaS@run.time` to deploy the `Asmeta` enforcement model as runtime model of the enforcement framework, executing in tandem with the simulated highway environment.

### 2.3 Runtime Safety Enforcement (RSE)

A promising approach to managing complexity in runtime environments is to develop adaptation mechanisms that leverage software models, known as *models@run.time* [4]. A model@run.time is a causally connected self-representation of the associated system (its structure and behavior) or goals of the system from a problem-space perspective. In this work, we use an ASM as *enforcement model* to specify the strategies/policies to apply at runtime to assure safety. Essentially, we implemented an enforcement mechanism from the framework presented in [7] for the *output sanitization* of the ego vehicle's control action before it is actuated. The ego vehicle is simulated by a trained AI-based driving agent as provided by the ABZ 2025 case study [18] for the driving simulation environment [17].

Borrowing the terminology used in [7], we apply a *black-box enforcement* mechanism $E$, which treats the target system $S$ as a black-box (indeed, the AI agent is a black box) by observing only the input ($I$) and output ($O$). Formally:

we denote by $\delta(I, O)$ an operational change made by $S$ producing output $O$ in response to the input $I$. We denote by $\Sigma$ the system state space and by $\Sigma \setminus A(\Sigma)$ the subset of *unsafe states* where safety assertions $A$ may be violated[1]. If $(\sigma, \delta(I, O), \sigma')$ is a state change of $S$ from $\sigma$ to $\sigma'$ with $\sigma' \in \Sigma \setminus A(\Sigma)$, then $E$ try to sanitize $O$ in $O' = E(I, O)$ such that $(\sigma, \delta(I, O'), \sigma'')$ is an operational change with $\sigma'' \in A(\Sigma)$.

Note that in practice $E$ may take more than one corrective step to effectively bring back the system $S$ to the safe region $A(\Sigma)$. Moreover, at runtime there is no guarantee that a safe state is always reached in all situations; the enforceable properties are impacted by uncertain environmental factors or uncontrolled variables that may make not timely and ineffective the enforcer adjustment. These factors include, for example, the monitoring frequency that may not be high enough for having fresh observations of the surrounding environment (see experiments in Section 4), and the abruptness with which the target entities in the system environment change their behavior or appear/disappear.

To apply the RSE approach [7] to a target system, the following key steps are to be carried out throughout the phases of the system life cycle:

1. **Enforcement Strategies Definition**. This involves defining safety assertions, I/O interfaces, and enforcement goals and strategies (@design.time).
2. **Formal Specification and Analysis**. The enforcement model is formally specified, validated and verified to ensure the safety assertions are correctly enforced over the global state of the runtime model (@design.time).
3. **RSE Framework Development and Binding**. An enforcer framework is developed, instantiated and connected to the target system, with the runtime model for safety enforcement (@development.time).
4. **Deployment and Running**. The enforced system is deployed, set up, and put into operation in its runtime environment (@run.time).
5. **Runtime Model Evolution**. Adaptation of safety assertions and enforcement rules to accommodate new requirements (@design.time or @run.time).

The next sections illustrate these steps, except step 5, for the ABZ case study.

## 3 Safety modeling and enforcement approach

In this section, we describe the requirements we have chosen to model in our `Asmeta` specification, our assumptions, and the RSE framework.

### 3.1 Considered requirements and assumptions

We consider the following goals (**step 1** of the RSE process) in order to guarantee the safety requirement `SAF` of the specification document (i.e., no collisions) [18], good performance, and a virtuous driving behavior:

- **G1**: *Maintain a safety distance*;
- **G2**: *Achieve a high total distance traveled safely*;

[1] E.g., driving without maintaining the RSS safety distance may lead to a collision.

Table 1: Enforcement goals, strategies and rules

| **Goal** | **Strategy name** | **Enforcement rule** |
|---|---|---|
| G1 | Go super safe | Brake if the worst case safety distance is violated |
| G1 | Go safe | Brake if the safety distance is violated |
| G2 | Go fast safely | Increase speed if *far away*, i.e. the distance to the front vehicle is $x\%$ (e.g., 70%) greater than the required safety distance |
| G3 | Take the rightmost free lane | Change lane to right if the lane directly right is free |

- **G3**: *Keep to the right-most free lane.*

The enforcement strategies adopted by the enforcer to achieve such goals are reported in Table 1. These strategies are examples of compensation actions that the enforcer can put in place to achieve the prefixed goals. In order to achieve all goals, more than one strategy must be used and combined. Note that for the same goal G1 two alternative strategies can be adopted. Strategy *Go super safe* is the most prudent; it aims to maintain an upper bound of the distance as calculated by the RSS formula in the *worst case scenario*. Such an upper bound is calculated using both the maximum speed and the maximum acceleration, and assuming that the front vehicle speed is zero. Strategy *Go safe*, instead, maintains the safety distance as calculated by the RSS formula (see Section 2.1).

To concretely implement these enforcement strategies in the simulated highway, we had to make some assumptions. Specifically, we considered all assumptions contained in the specification document: `VEH1-ENV`–`VEH7-ENV` for the concrete values of the vehicle parameters (length, width, maximum speed, etc.), `CON1` (all controlled vehicles observe the environment every $t$ seconds), `ENV1` (there is at least one vehicle on the highway), `ENV2` (all vehicles drive in the same direction), and `ENV3` (the number of lanes does not change over time). We also adhere to the *cycle*-based execution semantics of the specification document: every time interval $t$ (also called *response time*) a controlled vehicle observes its environment, and decides to perform an action until reaching the next cycle; the other vehicles may actuate actions at different times instead.[2]

In addition, we considered also the following assumptions, due in part to constraints of the provided simulation environment [17] and its configuration for agent training [18], as obtained by code inspection and direct and extensive experimentation:

1. Within the simulation environment, the response time $t$ of the controlled vehicle depends on the *policy frequency* (expressed in Hz). Whereas, the time interval between decisions made by other vehicles is determined by the *simulation frequency* (expressed in Hz).

2. The runtime observation interface allows us to observe presence, positions, and speeds of the controlled vehicle and of its four closest front vehicles up to a range of 200m.

[2] E.g., a vehicle braking in the first half of the cycle can accelerate in the second half.

3. As required by ML models for a better accuracy, the observed features are normalized (using the min-max method) in range [-1,1] and are relative to the controlled vehicle, while those of the controlled vehicle are absolute. Note that the RSS formula requires absolute values, therefore, the observations are de-normalized before they are used by the enforcement model.

4. The maximum observable distance from the front vehicle may be smaller than the safety distance, even with an infinitesimally small response time.

5. When calculating the safety distance RSS, three cases could be considered: (i) the speed of the controlled vehicle is less than the maximum speed and, after accelerating with the maximum acceleration during the response time, it is still less than the maximum speed; (ii) the speed of the controlled vehicle is less than the maximum speed, but when accelerating with the maximum acceleration it reaches the maximum speed before the response time has elapsed; (iii) the speed of the controlled vehicle is equal to the maximum speed. In the first case, the RSS formula is used as it is. In the second case, the RSS formula could be adapted to be more specific, but we use it as it is; therefore, a higher RSS value is used, making the model more stringent. In the last case, the RSS formula is used by setting the maximum acceleration to 0 $m/s^2$.

6. In the multi-lane case, when considering which vehicles are occupying a lane, the following assumptions are made:

– All vehicles traveling straight ahead in the lane occupy the lane;
– All vehicles leaving the lane but not yet traveling straight on the other lane occupy the lane;
– All vehicles leaving another lane and just entering the lane but not yet traveling straight on occupy the lane.

Consequently, it is possible for a vehicle to occupy multiple lanes.

7. In the multi-lane case, we consider a vehicle proceeding straight on a lane when its $y$ position is close enough to the $y_{lane}$ position associated to a lane with a given tolerance $\Theta$ (default $0.1\,m$):

$$y_{lane} - \Theta < y < y_{lane} + \Theta$$

8. When calculating the distance between two vehicles, the vehicle dimensions, particularly the length, are considered. The x and y coordinates when observing a vehicle refer to its center. Therefore, when considering the distance between two vehicles, it is necessary to consider the front half of the length for the rear vehicle and the rear half of the length for the front vehicle, i.e. the full length of a vehicle must be removed from the observed distance between two vehicles.

9. If there is no observable front vehicle (either because it is further than the maximum observable distance or there are four closest vehicles in the other lanes), the enforcer is not activated since not necessary. The enforcer is not activated also when the controlled vehicle is changing lane until the maneuver is

complete; this steady state ensures that any adaptation made by the enforcement is based on a stable snapshot of the controlled vehicle's status[3].

10. The RSS safety distance refers only to longitudinal (X-axis) positions and speeds. The lateral position (on the Y-axis) is used to determine which lane each vehicle is in, while the lateral speed is not used. The Y-axis position and speed could be used in some formulas such as the one defined in Lemma 4 in [21] to calculate the lateral (Y-axis) safety distance, but some variables (e.g., the maximum lateral acceleration) required by this formula are not provided.

11. We say that the lane directly right is free if no front vehicles would be observed by the ego vehicle once changed lane to right maintaining the same X-position.

## 3.2 Enforcement model details

We here illustrate excerpts of the `Asmeta` models specifying the enforcement strategies reported in Table 1 (specification task of **step 2** of the RSE process). We incrementally defined various models to achieve all the three goals (goals G1 and G2 for both the single-lane and multi-lane cases, and G3 for the multi-lane case only). A replication package containing these models, all software artifacts, and data sets used in the evaluation of our approach is available online at [5].

For this case study, the enforcement rules are extremely intuitive and take the form of an ASM conditional rule ***if*** *cond* ***then*** *updates*. They are actually aimed at correcting the output (0-ary) function `outAction` representing the final decision, i.e. the control action (chosen from the set `{FASTER, SLOWER, IDLE, LANE_LEFT, LANE_RIGHT}`) that replaces the one chosen by the AI agent and is actuated on the ego vehicle until the next observation and decision.

*Case single-lane.* To achieve G1, first we introduced the `Asmeta` model for the *Go super safe* strategy. Essentially, to promote a super safe policy we introduced the threshold `dRSS_upper_bound` denoting an upper bound on the safest distance observable in the worst-case scenario and calculated setting $v_r$ to $v_{max}$ and $v_f$ to 0:

$$d_{RSS\_upper\_bound} = \left[ v_{max} \cdot \rho + \frac{1}{2} \cdot a_{max} \cdot \rho^2 + \frac{(v_{max} + \rho \cdot a_{max})^2}{2 \cdot \beta_{min}} \right]_+$$

where $v_{max}$ is the maximum speed of the ego vehicle. The corresponding enforcement rule is reported in Code 1. The rule is triggered when the actual longitudinal distance to the front vehicle (the 0-ary function `actual_distance`)[4] is less than `dRSS_upper_bound`. In this case, the enforcement model prescribes slowing down. An alternative way to achieve G1 is by maintaining the RSS distance (*Go safe* strategy) as specified by the enforcement rule reported in Code

[3] In a self-adaptive system, it is usually assumed that the target system is in a steady state before an adaptation is triggered in response to environmental changes [23].

[4] In all models, the `actual_distance` is a derived function defined using the X-axis positions of the vehicles: `actual_distance = x_front - x_self - l_vehicle`.

```
macro rule r_unsafeDistanceSuperSafe =
 if actual_distance <= dRSS_upper_bound
      then outAction := SLOWER endif
```

Code 1: Enforcement rule for goal G1, strategy *Go super safe.*

```
macro rule r_unsafeDistanceSafe =
 if actual_distance <= dRSS
      then outAction := SLOWER endif
```

Code 2: Enforcement rule for goal G1, strategy *Go safe.*

```
macro rule r_goFast =
 if actual_distance > dRSS * gofast_perc
      then outAction := FASTER endif
```

Code 3: Enforcement rule for goal G2, strategy *Go fast.*

```
macro rule r_keepRight = if rightLaneFree
 then outAction := LANE_RIGHT
 endif
```

Code 4: Enforcement rule for goal G3, strategy *Take the rightmost free lane.*

2, where the (0-ary) function `dRSS` is calculated from the observations of the current cycle using the formula described in Section 2.1.

To achieve goal G2, we introduce the enforcement rule shown in Code 3 that prescribes to speed up safely, i.e. when the front vehicle is more than a certain percentage of the `dRSS` distance (e.g., 70%).

*Case multi-lane.* Enforcement rules introduced for goals G1 and G2 in the single-lane case can be used also in the multi-lane setting, where each vehicle can also change lanes. Additionally, we introduce the enforcement rule shown in Code 4 to achieve goal G3, namely to promote a virtuous driving style by preferring the lane directly right when free (as by our assumption 11). We postpone dealing with more complex multi-lane scenarios to future work (**step 5** of the RSE process). framework. Note that the enforcement model can evolve and be re-engineered separately, and re-deployed easily without re-building the component framework. This is the main flexibility of using runtime models that are causally linked to the target system, rather than integrated into the system via embed code or model synthesis.

To combine enforcement rules we use the ASM par-rule constructor in the main rule (entry point of execution) of an `Asmeta` model as shown in Code 5, where we select one rule per each goal. Moreover, since the main rule consists of guarded parallel updates of the same out function `outAction`, in order to avoid inconsistent function updates we applied the semantic pattern *mutual exclusive guards*[5] by making the guards of the enforcement rules mutually exclusive. As an example, Code 6 shows how the `r_unsafeDistance` rule has been restructured after applying such a pattern. Note that in the proposed rule scheduling, we prioritize the change to the right lane rather then other actions, whenever it is possible to do it safely (i.e., the lane directly right is free).

[5] The workflow of the machine follows only one of the possible parallel execution paths.

```
main rule r_Main = par
   r_unsafeDistance[]
   r_goFast[]
   r_keepRight[]
 endpar
```

Code 5: Main rule of the Asmeta enforcement model

```
macro rule r_unsafeDistance =
   if actual_distance <= dRSS
      if not rightLaneFree then
            outAction := SLOWER
   endif endif
```

Code 6: Enforcement rule for goal G1, strategy *Go safe* (refined).

### 3.3 RSE framework for driving agents on a simulated highway

According to **step 3** of the RSE process, we designed the enforcer mechanism to act as a proxy system which wraps the controlled AI driving agent. We developed it using the Python programming language and embedded it into a closed loop setting with the AI agent in the simulated highway environment of AVs.

Figure 1 shows an overview of the architecture of the enforcer component (the subsystem `Enforcer Framework`) once instantiated and bound to the simulated highway environment, using a UML-like notation. The I/O interfaces[6] used by the ego AI agent (the subsystem `Autonomous Driving System`) and by the `Enforcer` are as follows: the input interface `I` corresponds to the AVs observations as provided by the simulated environment, while the output interface `O` is the action decided by the ego autonomous agent. The `Observation Processor` is responsible for de-normalizing run-time observations and making them absolute (see assumption 3 of Sect. 3.1). The interface `I'` corresponds to the de-normalized and absolute observations and the interface `O'` corresponds to the sanitized action to actuate. The `Configuration Manager` initializes the environment and framework using the configuration file `Config.json`, setting parameters like the number of highway lanes and the vehicle's policy frequency. The `Model Uploader` and the `Enforcer` interact with the `ASM@run.time Simulator` via its `AsmetaS REST API`. The `Model Uploader` is responsible for uploading the proper `Asmeta` enforcement model into the online simulator, while the `Enforcer` is in charge of starting, executing (one single step per cycle), and stopping the `Asmeta` model. Finally, the `Experiment Data Exporter` and the `Logging Manager` collect quality metrics and logging data for debugging/manual inspection, respectively.

## 4 Evaluation

In this section, we evaluate our proposed approach. Our benchmark contains 4 enforcement models, listed in Table 2 along with the strategies they implement and the goals they address. The first three models (`SafetyEnforcerSuperSafe`, `SafetyEnforcerSlower`, and `SafetyEnforcerFaster`) were applied on a single-lane highway. The fourth model, `SafetyEnforcerKeepRight`, was applied on

[6] The circle (or ball) indicates input events or data that the component can handle; the semi-circle (or socket) indicates output events or data from the component.

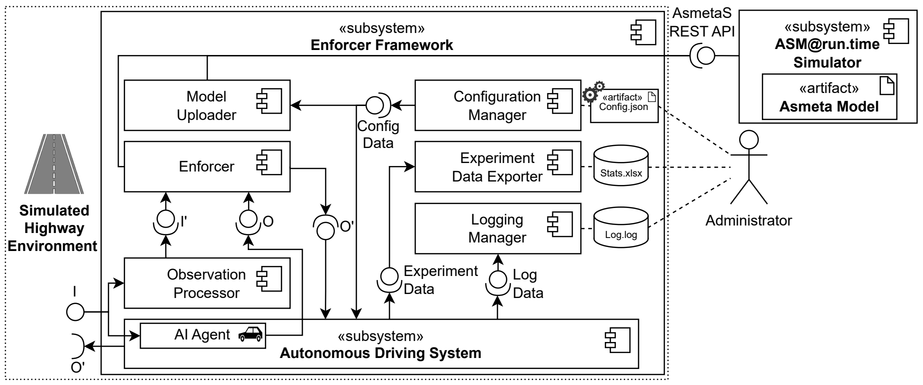


Fig. 1: RSE framework for simulated AVs driving on a highway.

Table 2: Mapping of Asmeta models to driving strategies and safety goals.

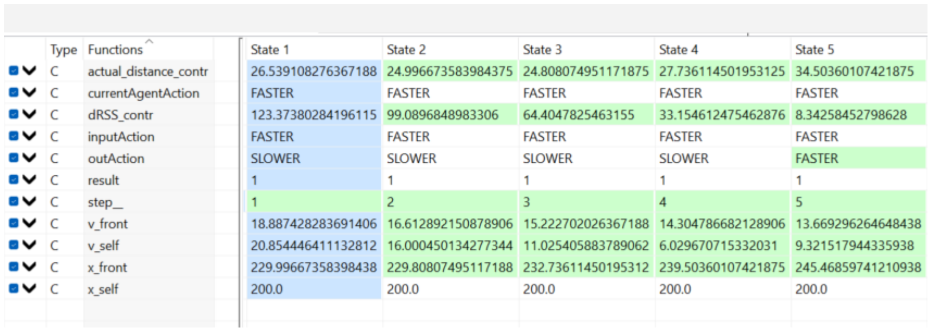
| Asmeta Enforcement Model | Strategies | Goals |
|---|---|---|
| SafetyEnforcerSuperSafe.asm | Go super safe | G1 |
| SafetyEnforcerSlower.asm | Go safe | G1 |
| SafetyEnforcerFaster.asm | Go safe, Go fast safely | G1, G2 |
| SafetyEnforcerKeepRight.asm | Go safe, Go fast safely, Take the rightmost free lane | G1, G2 G3 |

a 3-lane highway where all vehicles travel in the same direction. We start by presenting the offline validation and verification activities and, then, we discuss the experiments we run to assess the efficiency and efficacy of our framework.

## 4.1 Offline validation & verification

We here report on the validation and verification results (analysis task of **step 2** of the RSE process) for the functional correctness of the enforcement models. This stage is carried out offline (i.e., at design-time) before releasing the `Asmeta` enforcement model in production with the enforcer framework and letting them co-operate at runtime. To ensure that an enforcement model behaves as expected, i.e. it achieves the enforcement goal(s) for which it was designed, we carried out both model validation by scenarios using the validator `AsmetaV`, and property verification using the `Asmeta` model checker `AsmetaSMV`.

We run different scenarios on each model to cover all rules. In Figure 2 we report an example of scenario execution on the `SafetyEnforcerFaster.asm` specification. After a few steps, where the safety distance was violated and the enforcer forces the ego vehicle to go slower (`outAction=SLOWER`), once the front vehicle is far enough the strategy G2 is implemented (`outAction=FASTER`). Note that `AsmetaV` does not support natively approximate comparison among real numbers, so we had either to specify the values with all decimal digits (like in Fig. 2) or use the `abs` function explicitly.



| Type | Functions | State 1 | State 2 | State 3 | State 4 | State 5 |
|---|---|---|---|---|---|---|
| C | actual_distance_contr | 26.539108276367188 | 24.996673583984375 | 24.808074951171875 | 27.736114501953125 | 34.50360107421875 |
| C | currentAgentAction | FASTER | FASTER | FASTER | FASTER | FASTER |
| C | dRSS_contr | 123.37380284196115 | 99.0896848983306 | 64.4047825463155 | 33.154612475462876 | 8.34258452798628 |
| C | inputAction | FASTER | FASTER | FASTER | FASTER | FASTER |
| C | outAction | SLOWER | SLOWER | SLOWER | SLOWER | FASTER |
| C | result | 1 | 1 | 1 | 1 | 1 |
| C | step_ | 1 | 2 | 3 | 4 | 5 |
| C | v_front | 18.887428283691406 | 16.612892150878906 | 15.222702026367188 | 14.304786682128906 | 13.669296264648438 |
| C | v_self | 20.854446411132812 | 16.000450134277344 | 11.025405883789062 | 6.029670715332031 | 9.321517944335938 |
| C | x_front | 229.99667358398438 | 229.80807495117188 | 232.73611450195312 | 239.50360107421875 | 245.46859741210938 |
| C | x_self | 200.0 | 200.0 | 200.0 | 200.0 | 200.0 |

Fig. 2: Scenario execution for the `SafetyEnforcerFaster.asm` specification.

Regarding the verification process, we have used `AsmetaSMV` by invoking `NuXmv` [12], the symbolic model checker to analyze synchronous finite-state and infinite-state systems. This choice became necessary because the specification in `Asmeta` makes use of real, hence infinite, domains. At first, we started by proving Linear Temporal Logic (LTL) properties, with these general forms:

**LTLSPEC** g($\phi$ implies $\varphi$)
**LTLSPEC** g($\phi$ implies x($\varphi$))

However, the model checker was not able to prove their correctness. Due to their form, we expressed properties as invariants, propositional formulas that must always hold in the model. As an example, we report the invariants of the model `SafetyEnforcerFaster` we have verified using IC3 engines [13].

*/*If the ego vehicle is close to the front vehicle, break (go SLOWER)*/*
**INVARSPEC NAME** invar_01 := (actual_distance<=dRSS) −> **next**(outAction=SLOWER)

*/*If the front vehicle is far enough from the ego vehicle, increase the speed (go FAST)*/*
**INVARSPEC NAME** invar_02 := (actual_distance>(dRSS*gofast_perc)) −> **next**(outAction =FASTER)

*/*If there is no risk of collision, keep the action decided by the agent*/*
**INVARSPEC NAME** invar_03 := (actual_distance>dRSS and actual_distance<=(dRSS* gofast_perc)) −> **next**(outAction=currentAgentAction)

### 4.2 Online validation

Once in operation in the simulated highway (**step 4** of the RSE process), we check whether our enforcement framework, based on the enforcement rules as served at run-time by the `Asmeta` model, is able to ensure under change the required safety and the other considered driving requirements. Specifically, we address the following Research Questions (RQs):

**RQ1** *How effective is the enforcer in ensuring safety while adjusting speed and driving style?*

**RQ2** *What is the computation overhead of running the enforcer in combination with the simulated system?*

RQ1 investigates how well the corrective measures of our enforcement framework are able to deliver the intended behavior by enhancing three key outcomes: It ensures safety by avoiding collisions, travels a significant distance, and spends a significant time on the rightmost lane. RQ2 is regarding the cost, in terms of computation time (the wall-clock time), of the enforcement software. This section reports on the design of the evaluation and the major results.

## 4.3 Design of the evaluation

To answer RQ1 and RQ2, we compared the enforced driving agent of the controlled vehicle with the non-enforced one (baseline system). Both RQs are addressed with the same setup. The experiments were performed on a PC with Intel(R) Core(TM) i7-8550U CPU @ 1.80 GHz and 16 GB RAM, running Windows 11. The server exposing the Spring REST API of `AsmetaS` was executed natively on Windows 11, while the enforcement framework was executed on the same machine via WSL (Windows Subsystem for Linux) running Ubuntu 24.04, within a Python virtual environment using Python 3.11 as the runtime platform.

For both lane configurations (single-lane and multi-lane), the case study [18] provides two agents trained using Deep Q-learning (DQN): A *base agent*, which receives penalties for collisions and rewards for maintaining high speed and staying in the right lane, and an *adversarial agent*, which is rewarded for collisions, high speed, and frequent lane changes. This setup results in 8 different configurations for the single-lane scenario: no enforcement and 3 enforcement models applied to both the base and adversarial agents; and 4 configurations for the multi-lane scenario: no enforcement and the multi-lane enforcement model applied to both agent types.

The experiments were conducted extending the test run duration from 40 simulation seconds, as defined in the case study, to 100 simulation seconds. This duration was chosen based on manual observations of the agent's behavior executing without the enforcer. In the single-lane configuration, the base (non-adversarial) agent's performance deteriorates due to poor decision-making. A longer duration allows us to quantitatively assess this decline in performance. A test run concludes in one of two cases: either the ego vehicle crashes or the simulation duration ends. The experiments were initially conducted with a *policy frequency* of 1 Hz, which means that the agent made one decision per simulation second, resulting in a response time of 1 sec. The experiments were then repeated with a doubled policy frequency (2 Hz), reducing the response time to 0.5 sec. The simulation frequency was set to the default value of 15 Hz. Therefore, a total of 24 configurations were tested, with 50 test runs each, and metrics were recorded during every run.

Our efforts to introduce enforcement techniques aims to enhance the safety of autonomous vehicles. To evaluate this in RQ1, we meticulously record the *number of crashes* that occurred in various scenarios and with different enforcement models. More specifically, for each simulation run, we recorded whether it terminated because of a crash or because the simulation duration expired. Considering that one of our objective, beyond the safety, is to let the vehicle travel the longest distance, we also recorded the *distance* traveled by the ego vehicle during the test run, expressed in kilometers and computed by multiplying speed by the simulation time. Furthermore, in our experiments we favor scenarios in which the vehicle travels in the rightmost lane. Thus, we record (only when multiple lanes are available) the *distance traveled on the rightmost lane* by the ego vehicle during the test run, expressed in kilometers.

Additionally, in RQ2, we are interested in evaluating the overhead introduced by the enforcement framework. To measure the impact of *enforcer interventions* on output sanitization, we calculated the percentage of times the interventions changed the agent's action compared to the total number of actions performed by the ego vehicle (i.e., the product of the effective durations by the policy frequency). Moreover, we collected, for each test run, the total *execution time* and the *enforcement overhead* introduced by the enforcer, consisting of the time required to start and stop the execution of the `Asmeta` model and the time required to perform all output sanitizations. Both times are measured in wall-clock seconds. The overhead measurements include the time associated with the HTTP request and the execution time of the `Asmeta` model, which serves as the run-time model for enforcement. To collect the wall-clock time we used `time.perf_counter()` to capture high-precision timestamps.

## 4.4 Results

Table 3 reports the results of our statistical tests. In the following, we discuss them for the two proposed research questions separately.

*RQ1 - Effectiveness.* To assess the effectiveness of our enforcement approach, we have measured, for each test run, the number of crashes, the traveled distance and, in case multiple lanes were available, the distance traveled on the rightmost lane (see Table 3). The box plot depicted in Figure 3 visually shows the results.

Regarding the number of crashes (see Figure 3a), our results confirm the positive impact of enforcement techniques: regardless of the enforcement model, only one crash were registered when the enforcer was activated, while 210 happened if no enforcer was used. The only crash reported when using the enforcement framework happened in the multi-lane scenario with adversarial agents, when using the lowest policy frequency (i.e., 1 Hz). A manual inspection revealed that the limited observation interface that allows to observe only the four closest front vehicles, as described in assumption 2 of Sect. 3.1, made it possible to observe the nearest vehicle on the same lane only when overtaking a vehicle in a different lane. This limitation, combined with a high response time, ultimately resulted in a crash. Increasing the policy frequency, i.e., reducing the response time, allowed us to solve this limitation and to avoid any crash. To further validate this finding, we ran 50 additional experiments with the *KeepRight* model for the multi-lane scenario with the adversarial agent and a policy frequency of 2 Hz, which resulted in no occurrence of crashes. However, we acknowledge that rare failure cases may still be possible. Interestingly, reducing the response time never reduced the number of crashes when no enforcement was used. Overall, we

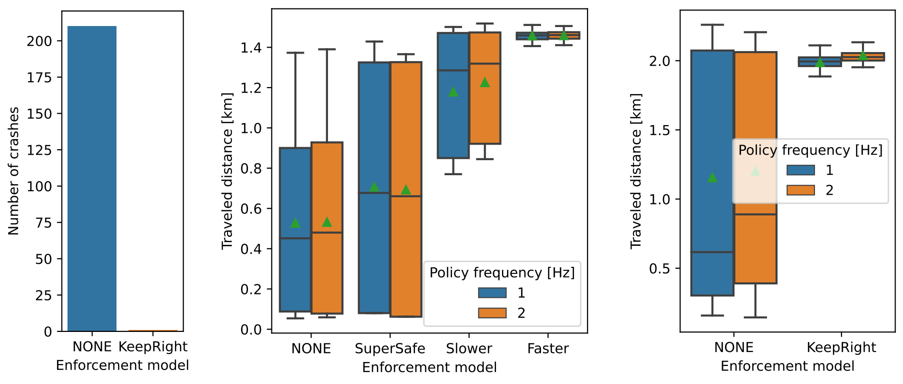
Table 3: Effectiveness and efficiency of the safety enforcement models.

| **Policy Freq. [Hz]** | **Lane Config.** | **Agent** | **Enforc. Model** | **Number of Crashes** | **Distance [km]** | **Distance on Right Lane [km]** | **Enforc. Interventions [%]** | **Execution Time [s]** | **Enforc. Overhead [s]** |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Single-lane | Base | — | 0 | 0.97 ± 0.17 | — | — | 11.34 ± 1.52 | — |
| | | | SuperSafe | 0 | 0.08 ± 0.00 | — | 14.70 ± 0.58 | 12.21 ± 1.30 | 0.70 ± 0.50 |
| | | | Slower | 0 | 0.89 ± 0.09 | — | 21.14 ± 0.93 | 13.94 ± 0.95 | 2.23 ± 0.28 |
| | | | Faster | 0 | 1.44 ± 0.02 | — | 40.28 ± 1.53 | 14.10 ± 1.59 | 2.42 ± 0.37 |
| | | Adversarial | — | 50 | 0.08 ± 0.01 | — | — | 0.29 ± 0.06 | — |
| | | | SuperSafe | 0 | 1.34 ± 0.04 | — | 63.04 ± 1.48 | 13.47 ± 0.60 | 1.93 ± 0.18 |
| | | | Slower | 0 | 1.47 ± 0.02 | — | 51.00 ± 0.00 | 14.53 ± 0.32 | 2.92 ± 0.06 |
| | | | Faster | 0 | 1.47 ± 0.02 | — | 51.00 ± 0.00 | 13.89 ± 0.39 | 2.33 ± 0.06 |
| | Multi-lane | Base | — | 4 | 1.97 ± 0.46 | 1.79 ± 0.58 | — | 13.61 ± 3.14 | — |
| | | | KeepRight | 0 | 2.01 ± 0.07 | 1.34 ± 0.87 | 48.62 ± 5.15 | 17.53 ± 0.75 | 2.21 ± 0.26 |
| | | Adversarial | — | 50 | 0.35 ± 0.18 | 0.09 ± 0.08 | — | 1.46 ± 0.73 | — |
| | | | KeepRight | 1 | 1.97 ± 0.26 | 0.83 ± 0.97 | 50.15 ± 2.55 | 17.46 ± 2.37 | 2.23 ± 0.31 |
| 2 | Single-lane | Base | — | 0 | 0.99 ± 0.15 | — | — | 11.99 ± 0.46 | — |
| | | | SuperSafe | 0 | 0.06 ± 0.00 | — | 14.56 ± 0.41 | 13.98 ± 0.40 | 1.02 ± 0.05 |
| | | | Slower | 0 | 0.98 ± 0.11 | — | 15.53 ± 2.48 | 18.71 ± 0.77 | 4.88 ± 0.44 |
| | | | Faster | 0 | 1.45 ± 0.02 | — | 32.51 ± 1.45 | 18.55 ± 0.55 | 4.87 ± 0.13 |
| | | Adversarial | — | 50 | 0.08 ± 0.01 | — | — | 0.28 ± 0.06 | — |
| | | | SuperSafe | 0 | 1.32 ± 0.03 | — | 59.02 ± 0.73 | 17.47 ± 0.32 | 3.65 ± 0.08 |
| | | | Slower | 0 | 1.48 ± 0.02 | — | 50.50 ± 0.00 | 19.78 ± 0.17 | 6.07 ± 0.06 |
| | | | Faster | 0 | 1.47 ± 0.02 | — | 50.50 ± 0.00 | 19.04 ± 0.48 | 4.85 ± 0.13 |
| | Multi-lane | Base | — | 6 | 1.96 ± 0.41 | 1.58 ± 0.72 | — | 15.83 ± 3.25 | — |
| | | | KeepRight | 0 | 2.03 ± 0.05 | 1.49 ± 0.82 | 46.53 ± 8.78 | 22.85 ± 1.13 | 4.53 ± 0.74 |
| | | Adversarial | — | 50 | 0.44 ± 0.20 | 0.09 ± 0.11 | — | 2.01 ± 0.93 | — |
| | | | KeepRight | 0 | 2.04 ± 0.06 | 0.89 ± 0.96 | 48.21 ± 6.08 | 22.22 ± 0.37 | 4.67 ± 0.15 |

Values report mean ± standard deviation, except for 'Number of Crashes' which is reported as the total count; Times values refer to clock-wall time. The prefix *SafetyEnforcer* is omitted from filenames, as well as the suffix *.asm*.

can state that adopting our `Asmeta`-based enforcement framework is effective in reducing the number of crashes of the considered agent.

When it comes to the traveled distance (see Figures 3b and 3c) we can see that its value increases as the goal moves from G1 to G2/G3, i.e., from *Super safe* to the *Go fast safely*/*Take the rightmost free lane* strategy. As for the number of crashes, using our proposed solution allows for improving the measured results. Moving towards G2/G3 increases the speed of the vehicle, but this does not influence its safety. Interestingly, when considering the distance traveled on the rightmost free lane, using the safety enforcement strategy leads to different results depending on the agent type. On the one hand, when the agent is *non-adversarial*, using a safety enforcement model leads to a decrease in the distance traveled on the rightmost lane, while still traveling a higher total distance. This is because the agent was trained with a reward for staying in the right lane, whereas the safety enforcer prioritizes the driving safety, changing to the right lane only when it is completely free. Consequently, the vehicle may spend less time in the rightmost lane when doing so helps to avoid crashes. On the other hand, if the agent is *adversarial* (trained without a reward for staying in the right lane), the vehicle spends more time on the rightmost lane when under enforcement.



(a) # Crashes (b) Distance with Single-lane (c) Distance with Multi-lane

Fig. 3: Effectiveness analysis.

**RQ1 Summary.** The `Asmeta`-based RSE framework effectively reduces the number of collisions, while favoring the right lane (when safe) and often achieving a higher distance traveled within the simulation time than driving without safety enforcement.

*RQ2 - Efficiency.* To assess the efficiency of the proposed enforcement framework and models, we have recorded (see Table 3), for each test run, the percentage of enforcer interventions (i.e., the number of times in which the enforcer changed the decision previously made by the agent divided by the total number of decisions), the total execution time, and the overhead due to the use of the enforcer. The overhead does not include the time to de-normalize and convert the observations to absolute values, as this computation is also necessary to collect metrics when the enforcer is not running. For the sake of completeness, we report that this computation is approximately 70-80% less costly than the enforcement overhead.

For what concerns the number of interventions we can observe that, both in single-lane and multi-lane scenarios, the number of interventions is greater with the adversarial agent. This is because the adversarial agent makes more dangerous decisions, which must be sanitized to return the AV to a safe state. Furthermore, in the single-lane scenario, we can observe two different trends, depending on the agent type. On the one hand, when the agent behaves as non-adversarial, we can see that the number of interventions increases as the goal moves from G1 to G2, i.e., from *Super safe* to the *Go fast safely* strategy. This is due to the fact that, if no front vehicle is observed, the enforcer is not activated, as described in assumption 9 in Section 3.1, and the agent slows down until the vehicle stops. In such a case, both the *Super safe* and *Go safe* strategies do not intervene, but the *Super safe* strategy brings the simulation to such a case earlier, resulting in fewer interventions. Meanwhile, the *Go fast safely* strategy actively intervenes even when the front vehicle is far, resulting in an increased number of interventions. On the other hand, with an adversarial agent, the number of enforcement interventions decreases when switching from the *Super safe* to the *Go safe* strategy and remains stable when adding the *Go fast safely* strategy. This is due to the fact that such an agent always tries to increase the speed. Therefore, at the beginning of the simulation, the *Super safe* strategy requires more interventions to ensure a safe state compared to the *Go safe* strategy and, after that, the two strategies will behave very similarly. In addition, the *Go fast safely* strategy is never activated, therefore it does not increase the number of interventions. In all cases, however, using one of the enforcement models allows for higher execution time w.r.t. the baseline scenario in which no enforcer is used. This result is compliant to what we observed in RQ1: with the enforcer active, the number of crashes is lower, and the AV can proceed more often till the end of the simulation time. Furthermore, the overhead introduced by the enforcer is negligible in all analyzed scenarios and does not exceed an average of 6.07 sec in any configuration.

**RQ2 Summary.** The `Asmeta`-based RSE framework does not introduce significant time overhead compared to driving without safety enforcement.

## 5 Related Work

The complexity of AI-based systems can hinder safety assurance for testers and developers. Some literature suggests using simple techniques to control these systems. For instance, [20] proposes leveraging a simple and verified controller taking control over an unverified system to enforce safety properties. Similarly, in this paper, we use a simple `Asmeta` model to force unsafe AI-based agents to behave in a safe and predictable manner. Our approach, presented in [7], is inspired by the RSS properties suggested in [21] and acts as a safety shield.

Runtime assurance for neural controllers is crucial in software engineering. In [19], the authors introduce *Neural Simplex Architecture* for potentially unsafe neural controllers, improving safety through online retraining without significantly impacting performance. Two different approaches are analyzed in [1], in which the safety shield is placed either before (pre-shielding) or after the system under control (post-shielding), and our approach is comparable with this last post-shielding technique. In the future, we may explore if corrective actions from an `Asmeta` model can aid in retraining AI-based controllers.

The approach in [22] uses a ProB specification alongside a reinforcement learning (RL) agent as a safety shield. Unlike this solution implemented in ProB, which requires the RL agent to receive a set of enabled operations and to chose among them, our approach uses an `Asmeta` model for correcting the output of the agent. The formulas we adopted are derived from those presented in the case study description and in [14], where the authors used them to prove safety properties through model-based assurance cases in Isabelle, or in Event-B [16].

In [7], we proposed a gray-box approach to safety enforcement. In addition to the I/O, this mechanism can observe specific system's operational changes and compute more effective enforcement actions to maintain safety through probing and effecting interfaces provided by the target system. This mechanism uses `Asmeta` enforcement models and the MAPE-K feedback loop [15] to architect the enforcer as an autonomic manager for self-adaptation. We here adopted a black-box enforcement mechanism instead, as no probe/effector interfaces (and/or explanation facilities about the agent behavior [3,11]) are available to monitor and adapt the AV; we treated it as a black box by observing only its I/O.

## 6 Conclusion

We presented a RSE framework for the output sanitization of the AI-based ego AV of the highway simulation environment [17] for the ABZ 2025 case study [18]. The main enforcement goals consist in maintaining safety while achieving good performance in terms of total distance traveled and time spent on the right-most free lane. The enforcer wraps the ego vehicle agent and, using decisions made by an `Asmeta` runtime model, adjusts the agent's control action to meet enforcement goals and, possibly, overrides agent's decisions when they are considered to be unsafe. Though the case study is based on a simulated environment and we considered a limited number of scenarios, the two forms of analysis we conducted, @design.time and @run.time, suggest a new way of analyzing safety-critical software. This new approach combines the rigor of formal safety-critical analysis environments during system design or development with the benefits of run-time analysis when the system is in operation and more realistic evidence is available.

**Acknowledgments.** This work has been partially funded by PNRR - ANTHEM (AdvaNced Technologies for Human-centrEd Medicine) - Grant PNC0000003 – CUP: B53C22006700001 - Spoke 1 - Pilot 1.4. and by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU and by the European Union - Next Generation EU. We also acknowledge financial support of the project PRIN 2022 SAFEST (Trust assurance of Digital Twins for medical cyber-physical systems), funded by the European Union - Next Generation EU, Mission 4, Component 2, Investment 1.1, CUP F53D23004230006, under the National Recovery and Resilience Plan (NRRP) – Grant Assignment Decree No. 959 adopted on 30 June 2023 by the Italian Ministry of University and Research (MUR).

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe Reinforcement Learning via Shielding. Proceedings of the AAAI Conference on Artificial Intelligence **32**(1) (Apr 2018). https://doi.org/10.1609/aaai.v32i1.11797

2. Arcaini, P., Bombarda, A., Bonfanti, S., Gargantini, A., Riccobene, E., Scandurra, P.: The ASMETA Approach to Safety Assurance of Software Systems. In: Logic, Computation and Rigorous Methods - Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday. Lecture Notes in Computer Science, vol. 12750, pp. 215–238. Springer (2021). https://doi.org/10.1007/978-3-030-76020-5_13
3. Bersani, M.M., Camilli, M., Lestingi, L., Mirandola, R., Rossi, M.G., Scandurra, P.: Architecting Explainable Service Robots. In: Tekinerdogan, B., Trubiani, C., Tibermacine, C., Scandurra, P., Cuesta, C.E. (eds.) Software Architecture - 17th European Conference, ECSA 2023, Istanbul, Turkey, September 18-22, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14212, pp. 153–169. Springer (2023). https://doi.org/10.1007/978-3-031-42592-9_11
4. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10), 22–27 (2009). https://doi.org/10.1109/MC.2009.326
5. Bombarda, A., Bonfanti, S., Gargantini, A., Pellegrinelli, N., Scandurra, P.: Replication Package for the Paper: Safety Enforcement for Autonomous Driving on a Simulated Highway Using Asmeta models@run.time. https://github.com/foselab/abz2025_casestudy_autonomous_driving (2025)
6. Bombarda, A., Bonfanti, S., Gargantini, A., Riccobene, E., Scandurra, P.: ASMETA Tool Set for Rigorous System Design. In: Formal Methods - 26th International Symposium, FM 2024, Milan, Italy, September 9-13, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14934, pp. 492–517. Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_28
7. Bonfanti, S., Riccobene, E., Scandurra, P.: A Component Framework for the Runtime Enforcement of Safety Properties. Journal of Systems and Software **198**, 111605 (2023). https://doi.org/https://doi.org/10.1016/j.jss.2022.111605
8. Börger, E., Raschke, A.: Modeling Companion for Software Practitioners. Springer, Berlin, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56641-1
9. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer Verlag (2003). https://doi.org/10.1007/978-3-642-18216-7
10. Calinescu, R., Kikuchi, S.: Formal Methods @ Runtime. In: Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems. pp. 122–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21292-5_7
11. Camilli, M., Mirandola, R., Scandurra, P.: XSA: eXplainable Self-Adaptation. In: 37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022. pp. 189:1–189:5. ACM (2022). https://doi.org/10.1145/3551349.3559552
12. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: Biere, A., Bloem, R. (eds.) CAV. Lecture Notes in Computer Science, vol. 8559, pp. 334–342. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_22
13. Cimatti, A., Griggio, A.: Software Model Checking via IC3. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification. pp. 277–293. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_23
14. Crisafulli, P., Taha, S., Wolff, B.: Modeling and analysing Cyber–Physical Systems in HOL-CSP. Robotics and Autonomous Systems **170**, 104549 (2023). https://doi.org/10.1016/j.robot.2023.104549
15. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. Computer **36**(1) (Jan 2003). https://doi.org/10.1109/MC.2003.1160055

16. Kobayashi, T., Bondu, M., Ishikawa, F.: Formal Modelling of Safety Architecture for Responsibility-Aware Autonomous Vehicle via Event-B Refinement. In: Chechik, M., Katoen, J.P., Leucker, M. (eds.) Formal Methods. pp. 533–549. Springer International Publishing, Cham (2023). https://doi.org/10.48550/arXiv.2401.04875
17. Leurent, E.: An Environment for Autonomous Driving Decision-Making. https://github.com/eleurent/highway-env (2018)
18. Leuschel, M., Vu, F., Rutenkolk, K.: Case Study: Safety Controller for Autonomous Driving on Highways – Specification document v3. https://abz-conf.org/case-study/abz25/ (2025)
19. Phan, D.T., Grosu, R., Jansen, N., Paoletti, N., Smolka, S.A., Stoller, S.D.: Neural Simplex Architecture. In: Lee, R., Jha, S., Mavridou, A., Giannakopoulou, D. (eds.) NASA Formal Methods. pp. 97–114. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-55754-6_6
20. Sha, L.: Using Simplicity to Control Complexity. IEEE Software **18**(4), 20–28 (2001). https://doi.org/10.1109/MS.2001.936213
21. Shalev-Shwartz, S., Shammah, S., Shashua, A.: On a Formal Model of Safe and Scalable Self-driving Cars (2018). https://doi.org/10.48550/arXiv.1708.06374
22. Vu, F., Dunkelau, J., Leuschel, M.: Validation of Reinforcement Learning Agents and Safety Shields with ProB. In: NASA Formal Methods: 16th International Symposium, NFM 2024, Moffett Field, CA, USA, June 4–6, 2024, Proceedings. p. 279–297. Springer-Verlag, Berlin, Heidelberg (2024). https://doi.org/10.1007/978-3-031-60698-4_16
23. Weyns, D.: Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective. Wiley (2020)
24. Weyns, D., Bencomo, N., Calinescu, R., Cámara, J., Ghezzi, C., Grassi, V., Grunske, L., Inverardi, P., Jézéquel, J., Malek, S., Mirandola, R., Mori, M., Tamburrelli, G.: Perpetual Assurances for Self-Adaptive Systems. In: Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers. Lecture Notes in Computer Science, vol. 9640, pp. 31–63. Springer (2013). https://doi.org/10.1007/978-3-319-74183-3_2