Abstract—The challenges in providing convincing arguments for safe and correct behavior of automated driving (AD) systems have so far hindered their widespread commercial deployment. Conventional development approaches such as testing and simulation are limited by non-exhaustive analysis, and can thus not guarantee safety in all possible scenarios. Formal methods can provide mathematical proofs that could be used to produce rigorous evidence to support the safety argument. This paper investigates the use of differential dynamic logic and the deductive verification tool KeYmaera X in the development of an AD feature. Specifically, this paper demonstrates how formal models and safety proofs of different design variants of a Decision & Control module can be used in the safety argument of an in-lane AD feature. In doing so, the assumptions and invariant conditions necessary to guarantee safety are identified, and the paper shows how such an analysis helps during the development process in requirement refinement and formulation of the operational design domain. Furthermore, it is shown how the performance of the different models is formally analyzed exhaustively, in all their allowed behaviors.

Index Terms—Automated driving, formal methods, hybrid systems, safety argument, theorem proving.

I. INTRODUCTION

AUTOMATED driving (AD) has many potential benefits [1], such as reducing road traffic accidents, improving energy efficiency, availing independent mobility to people who cannot or should not drive, etc. However, there are many barriers for the commercial deployment of full autonomy in road vehicles; particularly crucial and technically challenging is providing credible arguments for the safety of AD systems [2], [3].

Several approaches could be used to argue about AD safety [4]. A well-established industrial practice is to adopt a safety case approach where a convincing argument for the safety of the system is produced based on evidence. A safety case is used to show conformance to safety standards such as ISO 26262 [5], which addresses hazards due to malfunctioning behavior, and ISO/PAS 21448 [6], which addresses hazards due to unintended behavior. However, the high complexity of AD systems not only presents significant challenges but also makes it difficult to provide sufficient evidence to satisfy the safety objectives in the development activities recommended by such standards [2], [3], [7], [8].

A. Illustrative Example

To emphasize the challenges involved, consider an autonomous vehicle (hereafter referred as ego-vehicle) that offers in-lane unsupervised AD at speeds up to 100 km/h. This is realized using the simplified functional architecture in Fig. 1, where

1) Sense, perceives the environment and provides information such as the vehicle state, traffic state, etc.;
2) Decision & Control, decides on when and how to act (e.g. accelerating/steering commands); and
3) Actuate, executes the decisions using the respective actuators.

A safety argument for the above feature requires strong evidence for safe operation of the AD system in the infinitely many scenarios that can occur in the intended operational environment. The safety argument has to take into account several uncertainties that arise from the interaction of the AD system with other participants in the environment and also the complex interactions that stem from the different subsystems designed to realize the AD feature.

One way to tackle this challenge is to restrict the operational environment of the AD system through an operational design domain (ODD), defined in SAE J3016 [9] as “Operating conditions under which a given driving automation system or feature thereof is specifically designed to function [...].” The ODD limits the scope of the safety argument and development activities like hazard analysis and risk assessment (HARA) [5], requirement refinement, verification, etc. Even so, it is challenging to sufficiently identify the ODD such that it can provide unambiguous
The ego-vehicle shall maintain a safe distance to other objects in front such that a collision is at all times avoided.

As SG 1 is to be realized using Fig. 1, an implementation-independent functional safety requirement (FSR) necessary to achieve SG 1 could be:

**FSR 1:** Decision & Control shall at all times output a safe acceleration request to avoid collision with any object in front.

The successful verification of FSR 1 is crucial to fulfill SG 1 and consequently for the overall safety argument. FSR 1 requires the ego-vehicle to avoid collisions with other objects at all times. Obviously, this is desirable, but difficult to provide the necessary verification evidence due to the following challenges, among others:

a) **Infeasibility of testing:** An evidence based only on finitely many test cases is insufficient to demonstrate safety for the infinitely many operating scenarios that the ego-vehicle might encounter. Testing, irrespective of the level (e.g., unit, system) and type (e.g., field test, simulation) is infeasible to ensure AD safety [2], [4], [13]. Though a widespread industrial approach, testing can only provide evidence for the presence of bugs and cannot guarantee their absence. Therefore complementary techniques are needed.

b) **Requirement gaps:** The generality of FSR 1 presents difficulties to obtain subsequent implementation-specific safety requirements. Specifying unambiguous safety requirements at each phase of the development process is a significant challenge [2], [7], [8] that is further complicated by the presence of machine learning algorithms in the AD system. While a beneficial approach in this regard is to specify safety requirements in the form of invariants that describe system states required for safety [2], it remains a challenge to characterize such states.

Concretely, consider a case where the controller is to guarantee that the ego-vehicle stops at or before a critical position. This can, for instance, be a specification on the controller to fulfill FSR 1. Fig. 2 illustrates the ego-vehicle’s position and velocity on the position-velocity \((x, v)\) plane for a given set of inputs and parameter values. The dashed vertical line represents the critical position. The shaded region denotes the forbidden states where the requirement is violated. To the left of the critical position, the state-space is partitioned into the admissible states, denoted by the hatched region, from where the controller can guarantee that the forbidden states are avoided, and the inadmissible states, denoted by the white region, from where reaching the forbidden states cannot be avoided due to the system dynamics. Identifying the ODD assumptions and invariant conditions that characterize the inadmissible states is crucial to the verification of FSR 1, for obtaining implementation-specific safety requirements, and for the overall safety argument. In light of addressing these challenges, this paper demonstrates how formal methods can aid in the development of a safe AD system.

**B. Contributions**

Formal methods can provide mathematical proofs of safety and hence exhaustively verify and truly guarantee the absence of errors. Therefore, formal methods can complement other non-exhaustive methods like testing to provide the necessary evidence for the safety argument. This paper investigates how differential dynamic logic (dL) [14], [15], [16] and the associated deductive verification tool KeYmaera X [17] can be used in the safety argument of an in-lane AD feature.

Deductive verification and the expressiveness of dL enable symbolic verification of parametric systems. Furthermore, the logic dL supports nondeterministic modeling which can be used to capture uncertainties in the operational environment and also the complex interactions involved. Thus, the models cover infinitely many in-lane forward driving scenarios and several concrete controller implementations in a succinct way. Therefore, the safety proofs provide mathematical evidence that the models fulfill the safety requirements in all such cases. Additionally this work shows how the formulas of dL used to specify safety requirements can be represented in the form of assume-guarantee requirements to permit compositional reasoning based on principles of contract-based design [18], thereby reducing design and verification complexity.

To summarize, this paper:

1) demonstrates how formal modeling and verification based on dL can be used to provide evidence for the safety argument of an in-lane AD feature. To this end, it is shown how the decision & control module for the AD feature is modeled and subsequently refined by adding more aspects to describe different design variants (Section IV). The models employ a safety controller architecture to supervise nominal AD functions and thus guarantee safety even if hard-to-verify (e.g., learning-based) algorithms are used to realize the nominal AD functions. Additionally, it is shown how subtle modeling changes can lead to safety-critical edge cases;

2) formalizes (informal and textual) safety requirements in the form of assume-guarantee requirements in dL in addition to formulating the assumptions and the invariant conditions required to guarantee safety of the AD feature (Section IV);
3) shows how formal proofs can be used to analyze the performance (with respect to a defined metric) of the different models, in all behaviors allowed by the respective model (Section V); this, in contrast to conventional approaches that typically involve independent analysis of different (non-exhaustive) sets of scenarios;
4) illustrates how the formal analysis helps in the requirement refinement during a practical industrial development process (Section VI).

II. PROBLEM FORMULATION

The focus of this paper is the safety verification of the Decision & Control module for an in-lane AD feature. While the simplified functional architecture in Fig. 1 is representative of any AD feature, the Decision & Control needs to be refined to concretely formulate the verification problem.

A typical driving task can be divided into three levels of decision-making [19]: strategic (e.g. route planning over long time horizon), tactical (e.g. maneuvering over a few seconds), and operational (e.g. speed control on milliseconds level). For an AD feature, SAE J3016 [9] requires the AD system to completely perform the dynamic driving task (DDT), defined as “all of the real-time operational and tactical functions required to operate a vehicle in on-road traffic, excluding the strategic functions [...]”. Accordingly, this work considers only tactical and operational levels of the Decision & Control module.

While performing the DDT, the ego-vehicle is required to handle a variety of in-lane scenarios like maintaining a safe distance to a lead vehicle, stopping for an obstacle, etc., and is subjected to different constraints to ensure safety, comfort, etc. This driving task could be solved by any algorithm (e.g. feedback control law, reinforcement learning), some of which might be hard to analyze and verify. One way to make the safety verification tractable is to separate the nominal functions from the safety functions by means of the Nominal Controller and the Safety Controller as shown in Fig. 3. The Nominal Controller represents any algorithm solving the nominal driving task and requests a nominal acceleration $a_n$. The Safety Controller ensures that only safe decisions are communicated to the Vehicle Control by evaluating $a_n$ and calculating a safe acceleration $a_s$, thereby satisfying the safety requirement. Thus, the safety verification can be reformulated as a verification problem of a simpler component, the Safety Controller.

In addition to $a_n$, the Safety Controller receives information about the vehicle state from the Sense module and information about the safety constraint given by a critical position $x_c$ and a critical velocity $v_c$ (see Section IV) from the Situation Assessment function block. The safety constraint is formulated based on the safety requirement and thus $a_s$ calculated by the Safety Controller is considered safe if it fulfills the safety constraint $\langle x_c, v_c \rangle$. The Vehicle Control then, takes $a_s$ as a reference and calculates trajectories for vehicle motion to solve the operational function of the DDT. Finally, these motion trajectories are executed by the Actuate module through the various actuators in the ego-vehicle.

The architecture in Fig. 3 with a separation between the Nominal Controller and the Safety Controller is conceptually similar to the Simplicit architecture [20] and its variants [21], [22]. Such an architecture permits to abstract away from the possibly complex design of the Nominal Controller, and formally verify that $a_s$ always respects the safety constraint by reasoning about the decision-making in the Safety Controller. Moreover, specifying a safety requirement like FSR 1 using a safety constraint from a separate functional block provides the flexibility to dynamically calculate constraints for a variety of situations. Furthermore, this architecture also enables modular reasoning and therefore the verification approach discussed in this paper can easily be adapted to other AD features outside the scope of in-lane AD. Though this approach makes the safety verification efficient and tractable, one might rightly argue that certain assumptions have to be made about the Nominal Controller and possibly other functions within Decision & Control to guarantee that the safety requirements are satisfied. The rest of this paper deals with this safety verification problem where, the Decision & Control module of Fig. 3 is formally verified, and in doing so, the assumptions and invariant conditions required to guarantee safety are identified.

III. PRELIMINARIES

The logic $\mathfrak{dL}$ supports the specification and verification of hybrid systems, that is, mathematical models of systems that combine discrete dynamics (behavior that changes discretely) with continuous dynamics (behavior that changes continuously with time). This makes $\mathfrak{dL}$ particularly suitable for the modeling and verification of AD systems since it can reason about continuous state variables (e.g. position, velocity) and discrete decisions (e.g. to brake, to accelerate).

To model hybrid systems, $\mathfrak{dL}$ has the notion of hybrid programs (HP) that consist of different program statements, including differential equations to describe continuous behavior. HPs are defined by the following grammar, where $\alpha$, $\beta$ are HPs, $x$
is a variable, $e$ is a term\(^1\) possibly containing $x$, and $P, Q$ are formulas of first-order logic of real arithmetic:\(^2\)

$$\alpha, \beta ::= x := e | x := \ast | \gamma P | x' = f(x) & Q | \alpha \cup \beta | \alpha; \beta | \alpha^*$$

A summary of the program statements in HP and their informal semantics [16], is given in Table I.

Each HP $\alpha$ is semantically interpreted as a reachability relation $\rho(\alpha) \subseteq S \times S$, where $S$ is the set of states and a state $s \in S$ is defined as a mapping from the set of variables to $\mathbb{R}$. The test action $?P$ has no effect in a state where $P$ is true. If, however, $P$ is false when $?P$ is executed, then the current execution of the HP aborts, meaning that the entire current execution is removed from the set of possible behaviors of the HP. Test actions are often used together with non-deterministic assignment, like $a_n := \ast$; $\gamma P$ where $a_n$ is assigned an arbitrary value, which is then tested to be within the bounds $-a_n^{min}$ and $a_n^{max}$. This expresses that $a_n$ is outside those bounds, that branch of execution is aborted. This can model that some external component/environment chooses $a_n$ to be within the given bounds. Furthermore, test actions can be combined with sequential composition and the choice operator to define standard if-statements:

$$\text{if } (P) \alpha \text{ else } \beta \equiv (\gamma P; \alpha) \cup (\gamma \neg P; \beta) \quad (1)$$

HPs model continuous dynamics as $x' = f(x) & Q$, which describes the continuous evolution of $x$ along the differential equation system $x' = f(x)$ for an arbitrary duration (including zero) within the evolution domain constraint $Q$. The evolution domain constraint are first-order formulas that restrict the continuous evolution within certain bounds. $x'$ denotes the time derivative of $x$, where $x$ is a vector of variables and $f(x)$ is a vector of terms of the same dimension.

The nondeterministic actions (assignment $x := \ast$, choice $\alpha \cup \beta$, and repetition $\alpha^*$) help address two critical aspects in the safety verification:

1) they can describe unknown behavior, which is typically the case in modeling the highly uncertain environment for AD systems;
2) they can abstract away implementation specific details and thus reduce the dependency of the proof on such details.

For example, to reason about the correctness of the Safety Controller, the nominal acceleration $a_n$ can be modeled with a nondeterministic assignment together with a test action as described above. Such a model could describe the behavior of any Nominal Controller implementation irrespective of the algorithm used and therefore makes the analysis independent of implementation changes.

The formulas of $\text{dL}$ include formulas of first-order logic of real arithmetic and the modal operators $[\alpha]$ and $\langle \alpha \rangle$ for any HP $\alpha$ [15], [16]. The $\text{dL}$ formula $[\alpha] \phi$ expresses that all non-aborting runs of HP $\alpha$ (i.e., the runs where all test actions are successful) lead to a state in which the $\text{dL}$ formula $\phi$ is true. The $\text{dL}$ formula $\langle \alpha \rangle \phi$ says that there exists some non-aborting run leading to a state where $\phi$ is true. $\langle \alpha \rangle \phi$ is the dual to $[\alpha] \phi$, defined as $[\alpha] \phi \equiv \neg [\neg \alpha] \neg \phi$. The formulas of $\text{dL}$ are defined by the following grammar ($\phi, \psi$ are $\text{dL}$ formulas, $e, \tilde{e}$ are terms, $x$ is a variable, $\alpha$ is a HP):\(^3\)

$$\phi, \psi ::= e = \tilde{e} | e \geq \tilde{e} | \neg \phi | \phi \land \psi | \forall x \phi | [\alpha] \phi \quad (2)$$

The operators $>, \leq, <, \lor, \land, \lor, \exists x$ are defined using combinations of the operators in (2).

In our context of AD, if $M$ is an HP describing the Decision & Control module and $\text{guarantee}$ is the safety requirement to be verified, to specify the correctness of $M$, we use a $\text{dL}$ formula of the form:

$$(\text{init}) \rightarrow [M] (\text{guarantee}), \quad (3)$$

which expresses that if the initial conditions described by formula $\text{init}$ are true, then all (non-aborting) runs of $M$ only lead to states where the formula $\text{guarantee}$ is true. To prove $\text{dL}$ formulas, we use the interactive theorem prover KeYmaera X [17], which implements a verification technique for $\text{dL}$ [15], [16]. KeYmaera X takes a $\text{dL}$ formula as an input and proves it by successively decomposing it into several sub-goals according to the proof rules of $\text{dL}$ [15], [16]. To prove properties of loops, like the property $(\text{init}) \rightarrow [\alpha^*] (\text{guarantee})$, KeYmaera X uses loop invariants to inductively reason about all (non-aborting) executions of the loop through the loop invariant proof rule [16]. The loop invariant rule uses some (inductive) loop invariant $\xi$ to prove the above formula by proving three separate formulas:

i) $(\text{init}) \rightarrow \xi$
ii) $\xi \rightarrow [\alpha] \xi$
iii) $\xi \rightarrow (\text{guarantee})$

Though KeYmaera X provides some level of proof automation [17], [23], user interaction (e.g. providing loop invariants) is often needed to prove complex models.

### IV. Models and Proofs

As discussed in Section II, we formally analyze the safety of the Decision & Control module by reasoning about the decision-making in the Safety Controller, and in doing so,
formulate the assumptions and invariant conditions required to guarantee safety. The objective of the Safety Controller is to calculate a safe acceleration value \(a_s\) that always fulfills the safety constraint given by the pair \(\langle x_c, v_c \rangle\), the critical position and critical velocity, respectively. The requirement to guarantee safety of the ego-vehicle is to not have a velocity higher than \(v_c\) at or beyond \(x_c\).

The ego-vehicle’s (longitudinal)\(^3\) motion is described by the continuous time kinematic equations:

\[
\frac{dx}{dt} = v, \quad \frac{dv}{dt} = a,
\]  

(4)

where position \(x\) and velocity \(v\) are the state variables and the acceleration \(a\) is piece-wise constant. Fig. 4 illustrates an example simulation of the ego-vehicle’s motion model.

Furthermore, we consider four system parameters: maximum acceleration limit of the Nominal Controller \(a_n^{\text{max}} > 0\), maximum braking\(^4\), limit of the Nominal Controller and the Safety Controller given by \(a_n^{\text{min}} > 0\) and \(a_s^{\text{min}} > 0\) respectively, and sampling time \(T > 0\). Since the Nominal Controller is subject to different constraints (e.g. comfort constraints) during nominal driving, its braking capability is less than the vehicle’s maximum braking capability. In contrast, the Safety Controller can use the vehicle’s maximum braking capability and can therefore brake harder than the Nominal Controller.

In Models 1–5, the respective plant ((7), (18), (27), (39), (49)) models the continuous dynamics together with the evolution domain constraint. The ego-vehicle’s motion described in (4) is modeled as \(x' = v, \, v' = a\), where \(a\) is the safe acceleration from the Safety Controller. The evolution domain constraint \(v \geq 0\) restricts the continuous evolution to only non-negative velocities. In addition, the plant models a clock variable \(\tau\) that evolves along the differential equation \(\tau' = 1\), is bound by the domain constraint \(\tau \leq T\), and is set to \(\tau = 0\) before every evolution of the ego-vehicle’s motion. Intuitively, \(\tau\) represents the controller execution/sampling time. The constraint \(\tau \leq T\) accounts for non-periodic sampling as it allows evolution for any amount of time not longer than \(T\). In every execution loop of \([\text{env}; \text{ctrl}; \text{plant}]\), first \(\text{env}\) and \(\text{ctrl}\) get executed instantaneously, the clock is reset, and then the plant evolves for at most \(T\) time before the loop either repeats or terminates.

The rest of this section describes the models and proofs for different design variants of the Decision & Control module. The models differ in the decision logic used and the differences are highlighted in boldface. We first prove safety of Model 1, with a conservative safety controller and the critical velocity \(v_c = 0\). This model is then generalized to \(v_c \geq 0\) (Model 2). Then, we extend the proofs to models (Models 3–5) with different threat metrics [24], [25], [26]. Furthermore, we also remark on the implications of some modeling choices on the safety argument.

A. Model 1: Conservative With \(v_c = 0\)

In Model 1, the DL formula (5) refines \(M\) of (3) as a nondeterministic repetition of three sequentially composed HPs to represent a typical controller-plant model: \(\text{env} \circ \text{ctrl} \circ \text{plant}\). This model is then generalized to \(\text{env} \circ \text{ctrl} \circ \text{plant}\). Then, we extend the proofs to models (Models 3–5) with different threat metrics [24], [25], [26]. Furthermore, we also remark on the implications of some modeling choices on the safety argument.

\[\begin{align*}
\text{env} & ((x', v') = (\text{env}(x, v) \land v < v_c)) \\
\text{ctrl} & (v > 0) \\
\text{plant} & ((x, v) = (x', v'))
\end{align*}\]

Only in-lane scenarios are dealt with, so the terms position, velocity and acceleration describe longitudinal vehicle motion, unless otherwise noted.

\(^4\)Braking is modeled explicitly with negative acceleration (e.g. \(a_s := -a_s^{\text{min}}\)).
by the sum of the distance traveled from the current state by accelerating with $a_n^{\text{max}}$ for time interval $t$ and from there on, the distance traveled by braking with $a_s^{\text{min}}$ until $v = 0$ is reached, i.e., the ego-vehicle is completely stopped.

The distance traveled (change in position $x$) and the change in velocity $v$ due to a constant acceleration $a$ during the time interval $t$ can be computed from the solution to the differential equations (4) as (for initial values $x_0$ and $v_0$):

$$x(t) = x_0 + vt + \frac{at^2}{2}$$  \hspace{1cm} (14)

$$v(t) = v_0 + at$$  \hspace{1cm} (15)

The condition $ok$ (9) checks whether the distance between the critical position $x_c$ and current position $x$ is at least $msd_1(T)$, i.e., the minimal safe distance for one execution cycle $T$, the maximum time interval between two decisions.

The HP env models the behavior of the Situation Assessment (gives $\langle x_c, v_c \rangle$) and the Nominal Controller (gives $a_n$). The Situation Assessment module calculates the safety constraint $\langle x_c, v_c \rangle$ based on the safety requirement and is therefore dependent on other objects in the driving lane. Since $v_c = 0$ in this simplified case, we only consider $x_c$ in env. While it is desirable to prove that ctrl fulfills guarantee for all possible values of $x_c$, recall from Fig. 2 that such a proof cannot be obtained. For instance, no controller can guarantee safety from a state that already violates the safety constraint. However, it should be able to fulfill guarantee in all behaviors where it is practically feasible to act in a safe manner. The formula (11) describes this intuition where env is modeled as a nondeterministic choice with two branches. The first branch with a nondeterministic assignment followed by the test $? (x_c - x \geq msd_1(0))$ only admits behaviors where the distance between $x_c$ and $x$ is at least the minimal safety distance for zero duration $msd_1(0)$, i.e., the Safety Controller can fulfill the safety constraint by maximal braking from the current state. The inequality $x_c - x \geq msd_1(0)$ characterizes the admissible region (see Fig. 2) for this model and formulates the assumption on the Situation Assessment module to guarantee safety. The second branch $?true$ in (11) models a skip action where env can keep the current value of the constraint. Thus, in every execution of the loop, env can nondeterministically choose to either update $x_c$ based on the test for admissible behavior or keep $x_c$ unchanged by performing a skip action. The Nominal Controller described by (12) can nondeterministically output any value within the bounds, i.e., $(-a_s^{\text{min}} \leq a_n \leq a_n^{\text{max}})$.

The formula init (13) specifies the initial conditions for Model 1; the four symbolic system parameters are positive, the velocity $v$ is non-negative, and that the system starts within the admissible region $x_c - x \geq msd_1(0)$. The DL formula (5) is proved using the interactive theorem prover KeYmaera X. Recall from Section III that KeYmaera X requires a loop invariant to prove DL formulas with loops like (5) using the loop invariant proof rule [16].

**Theorem 1:** Model 1 for the Decision & Control module described by (7)–(12) guarantees to provide a safe acceleration request with respect to the safety constraint $\langle x_c, v_c \rangle$, with $v_c = 0$, as expressed by the DL formula (5).

**Proof:** Theorem 1 is proved [27] in KeYmaera X. The proof uses the loop invariant $\zeta \equiv x_c - x \geq msd_1(0)$. □

**Remark 1:** Though only $v_c = 0$ is considered in Model 1, the use of nondeterminism in the model enables a safety proof that covers a wide variety of designs. For instance, the nondeterministic assignments in (11) and (12) make the proof independent of the implementation of the Situation Assessment and the Nominal Controller (Fig. 3). Furthermore, symbolic bounds on the system parameters make the proof cover infinitely many design variants.

**Remark 2:** Note that guarantee (6) formalizes the safety requirement using the safety constraint $\langle x_c, v_c \rangle$, with $v_c = 0$. Theorem 1 verifies that ctrl provides a safe acceleration with respect to this constraint. However, the proof depends on the assumptions and invariant conditions described in Model 1. In Section VI, we illustrate how a safety requirement like FSR 1 is formalized and refined using Model 1 and its proof.

**B. Model 2: Conservative With $v_c \geq 0$**

Model 2 extends Model 1 to allow $v_c \geq 0$. This change is described by modifications to the guarantee (17), env (22), and init (24) to include $v_c$. Similarly, the formula for the minimal safety distance $msd_2(t)$ (21) is adjusted to reflect the generic case while following the same worst-case reasoning as Model 1. Here, $msd_2(t)$ is given by the sum of the distance traveled from the current state by accelerating with $a_n^{\text{max}}$ for time interval $t$ and from there on, the distance traveled to reach $v_c \geq 0$ (instead of $v_c = 0$ as in Model 1).

**Theorem 2:** Model 2 for the Decision & Control module, described by (18)–(23) guarantees to provide a safe acceleration request with respect to the safety constraint $\langle x_c, v_c \rangle$ as expressed by the DL formula (16).
Model 2: Conservative with $v_c \geq 0$.

\begin{equation}
\text{(init)} \rightarrow [(\text{env}; \text{ctrl}; \text{plant})^*] \text{(guarantee)}
\end{equation}

\begin{align}
\text{guarantee} & \triangleq (x \geq x_c \rightarrow v \leq v_c) \quad (16) \\
\text{plant} & \triangleq \tau := 0; \ x' = v, v' = a, \tau' = 1 & v \geq 0 \land \tau \leq T \\
proofrule{ctrl} & \triangleq \text{if (ok) } a_s := a_n \text{ else } a_s := -a_s^{\text{min}} \\
\text{ok} & \triangleq x_c - x \geq \text{msd}_2(T) \\
\text{msd}_2(t) & \triangleq vt + \frac{a_n^{\text{max}}t^2}{2} + \frac{(v + a_n^{\text{max}}t)^2 - v^2_c}{2a_s^{\text{min}}} \quad (20) \\
\text{env} & \triangleq (x_c := *; \ v_c := *; ?(v_c \geq 0 \land x_c - x \geq \text{msd}_2(0) \cup ?\text{true}) \\
\text{a}_n & := *; ?(a_n^{\text{min}} \leq a_n \leq a_n^{\text{max}}) \\
\text{init} & \triangleq a_s^{\text{min}} > 0 \land a_n^{\text{max}} > 0 \land a_n^{\text{min}} > 0 \land T > 0 \\
& \land v \geq 0 \land v_c \geq 0 \land x_c - x \geq \text{msd}_2(0) \quad (24)
\end{align}

\begin{proof}
Theorem 2 is proved [27] in KeYmaera X using the loop invariant $z \equiv v_c \geq 0 \land x_c - x \geq \text{msd}_2(0)$. \qed
\end{proof}

C. Model 3: Permissive With $v_c = 0$

The decision-making in the two models discussed so far is based on a threat metric determined from worst-case assumption of the ego-vehicle behavior. While provably safe, such worst-case reasoning often leads to a conservative design. The permissive models described by Model 3 and Model 4 are based on a threat metric that relaxes the worst-case assumption. This section discusses the case where $v_c = 0$ and Section IV-D extends it for the generic case $v_c \geq 0$.

In comparison to Model 1, the threat metric used to assess $a_n$ is changed according to the new relaxed assumption. Intuitively, while calculating $\text{msd}_3(t)$, the worst-case behavior of accelerating with $a_n^{\text{max}}$ for time interval $t$ is replaced with the actual behavior of accelerating with the requested $a_n$ for the time interval $t$. This change is reflected in Model 3 by replacing $a_n^{\text{max}}$ with $a_n$ in (10) to give:

\begin{equation}
\text{msd}_3(t) \triangleq vt + \frac{a_n^{\text{max}}t^2}{2} + \frac{(v + a_n^{\text{max}}t)^2 - v^2_c}{2a_s^{\text{min}}} \quad (35)
\end{equation}

The Safety Controller decides whether $a_n$ is ok by checking if $x_c - x \geq \text{msd}_3(T)$ is true. From (35), note that $\text{msd}_3(T)$ is given by the sum of the distance traveled from the current state due to $a_n$ for time interval $T$, and from there on the distance traveled by braking with $a_s^{\text{min}}$ until zero velocity is reached. Since $-a_s^{\text{min}} \leq a_n \leq a_n^{\text{max}}$, the ego-vehicle can either accelerate or brake during a given time interval depending on the requested $a_n$. However, when the ego-vehicle brakes ($-a_s^{\text{min}} \leq a_n < 0$), it is possible that zero velocity is reached during interval $T$ but not necessarily after $T$ time. In such edge cases, $\text{msd}_3(T)$ determined from (35) will lead to incorrect and unsafe decisions and therefore is not sufficient to fulfill the guarantee (26).

An example simulation of such an edge case is given in Fig. 5. Though the safety constraint is violated during time interval $T$ for some requested $a_n < 0$, the test condition $x_c - x \geq \text{msd}_3(T)$ incorrectly decides the requested $a_n$ to be ok using $\text{msd}_3(T)$ determined from (35). To account for such scenarios, $\text{msd}_3(t)$ is split into two cases depending on whether the velocity after time interval $t$ due to $a_n$ is non-negative or not, as formulated in (30). In the second case, when $v + a_n t < 0$ due to $-a_s^{\text{min}} \leq a_n < 0$, a choice of $a_n$ is considered ok if braking with $a_n$ to a complete stop is sufficient to satisfy the constraint, as described by $x_c - x \geq -\frac{v_c^2}{2a_s^{\text{min}}}$. Else, $a_s^{\text{min}}$ is set as the safe acceleration value.
Theorem 3: Model 3 for the Decision & Control module, described by (27)–(33) guarantees to provide a safe acceleration request with respect to the safety constraint \( (x_c, v_c) \), with \( v_c = 0 \), as expressed by the DL formula (25).

Proof: Theorem 3 is proved [27] in KeYmaera X using the loop invariant \( \xi = x_c - x \geq msd_3(0) \).

Remark 3: The edge case encountered in Model 3 highlights the need for reasoning about intermediate states to accurately prove safety throughout all executions of the model. In \( plant (27) \), the implicit nondeterminism introduced by \( \tau \leq T \) allows evolution of any duration \( \tau \) (including 0) that satisfies the domain constraint. Therefore, Theorem 3 shows that \( guarantee \) is fulfilled throughout all possible executions of the model. However, modifying \( plant (27) \) to include a test as:

\[
\tau := 0; x' = v, v' = a_s, \tau' = 1 \& v \geq 0 \& \tau \leq T \Rightarrow (\tau = T);
\]

makes (25) provable even with incorrect \( msd(t) \) (35), since (36) allows only evolution of exactly \( \tau = T \) duration in the model and thus requires \( guarantee \) to hold only in states reached at the end of evolution for precisely \( T \) duration.

D. Model 4: Permissive With \( v_c \geq 0 \)

The permissive model where \( v_c \geq 0 \) is similar to Model 2 in Section IV-B except for the threat metric \( msd_4(t) \) (42). Here, since \( msd_4(t) \) is determined based on actual behavior and not worst-case behavior of the ego-vehicle, it is split into two cases for account for edge cases described in the previous section and illustrated in Fig. 5.

Theorem 4: Model 4 for the Decision & Control module, described by (39)–(45) guarantees to provide a safe acceleration request with respect to the safety constraint \( (x_c, v_c) \) as expressed by the DL formula (37).

Proof: Theorem 4 is proved [27] in KeYmaera X using the loop invariant \( \xi = v_c \geq 0 \& x_c - x \geq msd_4(0) \).

E. Model 5: Required Acceleration as Threat Metric

In the previous models, the decision-making is based on a threat metric defined in the distance domain. In practice, threat metrics can be defined in other domains like time, acceleration, etc. and the choice of threat metric in the decision-making algorithm is influenced by factors such as application, performance, and robustness [24], [25], [26]. In this section, we show how the modeling approach in this paper can easily be adapted to analyze designs with different threat metrics. To this end, we show how Model 3 can be reformulated with a threat metric in the acceleration domain \( a_{\text{req}} \) defined as the longitudinal acceleration required to fulfill the safety constraint.

Though \( a_{\text{req}} \) can be defined in several ways [24], \( a_{\text{req}} \) in (52) is defined similar to (30) in Model 3 using the same assumption for the ego-vehicle behavior since the intention here is to show a reformulation of the threat metric in the acceleration domain. However, the decision-making in Model 3 and Model 5 differ in the threshold used for the Safety Controller’s intervention and this difference is further discussed in Section V. Furthermore, while no assumption on the relation between \( a_{\text{min}}^{\text{req}} \) and \( a_{\text{min}}^{\text{req}} \) was required in Models 1–4, \( init (57) \) requires \( a_{\text{min}}^{\text{req}} < a_{\text{min}}^{\text{req}} \) (or \( a_{\text{min}}^{\text{req}} \geq a_{\text{min}}^{\text{req}} \)) to fulfill \( guarantee \) (48) as \( a_{\text{min}}^{\text{req}} \) is used as a threshold for comparison in the \( ok \) condition (51).

Theorem 5: Model 5 for the Decision & Control module, described by (49)–(56) guarantees to provide a safe acceleration request with respect to the safety constraint \( (x_c, v_c) \), with \( v_c = 0 \), as expressed by the DL formula (47).

Proof: Theorem 5 is proved [27] in KeYmaera X using the loop invariant \( \xi = v_c \geq 0 \& x_c - x \geq msd_4(0) \).

Remark 4: Note that the formulation (52) can be extended to describe yet another threat metric like \( Brake \) Threat Number [28], defined as the ratio of the longitudinal acceleration required to the maximum longitudinal acceleration. Though the decision function in all the models are developed based on solutions to (4) and assumptions on the safe behavior of the ego-vehicle, the models differ in the decision-making when it comes to the Safety Controller’s interventions and this section highlights how different design variants can be modeled as hybrid programs. The main effort in the verification process consists in identifying appropriate assumptions and loop invariants.

V. MODEL PERFORMANCE ANALYSIS

In Section IV, theorems 1, 3, and 5 verified that the corresponding models provide a safe acceleration request with
Model 5: \( a^{\text{req}} \) as threat metric.

\[
\begin{align*}
\text{(init)} & \rightarrow [(\text{env}; \text{ctrl}; \text{plant})^*] \,(\text{guarantee}) \\
\text{guarantee} & \triangleq (x \geq x_c \rightarrow v = 0) \\
\text{plant} & \triangleq \tau := 0; x' = v, t' = a_s, t'' = 1 \land v \geq 0 \land \tau \leq T \\
\text{ctrl} & \triangleq \text{if} \,(\text{ok}) \ a_s := a_n \text{ else } a_s := -a_s^{\text{min}} \\
\text{ok} & \triangleq a_{\text{req}}^{\text{th}} \geq a_{\text{th}} \\
\text{a}_{\text{th}} & \triangleq \begin{cases} 
-a_s^{\text{min}} & \text{if } v + a_n T \geq 0 \\
-a_n & \text{otherwise}
\end{cases} \\
\text{env} & \triangleq (x_c := x; \ ? \ (a_0^{\text{req}} \geq -a_s^{\text{min}}) \cup \text{?true}); \\
& \quad a_n := x; \ ? \ (-a_s^{\min} \leq a_n \leq a_s^{\max}) \\
\text{a}_0^{\text{req}} & \triangleq \frac{v^2}{2(x_c - x)} \\
\text{init} & \triangleq a_s^{\min} > 0 \land a_s^{\max} > 0 \land a_s^{\min} > 0 \land T > 0 \\
& \land v \geq 0 \land a_n^\min < a_n^\min \land a_0^{\text{req}} \geq -a_s^{\min} \\

\end{align*}
\]

In Model 1, the minimal safety distance for intervention is given by

\[
\text{msd}_1 \triangleq v T + \frac{a_n^{\max} T^2}{2} + \frac{(v + a_n^{\max} T)^2}{2a_s^{\min}}
\]

and depends on the velocity \( v \) and the system parameters \( a_n^{\max}, a_s^{\min} \), and \( T \). Similarly, for Model 3 and Model 5, \( \text{msd}_3 \) and \( \text{msd}_5 \) can be obtained from (30) and (52) respectively. Though (52) uses \( a_{\text{req}}^{\text{th}} \) as the threat metric, it is straightforward to reformulate it to obtain a safety distance as discussed in Section IV-E. With the minimal safety distances for intervention obtained, the relation between the models for all parametric combinations can be captured by the first-order logic formula of real arithmetic:

\[
\forall \begin{bmatrix}
\text{a}_{n}^{\min} \\
\text{a}_{n}^{\max} \\
\text{T} \\
v \\
\text{a}_{n}
\end{bmatrix} \rightarrow \left( \text{msd}_1 \geq \text{msd}_3 \land \text{msd}_3 \geq \text{msd}_5 \right)
\]

Theorem 6: The Safety Controller in Model 3 uses a smaller safety distance for intervention compared to the Safety Controllers in Model 1 and Model 5 as expressed by (59).

Proof: Theorem 6 is proved [27] in KeYmaera X.

Remark 5: Note that no loop invariant was necessary to prove Theorem 6 since we only reason about a first-order logic formula of real arithmetic without a hybrid program.

From Theorem 6, we can conclude that the Safety Controller in Model 3 does not intervene earlier than the controllers in Model 1 and Model 5 to guarantee safety and therefore performs better. It is indeed possible to arrive at the same conclusion by analytically deriving the relation between the different safety distances. Of course, such a manual approach does not scale in practice. However, obtaining a formal machine checked proof as discussed in this section scales well to reason about different designs and to compare them.

In all the models discussed so far, the Safety Controller intervenes with maximal braking, \( a_s := -a_s^{\min} \). Though proven safe, it might not always be necessary to intervene with maximal braking in order to satisfy the constraint. Certainly, it is possible to change how the intervention is made in the models. For instance, if \( a_{\text{req}} \) is the acceleration required to satisfy the constraint at any given point, a modification to \( \text{ctrl} \) as shown in (60) describes that, as proved [27] by KeYmaera X, the Safety Controller can intervene with any acceleration value bounded by \( a_s^{\min} \) and \( a_{\text{req}}^{\text{th}} \):

\[
\text{ctrl} \triangleq \text{if} \,(\text{ok}) \ a_s := a_n \text{ else } a_s := v; \\
?(-a_s^{\min} \leq a_s \land a_s \leq a_{\text{req}}^{\text{th}})
\]

One way to conduct such an analysis is to simulate the models in different sets of scenarios (e.g. Fig. 4) and compare them with a suitable performance metric. The shortcoming with such an approach is the intractability to compare all possible scenarios. An alternative approach is to obtain a formal machine checked proof about the relation between different models for all parametric combinations. For instance, the condition \( \text{ok} \) (9), (29), and (51) determines when the Safety Controller intervenes in models 1, 3, and 5, respectively. A Safety Controller that intervenes as late as possible and still guarantees safety is certainly a preferable choice for a good performance. Therefore, an obvious choice is to use the minimal safety distance to compare the models.

respect to \( (x_c, v_c) \) where \( v_c = 0 \). Though all three models are proved safe, they differ in their decision-making and hence have different performance. In all the models, the Safety Controller, using a threat metric, assesses whether the nominal acceleration \( a_n \) compromises safety and if so, intervenes with maximal braking. Consequently, the performance depends on when and how the intervention is made. While an overly conservative Safety Controller that intervenes often with maximal braking can guarantee safety, it also possesses a risk of limited user acceptance. Therefore, it is valuable to analyze the performance of the Safety Controller to choose a good design.

One way to conduct such an analysis is to simulate the models in different sets of scenarios (e.g. Fig. 4) and compare them with a suitable performance metric. The shortcoming with such an approach is the intractability to compare all possible scenarios. An alternative approach is to obtain a formal machine checked proof about the relation between different models for all parametric combinations. For instance, the condition \( \text{ok} \) (9), (29), and (51) determines when the Safety Controller intervenes in models 1, 3, and 5, respectively. A Safety Controller that intervenes as late as possible and still guarantees safety is certainly a preferable choice for a good performance. Therefore, an obvious choice is to use the minimal safety distance to compare the models.
Clearly, modeling the intervention with nondeterminism as in (60) covers different variations of controller implementation with the same proof. Section IV described how $\mathcal{DL}$ is used to formally analyze the Decision & Control module for an in-lane AD feature, which was the primary goal. In this section is shown how KeYmaera X can be used to compare the verified models to aid in the design and development of the AD feature.

VI. REQUIREMENT REFINEMENT

In this section, we discuss how $\mathcal{DL}$ can be used to refine the requirements for the components interacting with the Decision & Control module. Theorems 1–5 of Section IV verified that the respective models provide a safe acceleration request with respect to the safety constraint. The nondeterminism in the models verifies a wide range of concrete implementations. However, certain assumptions are included in the models to prove the guarantee in each case. Broadly, the assumptions in the models are described in three ways:

i) bounds on the system parameters,

ii) assumptions on the interacting component/environment behavior, and

iii) evolution domain constraint.

These assumptions have resulted from the formal analysis of the Safety Controller and here, we show how such insights are used to formalize safety requirements for all the components in the Decision & Control module and also help to identify relevant ODD conditions.

Consider the functional safety requirement, FSR 1 in Section I-A. Of course, FSR 1 can be formulated using the safety constraint pair, $\langle x_c, v_c \rangle$ and thus Theorem 1 verifies FSR 1 for Model 1 in Section IV-A. Specifically in Model 1, the formulas $\text{init}$ (13) and $\text{plant}$ (7) include the assumptions on the system parameters; $\text{env}$ (11)–(12) and $\text{plant}$ (7) include assumptions on the components interacting with the Safety Controller; and finally the $\text{plant}$ (7) includes the evolution domain constraint. A straightforward consequence of the domain constraint $v \geq 0$ is that the evolution of the $\text{plant}$ would stop before reaching negative velocity, thus the ego-vehicle does not travel backwards. Naturally, such constraints can be used to refine the ODD for the AD feature as the safety guarantee clearly does not hold in situations where the ego-vehicle might possibly travel backwards, e.g. road geometries with high slope and less friction. Furthermore, the assumptions on the interacting components obtained from Model 1 can be used to further refine FSR 1 as:

FSR 1.1: The Nominal Controller shall output a nominal acceleration $a_n$ such that $(-a_n^{\text{min}} \leq a_n \leq a_n^{\text{max}})$.

FSR 1.2: The Situation Assessment shall output a critical position $x_c$ for a given ego-vehicle position $x$, velocity $v$, and maximum braking capability $a_v^{\text{min}}$ such that $x_c - x \geq \frac{v^2}{2a_v^{\text{min}}}$.

FSR 1.3: The Safety Controller shall at all times output a safe acceleration value $a_s$ to avoid a collision with any object in front if FSR 1.1 and FSR 1.2 are met.

FSR 1.4: The Vehicle Control shall always control the ego-vehicle according to the safe acceleration request $a_s$ to avoid a collision with any object in front.

The safety requirements thus obtained can be used in the subsequent analysis of the respective components using $\mathcal{DL}$. For instance, one conceivable but naïve algorithm for the Situation Assessment component is to sort the objects in front of the ego-vehicle and select the position of the closest object $x_l$ to obtain the critical position as

$$sa \triangleq x_c := x_l + d$$

where parameter $d$ denotes an admissible separation between the ego-vehicle and the object in front when the ego-vehicle is completely stopped. Though a simple description, it follows from a worst-case reasoning for object behavior (e.g. a leading vehicle coming to an immediate stop anytime). In this case, verifying FSR 1.2 can be translated into proving the $\mathcal{DL}$ formula:

$$(\text{init}) \rightarrow [(\text{sense}; sa; \text{ctrl}; \text{plant})^+] \left( x_c - x \geq \frac{v^2}{2a_{sv}^{\text{min}}} \right)$$

where $\text{ctrl}$ and $\text{plant}$ model dynamics of both the ego-vehicle and the object in the environment. Verifying (62) using KeYmaera X requires the identification of assumptions on the Sense module, which can subsequently be used to obtain requirements on the sensor range (e.g. see [29]) for the AD feature and/or refine the ODD conditions. Modeling the Situation Assessment in a modular way as described by (62) also gives the flexibility to easily reason about various algorithmic variants similar to how different threat metrics are handled in the models for the Safety Controller in Section IV. For example, to relax the worst-case reasoning for leading vehicles in (61) from an immediate stop to braking with at least $B$ from velocity $v_l$ to a stop, then $sa$ in (62) can be replaced by:

$$sa \triangleq x_c := x_l + \frac{v_l^2}{2B} + d.$$ (63)

Admittedly, the validity of the safety proofs in the deployed systems are highly dependent on the validity of the models, including the assumptions. For example, in Model 1, the $\text{plant}$ (7) models the ego-vehicle behavior such that the safe acceleration request $a_s$ is accurately tracked by the Vehicle Control. However, it is often the case that the deployed systems encounter disturbances, delays, etc., which makes it difficult to accurately track the request. Though $\mathcal{DL}$ can model such disturbances and delays, the challenge manifests in identifying the invariant conditions to manage the complexity of the proofs to be constructed by the proof system.

VII. RELATED WORK

This section presents a broad but inevitably incomplete overview of some research related to the use of formal methods to guarantee safety of automated systems. A more comprehensive survey on the formal specification and verification of autonomous robotic systems is found in [30].

Several approaches like testing, simulation, formal methods, etc. have been investigated to provide credible arguments for the safety of AD systems [4], [31]. Formal methods, unlike approaches like testing and simulation, can exhaustively verify and guarantee the absence of errors through mathematical proofs of correctness of a model of the system. Formal analysis based on finite-state methods like supervisory control theory (SCT) [32], or model checking [33] have previously been used to reason
about advanced driver assistance systems [34], AD [35], [36], [37] and other autonomous robotic systems [30]. Finite-state methods, though impressive in their own domains, limit the expressiveness of the models and require finite-state abstractions or approximations of the system. Formal analysis of traffic situations typically requires reasoning about continuous state variables like position, velocity, etc., that vary continuously with time, and obtaining finite-state abstractions of such entities risks an unconvincing safety argument. Furthermore, the use of exhaustive state-space exploration in SCT and model checking approaches is intractable for highly parametric AD systems. In comparison, the approach of this paper uses hybrid programs to model both continuous and discrete dynamics, and verifies them using mathematical proofs instead of exhaustive exploration, thereby covering infinitely many scenarios in each theorem and proof. Thus, a suitable trade-off between models closer to reality and a tractable formal analysis is achieved.

Another approach to formally verify the safety of AD is through online reachability analysis [38], [39], where the verification is performed online by predicting the reachable sets from models of the AD vehicle and other traffic participants. A notable shortcoming in this approach is the heavy computational demand in the calculation of the reachable sets. Recent progress has been made in reducing the computational demand by making conservative model abstractions [40] or by combining set-based reachability analysis with convex optimization [41]. In contrast, the verification approach used in our paper is completely offline and therefore does not contribute to the real-time computational demand in the AD vehicle. Moreover, the decision logic in the Safety Controller is by design intentionally simple in its behavior, thereby accommodating to the demands of the possibly complex Nominal Controller. Of course, the approach can be extended to include complex and more realistic models. However, a consequence is to deal with the proof complexity which might require additional manual effort to identify invariants and arithmetic simplifications to decide the validity of first-order formulas of real arithmetic.

Yet another approach to guarantee safety is to enforce set-invariance through control barrier functions as investigated in [42], [43]. An important limitation in this method lies in the construction of such control barrier functions [44]. In this regard, a similar problem with the deductive verification approach used in our paper is the identification of continuous invariants and loop invariants to improve proof automation [45], [46]. A comparison of the safety methods based on control barrier functions and reachability analysis is found in [44].

Differential dynamic logic (dL) has been used in the specification and verification of adaptive cruise control [29], [47], the European train control system [48], and aircraft collision avoidance [49]. The primary objective in those works is to demonstrate the application of dL based verification in the respective case-studies.

Our paper, in addition to showing how dL is used in the safety argument of an AD feature, discusses how such an approach can further aid in other development activities like comparison of the verified models and in the requirement refinement process. In [50], [51], KeYmaera X in combination with runtime monitoring is used to guarantee safety of reinforcement learning-based controllers. Though our work does not directly deal with reinforcement learning, as mentioned in Section II, the models and proofs presented can be used to guarantee the safety of any nominal control algorithm, including those based on reinforcement learning.

VIII. CONCLUSION

The challenges in providing convincing arguments for the safe and correct behavior of AD systems is one obstacle for their widespread deployment. Formal methods can help ensure the safety in various stages of AD development. This paper shows how differential dynamic logic (dL) and the theorem prover KeYmaera X [17] can be used in the development of an in-lane AD feature. Specifically, we:

1) illustrate how formal models and safety proofs of different design variants of a Decision & Control module is used in the safety argument of an in-lane AD feature in infinitely many operational scenarios,
2) show how the formal analysis helps to formalize the (informal) safety requirement and to identify the assumptions and invariant conditions to guarantee safety, and
3) discuss and demonstrate how formal analysis using dL and KeYmaera X can be used to verify the different models and aid in development activities like refinement requirement and evaluation of the verified models.

Furthermore, the models employ a modular architecture that uses a safety controller to supervise nominal control functions and guarantee safety. Additionally, the formal modeling approach identifies the conditions on the interactions between the Safety Controller and the other components to enforce safe behavior. Therefore, the design and verification approach in this paper can be used to guarantee safety even if the Nominal Controller implements hard-to-verify (e.g., learning-based) algorithms. Though this paper only considers an in-lane AD feature, the modular modeling approach can be extended to reason about other types of AD features. For instance, in the case of AD features involving lane changes, the env, ctrl, and plant components in the models have to be extended to describe vehicle motion in both longitudinal and lateral dimensions, similar to the models presented in [52].

A significant part of the verification effort in this paper was spent on manually identifying loop invariants. As future work, exploring different methods to automatically identify invariant candidates would be valuable to increase the industrial adoption of the approach in this paper. Another notable challenge in using formal methods for the safety argument is the presence of modeling errors like the edge case discussed in Remark 3. Such modeling errors might result in proving a faulty controller safe which could potentially be catastrophic in practice. In [53], we present methods to address two such errors and any further work in that direction will be highly beneficial. In the future we would also like to refine the models such as introducing delays and disturbances, control decisions that combine steering and braking commands, etc., and investigate the proof effort required to guarantee their safety. We believe that our work provides valuable insights for the use of formal methods in the design, development and the safety argument of AD features.
[50] N. Fulton and A. Platzer, “Safe reinforcement learning via formal methods: Toward safe control through proof and learning,” in Proc. AAAI Conf. Artif. Intell., 2018, vol. 32, pp. 6485–6492.

[51] J. Adelt, T. Liebrenz, and P. Herber, “Formal verification of intelligent hybrid systems that are modeled with simulink and the reinforcement learning toolbox,” in Proc. Int. Symp. Formal Methods, 2021, pp. 349–366.

[52] S. Mitsch, K. Ghorbal, D. Vogelbacher, and A. Platzer, “Formal verification of obstacle avoidance and navigation of ground robots,” Int. J. Robot. Res., vol. 36, no. 12, pp. 1312–1340, 2017.

[53] Y. Selvaraj, J. Krook, W. Ahrendt, and M. Fabian, “On how to not prove faulty controllers safe in differential dynamic logic,” 2022. [Online]. Available: https://arxiv.org/abs/2207.05854

Yuvaraj Selvaraj received the B.E. degree in electrical engineering from Anna University, Chennai, India, and the M.S. degree in systems, control, and mechatronics from the Chalmers University of Technology, Gothenburg, Sweden, in 2011 and 2016, respectively. Since 2017, he has been working toward the Ph.D. degree with Chalmers and Zenseact, Gothenburg, Sweden. Between 2011 and 2014, he was with ABB, India, developing control software for industrial automation. Between 2016 and 2017, he was with Volvo Car Corporation, Gothenburg, Sweden. His research focuses on the application of formal methods for automated driving.

Wolfgang Ahrendt received the Ph.D. degree in computer science from the University of Karlsruhe, Karlsruhe, Germany, in 2001. He is currently the Professor of computer science with the Chalmers University of Technology, Gothenburg, Sweden. His research interests and contributions mainly include deductive verification of software, in runtime verification, and combinations of static verification with runtime verification and testing. Recent application areas of his work include safety of blockchain applications and automotive software safety.

Martin Fabian (Senior Member, IEEE) received the Ph.D. degree in control engineering from Chalmers University of Technology, Gothenburg, Sweden, in 1995. He is currently the Full Professor of automation and also the Head of the Automation Research Group with the Department of Electrical Engineering, Chalmers University of Technology. He is co-developer of formal methods tool Supremica, which implements state-of-the-art algorithms for supervisory control synthesis. His research interests include formal methods for automation systems in a broad sense, merging the fields of control engineering, and computer science.