

On How to Not Prove Faulty Controllers Safe in Differential Dynamic Logic^{*}

Yuvaraj Selvaraj^{1,3}[0000-0003-2184-3069], Jonas Krook^{1,3}[0000-0002-9810-4697],
Wolfgang Ahrendt²[0000-0002-5671-2555], and Martin Fabian¹[0000-0003-1287-9748]

¹ Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden
{yuvaraj,krookj,fabian}@chalmers.se

² Department of Computer Science and Engineering, Chalmers University of Technology,
Göteborg, Sweden
ahrendt@chalmers.se

³ Zenseact, Göteborg, Sweden
{firstname.lastname}@zenseact.com

Abstract. Cyber-physical systems are often safety-critical and their correctness is crucial, as in the case of automated driving. Using formal mathematical methods is one way to guarantee correctness. Though these methods have shown their usefulness, care must be taken as modeling errors might result in proving a faulty controller safe, which is potentially catastrophic in practice. This paper deals with two such modeling errors in *differential dynamic logic*. Differential dynamic logic is a formal specification and verification language for *hybrid systems*, which are mathematical models of cyber-physical systems. The main contribution is to prove conditions that when fulfilled, these two modeling errors cannot cause a faulty controller to be proven safe. The problems are illustrated with a real world example of a safety controller for automated driving, and it is shown that the formulated conditions have the intended effect both for a faulty and a correct controller. It is also shown how the formulated conditions aid in finding a *loop invariant* candidate to prove properties of hybrid systems with feedback loops. The results are proven using the interactive theorem prover KeYmaera X.

Keywords: Hybrid Systems · Automated Driving · Formal Verification · Loop Invariant · Theorem Proving

1 Introduction

Cyber-physical systems (CPS) typically consist of a digital *controller* that interacts with a physical dynamic system and are often employed to solve safety-critical tasks. For example, an automated driving system (ADS) has to control an autonomous vehicle (AV) to safely stop for stop signs, avoid collisions, etc. It is thus paramount that CPS work correctly with respect to their requirements. One way to ensure correctness of CPS

^{*} This work was supported by FFI, VINNOVA under grant number 2017-05519, *Automatically Assessing Correctness of Autonomous Vehicles – Auto-CAV*, and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

is to use formal verification, which requires a formal model of the CPS. An increasingly popular family of models of CPS are *hybrid* systems, which are mathematical models that combine discrete and continuous dynamics.

To reason about the correctness of a CPS, hybrid systems can model different components of the CPS and their interactions, thus capturing the overall *closed-loop* behavior. In general, hybrid systems that model real world CPS may involve three main components: a *plant* model that describes the physical characteristics of the system, a controller model that describes the control software, and an *environment* model that captures the behaviors of the surrounding world in which the controller operates, thereby defining the *operational domain*. The goal for the controller is to choose control actions such that the requirements are fulfilled for *all* possible behaviors of the hybrid system.

Typically, the environment is modeled using nondeterminism to capture all possible behaviors. However, assumptions on the environment behavior are necessary to limit the operational domain and remove behaviors that are too hostile for any controller to act in a safe manner. For example, if obstacles are assumed to appear directly in front of an AV when driving, no controller can guarantee safety. While the assumptions in the formal models are necessary to make the verification tractable, there are subtle ways in which formal verification can provide less assurance than what is assumed [4]. In other words, as a result of the verification, the designer may conclude the controller to be safe in the entire assumed operational domain, whereas in reality some critical behaviors where the controller is actually at fault might be excluded from the verification. One possible cause for such a disparity between what is verified and what is assumed to be verified is the presence of modeling errors. In such cases, if a controller is verified to be safe, it leads to unsafe conclusions which might be catastrophic in practice. This paper deals with two such modeling errors by making them subject to interactive verification. In the first erroneous case, the environment assumptions and the controller actions interact in such a way that the environment behaves in a friendly way to adapt to the actions of a controller that exploits its friendliness. Then, it is possible that a faulty controller can be proven safe since the environment reacts to accommodate the bad control actions. An example of this is a faulty ADS controller that never brakes, together with an environment that reacts by always moving obstacles to allow the controller not to brake.

In the second erroneous case, the assumptions about the environment and/or other CPS components remove all behaviors in which any action by the controller is needed. In this case, the assumptions over-constrain the allowed behaviors. For example, if the assumptions restrict the behavior of the AV to an extent that only braking is possible, then a faulty ADS controller can be proven safe because nothing is proven about the properties of the controller. In the worst-case, the assumptions remove all *possible* behaviors, thereby making the requirement vacuously true.

In both cases, a faulty controller can be proven safe with respect to the requirements for the wrong reasons, i.e., unintended modeling errors, thus resulting in potentially catastrophic operation of the CPS in practice. Modeling errors are in general hard to address because every model is an abstraction and there exists no ubiquitous notion of what a *correct model* means. Therefore, a systematic way to identify and avoid modeling errors is highly desirable as it reduces the risk of unsound conclusions when a

model is formally proven safe with respect to the requirements. Typically, the requirements specify (un)desired behavior of the closed-loop system within the operational domain and are expressed in some logical formalism to apply formal verification. *Differential dynamic logic* (dL) [8, 9] is a specification and verification language that can be used to formally describe and verify hybrid systems. The interactive theorem prover KeYmaera X [2] implements a sound proof calculus [8, 9] for dL and can thus mathematically prove that the models fulfill their specified requirements.

The main contributions of this paper, Theorem 1 and Theorem 2, formulate and prove conditions that when fulfilled, ensure the model cannot be proven safe if it is susceptible to the above modeling errors. Essentially, a loop invariant is used not only to reason about the model inductively but also to ensure that the interaction between the controller and the other components in the model is as intended; the two theorems provide conditions on the relation between the assumptions and the loop invariant. Furthermore, these conditions give hints as to when a suggested loop invariant for the model is sufficiently strong to avoid modeling errors. The problems are illustrated with a running example of an automated driving controller that shows that they can appear in real models. It is then proven that the formulated conditions have the intended effect. Finally, it is shown by example that the method captures the problematic cases and also increases confidence in a correct model free from the considered modeling errors.

2 Preliminaries

The logic dL uses *hybrid programs* (HP) to model hybrid systems. An HP α is defined by the following grammar, where α, β are HPs, x is a variable, e is a term⁴, and P and Q are formulas of first-order logic of real arithmetic (FOL)⁵:

$$\alpha ::= x := e \mid x := * \mid ?P \mid x' = f(x) \& Q \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$$

Each HP α is semantically interpreted as a reachability relation $\llbracket \alpha \rrbracket \subseteq \mathcal{S} \times \mathcal{S}$, where \mathcal{S} is the set of all states. If \mathcal{V} is the set of all variables, a state $\omega \in \mathcal{S}$ is defined as a mapping from \mathcal{V} to \mathbb{R} , i.e., $\omega: \mathcal{V} \rightarrow \mathbb{R}$. The notation $(\omega, \nu) \in \llbracket \alpha \rrbracket$ denotes that final state ν is reachable from initial state ω by executing the HP α . $\omega \llbracket e \rrbracket$ denotes the value of term e in state ω , and for $x \in \mathcal{V}$, $\omega(x) \in \mathbb{R}$ denotes the real value that variable x holds in state ω . Given a state ω_1 , a state ω_2 can be obtained by assigning the terms $\{e_1, \dots, e_n\}$ to the variables $y = \{y_1, \dots, y_n\} \subseteq \mathcal{V}$ and letting the remaining variables in \mathcal{V} be as in ω_1 , that is, $\omega_2(y_i) = \omega_1 \llbracket e_i \rrbracket$ for $1 \leq n$ and $\omega_2(v) = \omega_1(v)$ for all $v \in \mathcal{V} \setminus y$. Let $\omega_2 = \omega_1(y_1 := e_1, \dots, y_n := e_n)$ be a shorthand for this assignment. For a FOL formula P , let $\llbracket P \rrbracket \subseteq \mathcal{S}$ be the set of all states where P is true, thus $\omega \in \llbracket P \rrbracket$ denotes that P is true in state ω . If P is parameterized by y_1, \dots, y_n , then $\omega \in \llbracket P \rrbracket$ means that $\omega \in \llbracket P(\omega(y_1), \dots, \omega(y_n)) \rrbracket$. A summary of the program statements in HP and their transition semantics [9] is given in Table 1.

⁴ Terms are polynomials with rational coefficients defined by $e, \tilde{e} ::= x \mid c \in \mathbb{Q} \mid e + \tilde{e} \mid e \cdot \tilde{e}$.

⁵ First-order logic formulas of real arithmetic are defined by $P, Q ::= e \geq \tilde{e} \mid e = \tilde{e} \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \rightarrow Q \mid P \leftrightarrow Q \mid \forall xP \mid \exists xP$

Table 1: Semantics of HPs [9]. P, Q are first-order formulas, α, β are HPs.

Statement	Semantics
$\llbracket x := e \rrbracket$	$= \{ (\omega, \nu) : \nu = \omega(x := e) \}$
$\llbracket x := * \rrbracket$	$= \{ (\omega, \nu) : c \in \mathbb{R} \text{ and } \nu = \omega(x := c) \}$
$\llbracket ?P \rrbracket$	$= \{ (\omega, \omega) : \omega \in \langle P \rangle \}$
$\llbracket x' = f(x) \& Q \rrbracket$	$= \{ (\omega, \nu) : \phi(0) = \omega(x' := f(x)) \text{ and } \phi(r) = \nu \text{ for a solution } \phi : [0, r] \rightarrow \mathcal{S} \text{ of any duration } r \text{ satisfying } \phi \models x' = f(x) \wedge Q \}$
$\llbracket \alpha \cup \beta \rrbracket$	$= \llbracket \alpha \rrbracket \cup \llbracket \beta \rrbracket$
$\llbracket \alpha; \beta \rrbracket$	$= \llbracket \alpha \rrbracket \circ \llbracket \beta \rrbracket = \{ (\omega, \nu) : (\omega, \mu) \in \llbracket \alpha \rrbracket, (\mu, \nu) \in \llbracket \beta \rrbracket \}$
$\llbracket \alpha^* \rrbracket$	$= \llbracket \alpha \rrbracket^* = \bigcup_{n \in \mathbb{N}_0} \llbracket \alpha^n \rrbracket \text{ with } \alpha^0 \equiv ?\text{true} \text{ and } \alpha^{n+1} \equiv \alpha^n; \alpha.$

The sequential composition $\alpha; \beta$ expresses that β starts executing after α has finished. The *nondeterministic choice* operation expresses that the HP $\alpha \cup \beta$ can nondeterministically choose to follow either α or β . The *test* action $?P$ has no effect in a state where P is true, i.e., the final state ω is same as initial state ω . If however P is false when $?P$ is executed, then the current execution of the HP *aborts*, meaning that no transition is possible and the entire current execution is removed from the set of possible behaviors of the HP. The *nondeterministic repetition* α^* expresses that α repeats n times for any $n \in \mathbb{N}_0$. Furthermore, test actions can be combined with sequential composition and the choice operation to define *if-statements* as:

$$\text{if}(P) \text{ then } \alpha \text{ fi} \equiv (?P; \alpha) \cup (? \neg P) \quad (1)$$

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 applies bounds on the continuous dynamics and are first-order formulas that restrict the continuous evolution within that bound. 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 formulas of dL include formulas of first-order logic of real arithmetic and the modal operators $[\alpha]$ and $\langle \alpha \rangle$ for any HP α [8, 9]. A formula θ of dL is defined by the following grammar (ϕ, ψ are dL formulas, e, \tilde{e} are terms, x is a variable, α is an HP):

$$\theta ::= e = \tilde{e} \mid e \geq \tilde{e} \mid \neg \phi \mid \phi \wedge \psi \mid \forall x \phi \mid [\alpha] \phi \quad (2)$$

The dL formula $[\alpha] \phi$ expresses that *all* non-aborting executions of HP α (i.e., the executions where all test actions are successful) end in a state in which the dL formula ϕ is true. The formal semantics are defined by $\langle \llbracket [\alpha] \phi \rrbracket \rangle = \{ (\omega \in \mathcal{S} : \forall \nu \in \mathcal{S}. (\omega, \nu) \in \llbracket \alpha \rrbracket \rightarrow \nu \in \langle \phi \rangle) \}$. The dL formula $\langle \alpha \rangle \phi$ means that there exists *some* non-aborting execution leading to a state where ϕ is true. $\langle \alpha \rangle \phi$ is the dual to $[\alpha] \phi$, defined as $\langle \alpha \rangle \phi \equiv \neg [\alpha] \neg \phi$. Similarly, $>, \leq, <, \vee, \rightarrow, \leftrightarrow, \exists x$ are defined using combinations of the operators in (2). A dL formula θ is *valid*, denoted $\models \theta$ if $\langle \theta \rangle = \mathcal{S}$.

The logic dL and the interactive theorem prover KeYmaera X support the specification and verification of hybrid systems. The dL formula $(\text{init}) \rightarrow [\alpha] (\text{guarantee})$ can

Model 1: The general model considered

$$(init) \rightarrow [(env; aux; ctrl; plant)^*](guarantee) \quad (3)$$

$$env \triangleq e := *; ?P(s, e, a) \quad (4)$$

$$aux \triangleq a := *; ?Q(s, e, a) \quad (5)$$

$$ctrl \triangleq \text{if } \neg ok(s, e, a) \text{ then } a := *; ?C(s, e, a) \text{ fi} \quad (6)$$

$$plant \triangleq \tau := 0; s' = f(s), \tau' = 1 \ \& \ F(s, e, a, \tau) \wedge \tau \leq T \quad (7)$$

be used to specify the correctness of an HP α with respect to the requirement *guarantee*. It expresses that, if the initial conditions described by the formula *init* are true, then all (non-aborting) executions of α only lead to states where formula *guarantee* is true. KeYmaera X takes such a dL formula as input and successively decomposes it into several sub-goals according to the sound proof rules of dL to prove the formula [8, 9].

Often, modeling CPS as HPs involves execution of a controller together with a plant in a loop described by the nondeterministic repetition α^* . To prove properties of loops, like the property $(init) \rightarrow [\alpha^*](guarantee)$, KeYmaera X uses *loop invariants*, provided by the user, to inductively reason about all (non-aborting) executions. Given a loop invariant (candidate) ζ , applying the loop invariant rule to the above formula would make the proof branch into the following three cases:

loop (i) : $(init) \rightarrow \zeta$, i.e., the initial state satisfies the invariant,

loop (ii) : $\zeta \rightarrow [\alpha] \zeta$, i.e., invariant remains true after one iteration of α from any state where the invariant was true,

loop (iii) : $\zeta \rightarrow (guarantee)$, i.e., the invariant implies the requirement.

3 Problem Scope

The scope of CPS considered in this paper are hybrid systems with closed-loop feedback control as described by Model 1. The dL formula (3) models the CPS as a HP that repeatedly executes in a loop and expresses the requirement on the CPS by the formula *guarantee*. The HP in (3) is composed of four different components, each of which is an HP and assigns four variables: the dynamic state s which evolves continuously, the control actions a , the environment actions e , and the time progress τ . Though the variables in Model 1 are scalars, they can in general be vectors of any dimension.

The environment (*env*) in (4) describes the environment behavior using a nondeterministic assignment followed by a test. The environment action e is nondeterministically assigned a real value which is then checked by the subsequent test for adherence to the environment assumptions P , which define the operational domain. The auxiliary system (*aux*) describes the internal digital system that the controller interacts with, in addition to the environment. Similarly to *env*, *aux* (5) nondeterministically assigns a real value to the control action a followed by a subsequent test which checks whether the internal assumptions Q hold. These internal assumptions typically describe conditions that stem from the design of the CPS such as physical limits on the system actuators.

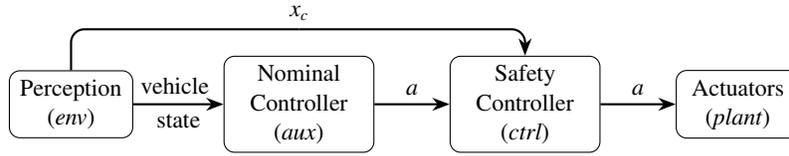


Fig. 1: Architecture of the automated driving feature.

The controller’s (*ctrl*) task is to ensure that the requirement *guarantee* is fulfilled and is modeled as an if-statement as seen in (6). First, the control action *a* set by *aux* is tested with *ok*. If the test is not *ok*, then *ctrl* overrides the control action *a* by the control law *C*, and finally it passes on the control action to the *plant* (7), which models the physical part of the system. It is described as an ordinary differential equation. However, the sampling time of *ctrl* is bounded, so the evolution of *plant* must stop before the sampling time *T* [11].

In the most abstract setting, the parameterized FOL formulas in Model 1 are treated as uninterpreted predicates, which could be replaced by any concrete hybrid model with specific formulas and HPs, as long as the assignment of values to variables follows the flow of Model 1. Hence, the conclusions drawn from Model 1 can be applied and used for a wide variety of hybrid systems.

Running Example: Automated Driving Controller

To illustrate the problems and solutions, this paper considers an example of an in-lane automated driving feature for an AV, the *ego-vehicle*. Fig. 1 shows a simplified architecture of the automated driving feature, which can be modeled as a HP of the general form in Model 1. The safety requirement is for the ego-vehicle to safely stop for stationary obstacles that have entered its path.

The *perception* senses the world around the ego-vehicle and corresponds to the *env* in Model 1. The *env* models the perception algorithms that communicate the obstacle position x_c to the controller and thus the *env* assumptions describe the dynamics of the obstacles appearing in the ego-vehicle’s path. The *nominal controller*, described by *aux*, represents any algorithm solving the nominal driving task subjected to different constraints (e.g. comfort) and requests a nominal acceleration. Thus, *aux* of the form in (5) allows to keep the model parametric to arbitrary nominal controller implementations while being regarded as a black box. The *aux* assumptions therefore capture design conditions on the nominal controller such as always requesting an acceleration within certain bounds.

The *safety controller* described by *ctrl* ensures that only safe control actions, i.e., acceleration commands *a*, are communicated to the actuators. It evaluates the nominal acceleration and overrides it with a safe acceleration if needed, thereby satisfying the safety requirement. Thus, the verification of the safety requirement can be limited to verifying the decision logic in one component, the safety controller.

The *plant* is a dynamic model of the ego-vehicle. It is modeled as a double integrator with position x and velocity v of the ego-vehicle as the dynamic states and the

Model 2: Example hybrid system

$$\begin{aligned} \mathit{init} \triangleq & v = 0 \wedge x \leq x_c \wedge a_s^{\min} > 0 \wedge a_n^{\max} > 0 \\ & \wedge a_n^{\min} > 0 \wedge a_s^{\min} > a_n^{\min} \wedge T > 0 \end{aligned} \quad (8)$$

$$\mathit{guarantee} \triangleq (x \leq x_c) \quad (9)$$

$$\mathit{plant} \triangleq \tau := 0; x' = v, v' = a, \tau' = 1 \ \& \ v \geq 0 \wedge \tau \leq T \quad (10)$$

acceleration a as control input, as seen in (10) of Model 2. The ego-vehicle is not allowed to drive backwards, so v must be non-negative through the entire evolution. In other words, the evolution would stop before v gets negative.

In the next section, the general dL formula in (3) is refined with concrete descriptions of env , aux , and ctrl to illustrate the modeling errors where a faulty controller can be proven safe. However, init , plant , and $\mathit{guarantee}$ remain unchanged in the subsequent models and are shown in Model 2. The initial condition init (8) specifies that the ego-vehicle starts stationary ($v = 0$) at an arbitrary position x before the position x_c of an obstacle. It also sets up assumptions on the *constant* parameters such as the minimum safety and nominal acceleration a_s^{\min} and a_n^{\min} , and maximum nominal acceleration, a_n^{\max} , and that the sampling time T is positive. These constant parameters do not change value during the execution of the HP $[(\mathit{env}; \mathit{aux}; \mathit{ctrl}; \mathit{plant})^*]$, and therefore the assumptions on the constant parameters remain true in all contexts. The requirement that the ego-vehicle must stop before stationary obstacles is expressed by the post condition $\mathit{guarantee}$ (9), which says that the obstacle's position may not be exceeded.

4 Discover Modeling Errors

This section presents two erroneous models to illustrate how a faulty ctrl can be proven safe with respect to $\mathit{guarantee}$. In the first case, shown in Model 3, improper interaction between env and ctrl results in env adapting to faulty ctrl actions. Such an erroneous model can be proven safe since the loop invariant ζ is not strong enough to prevent improper interactions. Theorem 1 gives conditions to strengthen ζ to avoid such issues. In the second erroneous case, Model 4, the error arises due to over-constrained env and aux assumptions that discard executions where ctrl is at fault. Theorem 2 presents conditions to identify and avoid errors due to such over-constrained assumptions.

4.1 Exploiting Controller

Consider Model 3 where the assumptions on env and aux are given by (11) and (12) respectively. env assigns x_c such that it is possible to brake and stop before the position of the obstacle. This is necessary since if an obstacle appears immediately in front of the moving ego-vehicle it is physically impossible for any controller to safely stop the vehicle. aux is a black box, but it is known that the nominal acceleration request a is bounded. The ctrl test ok (14) checks whether maximal acceleration for a time period of T leads to a violation of the requirement, and if it does, the controller action C (15)

Model 3: *ctrl* is exploiting

$$env \triangleq x_c := *; ? \left(x_c - x \geq \frac{v^2}{2a_n^{min}} \right) \quad (11)$$

$$aux \triangleq a := *; ? (-a_n^{min} \leq a \leq a_n^{max}) \quad (12)$$

$$ctrl \triangleq \text{if } \neg ok(x, v, x_c, a) \text{ then } a := *; ? C(x, v, x_c, a) \text{ fi} \quad (13)$$

$$ok(x, v, x_c, a) \triangleq \left(x_c - x \geq vT + \frac{a_n^{max} T^2}{2} \right) \quad (14)$$

$$C(x, v, x_c, a) \triangleq a = -a_s^{min} \quad (15)$$

sets the deceleration to its maximum. This maximum deceleration is a symbolic value, parameterized over the other model variables.

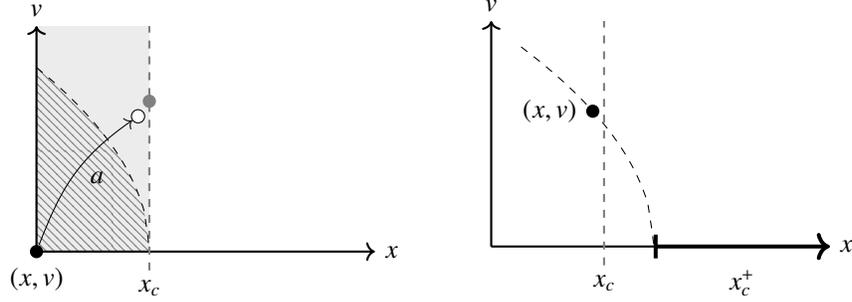
Denote by θ the dL formula (3) together with the definitions of Model 2 and Model 3. θ is proved [12] with the loop invariant $\zeta_1 \equiv x \leq x_c$. Though the goal is to find a proof that θ is valid, and thereby establish that *ctrl* is safe with respect to *guarantee*, it is in this case incorrect to draw that conclusion from the proof, as will now be shown.

The *env* assumption (11) discards executions where the distance between the obstacle position x_c and the ego-vehicle position x is less than the minimum possible braking distance of the ego-vehicle. This assumption is reasonable as it only discards situations where it is physically impossible for *ctrl* to safely stop the vehicle. Still, infinitely many *env* behaviors are possible since x_c is nondeterministically assigned any value that fulfills the assumption. Among other behaviors, this allows x_c to remain constant, as would be the case for stationary obstacles. However, due to improper interaction between *env* and a faulty *ctrl*, *env* can be forced by *ctrl* to not have x_c constant.

Consider a state $\omega_0 \in \llbracket \zeta_1 \rrbracket$, illustrated in Fig. 2a, such that

$$\begin{array}{llll} \omega_0(x) = 0 & \omega_0(x_c) = 1 & \omega_0(T) = 1 & \\ \omega_0(v) = 0 & \omega_0(a) = 1.8 & \omega_0(a_n^{max}) = 2 & \omega_0(a_n^{min}) = 3. \end{array}$$

The ego-vehicle is currently at $(x, v) = (0, 0)$ as shown by the black circle. The hatched area represents all the points in the xv -plane from which it is possible to stop before the obstacle position, x_c , at the dashed vertical line. It holds that $(\omega_0, \omega_0) \in \llbracket env \rrbracket$ since $x_c - x = 1 \geq 0^2 / (2 \times 3) = v^2 / (2a_n^{min})$, so the assumptions on *env* allow $x_c = 1$. This can also be seen in the figure since the black circle is within the hatched area. The arrow labeled a in Fig. 2a represents the acceleration request by *aux*, and if *plant* evolves for 1 second with a as input, the ego-vehicle ends up at the white circle. As a is within the bounds of *aux*, it holds that $(\omega_0, \omega_0) \in \llbracket aux \rrbracket$. The controller *ctrl* is *ok* with this choice since x_c is not passed if maximum acceleration a_n^{max} is input to *plant*, as illustrated by the gray circle in the figure. Formally, $x_c - x = 1 \geq 0 \times 1 + 2 \times 1^2 / 2 = vT + a_n^{max} T^2 / 2$ and therefore it holds by (14) that $\omega_0 \in \llbracket ok(x, v, a_n, a) \rrbracket$. Thus, $(\omega_0, \omega_0) \in \llbracket ctrl \rrbracket$. Let $\omega_1 = \omega_0(x := 0.9, v := 1.8)$. Now it holds that $(\omega_0, \omega_1) \in \llbracket plant \rrbracket$, i.e., starting at $x = 0$ and $v = 0$, with $a = 1.8$ as input, *plant* evolves to $x = 0.9$ and $v = 1.8$ in 1 second.



(a) Graphical representation of the state ω_0 . The hatched area contains all points in the xv -plane from which it is possible to stop before the obstacle x_c . The invariant ζ_1 evaluates to true in the shaded area.

(b) Graphical representation of the state ω_1 . A friendly *env* discards all obstacle positions in the interval between x_c and the start of the thick black line, and places the obstacle along the interval indicated by x_c^+ .

Fig. 2: The controller chooses an action such that the *plant* evolves to a state where $x \leq x_c$. In the next loop iteration, *env* moves x_c to adapt to the controller's action.

After *plant* has evolved and the system has transited to ω_1 , the ego-vehicle is now at the black circle in Fig. 2b. It is clear that $\omega_1 \in \langle \zeta_1 \rangle$ as $x \leq x_c$. The intersection of the dashed curve with the x -axis in Fig. 2b represents the lower bound for x_c to satisfy (11) in the state ω_1 . Therefore, in the next iteration, x_c can only be positioned somewhere along the interval indicated by the thick black line in Fig. 2b and all other values are discarded by (11). Semantically, as $x_c - x = 0.1 < 2^2 / (2 \times 3) = v^2 / (2a_n^{\min})$, it follows that $(\omega_1, \omega_1) \notin \llbracket env \rrbracket$ so x_c cannot be kept constant between iterations.

To summarize, it holds that $\omega_0 \in \langle \zeta_1 \rangle$, $(\omega_0, \omega_1) \in \llbracket env; aux; ctrl; plant \rrbracket$, and $\omega_1 \in \langle \zeta_1 \rangle$. The acceleration requested by *aux* is *ok'd* by *ctrl* in ω_0 because the worst-case acceleration a_n^{\max} in ω_0 leads to a state that fulfills ζ_1 , and therefore also fulfills *guarantee*. Since there exists no control action allowed by the system dynamics in the assumed operational domain that can fulfill *guarantee* from ω_1 , the decision made by *ctrl* is unsafe in this case. However, since $(\omega_1, \omega_1) \notin \llbracket env \rrbracket$, Model 3 can be proven to fulfill *guarantee* with this faulty *ctrl*.

So, the model is proven to fulfill *guarantee* only because *env* is not allowed to keep the obstacle stationary. Thus, *ctrl* exploits the behavior of *env* to move the obstacle so *ctrl* can keep accelerating rather than stopping safely. Though *env* is assumed to discard only those behaviors where it is physically impossible for *ctrl* to fulfill *guarantee*, the interaction between *env* and *ctrl* causes *env* to behave in a friendly way to adapt to faulty *ctrl* actions, thereby discarding *env* behaviors in which x_c remains constant.

Problem 1. How can the dL formula (3) be guaranteed not to be valid with a controller that exploits the environment?

Observe from Fig. 2a that for the state ω_0 , the shaded area describes the region where the loop invariant ζ_1 holds. The hatched area describes the states from where it is possible for *ctrl* to stop before the obstacle x_c , i.e., all the xv -points for which the *env*

assumption $x_c - x \geq \frac{v^2}{2a_n^{min}}$ in (11) is true. The shaded area contains some states in the xv -plane that are outside of the hatched area. From these states it is not possible for *ctrl* to stop before x_c . Thus, control actions leading to such states should not be allowed. However, ζ_1 is not strong enough to prevent this. If ζ_1 is strengthened to allow only states contained in the hatched area then the controller is prevented from exploiting the environment. In other words, any state allowed by the loop invariant shall also be allowed by the *env* assumptions, i.e., the loop invariant should imply the *env* assumptions.

The assumption $x_c - x \geq \frac{v^2}{2a_n^{min}}$ in (11) corresponds to P in the generalized Model 1. Therefore, it can be hypothesized from the above observation that the required condition to solve Problem 1 can be stated as $\zeta \rightarrow P$, where ζ is the loop invariant and P is the *env* assumptions. Indeed, the condition $\zeta \rightarrow P$ solves Problem 1 for Model 3. However, Problem 1 is not specific to Model 3 and it remains unestablished whether $\zeta \rightarrow P$ solves Problem 1 for models of the general form considered in Model 1. For example, in Model 3, the controller exploits the friendliness of *env* to not keep the obstacle position x_c constant between iterations, i.e., $x_c \neq x_c^+$ for two *env* actions (x_c, x_c^+) . Admittedly, such a behavior does not characterize friendly behavior in all models. In general, the relation between two *env* actions (e_0, e_1) can be any relation $R \subseteq \mathbb{R} \times \mathbb{R}$ such that $(e_0, e_1) \in R$. Note that R only defines certain behaviors in the assumed operational domain. In Model 3, the exploiting controller could be proven safe because the environment behaves *friendly* by discarding some behaviors characterized by R . This is illustrated in Fig. 2b where x_c cannot be kept constant as $(\omega_1, \omega_1) \notin \llbracket env \rrbracket$.

Definition 1. *If there exists two states ω_0 and ω_1 that differ only in the assignment of the env variable e , i.e., $\omega_0(e) = e_0$ and $\omega_1 = \omega_0(e := e_1)$, and such that $(e_0, e_1) \in R$ and $(\omega_0, \omega_1) \notin \llbracket env \rrbracket$, then the environment *env* is friendly w.r.t the relation R . Thus, *env* is unfriendly if $(e_0, e_1) \in R \rightarrow (\omega_0, \omega_1) \in \llbracket env \rrbracket$ is true in all states ω_0 and ω_1 that differ only in the assignment of the env variable e .*

The hypothesis $\zeta \rightarrow P$ can now be generalized to include the relation R to describe the existence of an unfriendly *env* as:

$$\rho \equiv \forall s. \forall e. \forall e_1. \left(\zeta(s, e) \wedge R(e, e_1) \rightarrow \langle env \rangle (e = e_1) \right), \quad (16)$$

where ζ is parameterized to make it explicitly depend on the variables of the HP. The meaning of ρ is that, if a state fulfills the invariant, then for every next *env* action e_1 characterized by R there is at least one execution of *env* in which the action e_1 is chosen.

The loop invariant $\zeta_1 \equiv x \leq x_c$ is used to prove the dL formula (3) with the definitions of Model 2 and Model 3. Thus, it follows that $\models \zeta_1 \rightarrow [env; aux; ctrl; plant] \zeta_1$ holds by **loop** (ii). But, ζ_1 is not strong enough to prevent control actions that exploit friendly *env* behaviors. For instance, as illustrated in Fig. 2, the control action that leads to ω_1 from ω_0 should not be allowed since *env* must discard some behaviors from ω_1 to preserve ζ_1 . These discarded behaviors include all executions where $(aux; ctrl; plant)$ do not preserve ζ_1 . Thus *ctrl exploits env* to act friendly such that ζ_1 is preserved.

Definition 2. *A controller *ctrl* exploits a friendly environment *env* w.r.t the relation R if the loop invariant ζ is preserved by the loop body, i.e. $\models \gamma$, where*

$$\gamma \equiv \forall s. \forall e. \left(\zeta(s, e) \rightarrow [env; aux; ctrl; plant] \zeta(s, e) \right), \quad (17)$$

but

$$\exists s. \exists e. \exists e_0. \left(\zeta(s, e_0) \wedge R(e_0, e) \wedge \langle aux; ctrl; plant \rangle \neg \zeta(s, e) \right). \quad (18)$$

Thus, *ctrl* exploits *env* if it makes it necessary for *env* to behave friendly. In the following theorem it is shown that an exploiting controller can be prevented if the loop invariant is strong enough to ensure the existence of an unfriendly environment.

Theorem 1. *Let s and e be variables used in *plant* and *env* respectively as defined in Model 1. Let $\zeta(s, e)$ be a loop invariant candidate, and let R be a relation over the domain of e . Let γ (17) be the dL formula from the inductive step `Loop` (ii) of the loop invariant proof rule, and let ρ be as defined by (16). If $\gamma \wedge \rho$ is valid, then the loop invariant candidate $\zeta(s, e)$ is sufficiently strong to prevent an exploiting controller.*

Proof. The following dL formula is proved [12] in KeYmaera X:

$$\gamma \wedge \rho \rightarrow \forall s. \forall e_0. \forall e. \left(\zeta(s, e_0) \wedge R(e_0, e) \rightarrow [aux; ctrl; plant] \zeta(s, e) \right). \quad (19)$$

This asserts that the loop invariant is strong enough to prevent *ctrl* from exploiting *env*'s friendly behavior because the clause implied by $\gamma \wedge \rho$ in (19) is the negation of (18). \square

In addition to solving Problem 1, Theorem 1 gives hints on how the loop invariant must be constructed. In some cases, as in Fig. 2 where $x_c \leq x_c^+$, it suggests that $\zeta \equiv P$ might be a loop invariant candidate. In summary, Theorem 1 is useful in two ways: (i) By adding ρ to a dL formula, it is known that a proof of validity is not because *env* is friendly to *ctrl*, (ii) ρ can also be a useful tool to aid in the search for a loop invariant.

For the specific model instance considered in this section, and probably others, changes to the model can ensure that the environment is not too friendly. However, as this paper deals with modeling errors and ascertaining that models cannot be proven valid for wrong reasons, such changes do not solve the general problem, but might nonetheless be good as best practices to avoid modeling pitfalls.

4.2 Unchallenged Controller

The previous section dealt with modeling problems where *ctrl* causes *env* to exhibit friendly behaviors despite correct *env* assumptions. This section discusses modeling problems due to over-constrained assumptions, whereby *ctrl* is never challenged.

Consider Model 4, identical to Model 3, except for *aux* ((20) and (21)). As before, *aux* is a black box. However, in addition to the acceleration bounds, *aux* also fulfills a design requirement *req* given by (21). *req* describes that the nominal controller only requests an acceleration a such that the ego-vehicle does not travel more than the braking distance (with a_n^{min}) from any given state in one execution of T duration. Similar to Model 3, the requested acceleration is passed to the *plant* if the *ctrl* test *ok* (14) is true; if not, the controller action C (15) sets the maximal possible deceleration.

To verify that *ctrl* fulfills *guarantee* (9), the dL formula (3) together with the definitions in Model 2 and Model 4 must be proven valid. Though the validity can indeed be proven in KeYmaera X using the loop invariant $\zeta_1 \equiv x \leq x_c$, *ctrl* is faulty. Strong *env* and *aux* assumptions might result in the invariant ζ being true in all HP executions

Model 4: *ctrl* is unchallenged

$$aux \triangleq a := *; ?(-a_n^{min} \leq a \leq a_n^{max} \wedge req) \quad (20)$$

$$req \triangleq (v + aT \geq 0) \rightarrow vT + \frac{aT^2}{2} \leq \frac{v^2}{2a_n^{min}} \quad (21)$$

$$\wedge (v + aT < 0) \rightarrow a \leq -a_n^{min}$$

irrespective of *ctrl*'s actions, and hence *ctrl* is never verified. This manifests itself in Model 4; *env* assigns x_c such that it is possible to brake to stop before the position of the obstacle, and *aux* assumes that the ego-vehicle does not travel more than the braking distance in T time. Therefore, *guarantee* is true for all executions of $[env; aux; plant]$, i.e., the model fulfills *guarantee* no matter which branch of *ctrl* is executed. Thus, this problem with strong *env* and *aux* assumptions, i.e., an over-constrained model such that *ctrl* is not challenged in any HP execution, may allow a faulty controller be proven safe.

Problem 2. How can the dL formula (3) be guaranteed not to be valid with an unchallenged controller?

In general, if *aux* and/or *env* assumptions are too strong, many relevant executions may be discarded when the respective tests fail. A worst-case situation is when a contradiction is present in the assumption, thereby discarding all possible executions of the HP. In that case, the dL formula (3) is vacuously true, irrespective of the correctness of *ctrl*. In situations where all possible executions are discarded due to failed tests, a potential work-around is to check for such issues by proving the validity of $init \rightarrow \langle env; aux; ctrl; plant \rangle (guarantee)$ to verify that there exists at least one execution of the hybrid program that fulfills *guarantee*. However, that work-around is not helpful to discover models susceptible to Problem 2 because it is possible to prove that there is at least one execution of $(env; aux; ctrl; plant)$ for which *guarantee* is true even in over-constrained systems as seen in the HP with definitions of Model 2 and Model 4.

Observe that if *ctrl* is removed from the dL formula (3) and the formula is still valid, then *ctrl* is not verified. Equivalently, if the invariant is preserved when *ctrl* is removed from the dL formula, i.e., $\chi \equiv \forall s. \forall e. \forall a. \zeta \rightarrow [env; aux; plant] \zeta$ is valid, then *ctrl* is not verified. So the negation, i.e.,

$$\neg\chi \equiv \exists s. \exists e. \exists a. \zeta \wedge \langle env; aux; plant \rangle \neg\zeta. \quad (22)$$

can be proved to ascertain the absence of Problem 2 in the proof of (3).

Definition 3. For hybrid systems described by Model 1 where the loop body is defined by $(env; aux; ctrl; plant)$, *ctrl* is challenged w.r.t. *env*, *aux*, *plant*, and the loop invariant ζ if $\zeta \wedge \langle env; aux; plant \rangle \neg\zeta$ is true in some state.

However, proving $\neg\chi$ (22) might not be beneficial in practice. While failed attempts to prove $\neg\chi$ might illuminate modeling errors, the presence of *env*, *aux*, *plant*, and their interaction might complicate both the proof attempts and the identification of problematic fragments of the HP, especially for large and complicated models.

Note that if there exists one execution of $(env; aux)$ that does not preserve the invariant ζ , then $ctrl$ must choose a safe control action such that the hybrid system can be controlled to remain within the invariant states, i.e. $\langle \zeta \rangle$. However, this is not sufficient to conclude that the controller is verified to be safe since it could be the case that for all such invariant violating executions, the $plant$ forces the hybrid system back into the invariant states. Therefore, it is necessary that not all executions of the uncontrolled $plant$ reestablish the invariant. So, if $(env; aux)$ does not preserve the invariant, $plant$ does not reestablish the invariant, then $ctrl$ is indeed verified to be safe as shown in Theorem 2.

Theorem 2. *Let s , e , and a be variables used in $plant$, env , and $ctrl$ respectively as defined in Model 1, and let the loop invariant candidate $\zeta(s, e, a_1)$ be a specific instantiation of the dL formula $\zeta(s, e, a)$. Let*

$$\psi \equiv \exists s. \exists e. \exists a_1. \left(\zeta(s, e, a_1) \wedge \langle env; aux \rangle \left(\neg \zeta(s, e, a) \wedge \langle plant \rangle \neg \zeta(s, e, a_1) \right) \right). \quad (23)$$

Then, if ψ is valid, $ctrl$ is challenged in some executions of $[env; aux; ctrl; plant]$.

Proof. The following dL formula is proved [12] in KeYmaera X:

$$\psi \rightarrow \exists s. \exists e. \exists a_1. \zeta(s, e, a_1) \wedge \langle env; aux; plant \rangle \neg \zeta(s, e, a_1). \quad (24)$$

The dL formula ψ (23) states that there exists at least one execution of $(env; aux)$ where the invariant is not preserved, and $plant$ does not always reestablish the invariant. The implied clause (24) asserts that $ctrl$ is challenged by Definition 3. \square

By the conjunction of ψ (23) to a dL formula of the form (3), Theorem 2 can be used to identify Problem 2 and also the problematic fragments in all models of the form of Model 1. Furthermore, in HPs of the form $(env; ctrl; plant)^*$, with no distinction between env and aux , Theorem 2 can still be used to determine whether the env assumption is over-constrained. In addition, ψ provides insights to aid in the search of a loop invariant and its dependency on the HP variables.

5 Results

This section shows how Theorem 1 and Theorem 2 are used to (i) identify that Model 3 and Model 4 are deceptive for the verification of $ctrl$, (ii) aid in the identification of a candidate loop invariant, and (iii) increase confidence in the fidelity of Model 5 where the errors are corrected. The HPs and the KeYmaera X proofs are available from [12].

The dL formula (3) with the definitions in Model 2 and Model 3, denoted as θ , is proved in KeYmaera X with the loop invariant $\zeta_1 \equiv x \leq x_c$. Therefore it follows from loop (ii) that $\models \gamma$, where $\gamma \equiv \zeta_1 \rightarrow [env; aux; ctrl; plant] \zeta_1$. By Theorem 1, ρ must hold for Model 3 to conclude the absence of Problem 1. The formula

$$\neg \rho_1 \equiv \exists x. \exists v. \exists x_c. \exists x_c^+. \neg \left(x \leq x_c \wedge x_c \leq x_c^+ \rightarrow \langle x_c := *; ?(v^2 \leq 2a_n^{min}(x_c - x)) \rangle (x_c = x_c^+) \right), \quad (25)$$

Table 2: Summary of validity results for incorrect and correct models

Model Loop	Conjuncts	Valid Reason	invariant
3	ζ_1	-	Yes Exploiting controller
3	ζ_1	ρ_1	No Invariant not strong enough
3	ζ_2	ρ_2	No Controller does not fulfill requirement
4	ζ_1	-	Yes Unchallenged controller
4	ζ_1	$\neg\chi_1$	No Invariant preserved without controller
5	ζ_1	-	Yes
5	ζ_1	$\rho_1 \wedge \neg\chi_1$	No Invariant not strong enough
5	ζ_2	$\rho_2 \wedge \neg\chi_2$	Yes

Model 5: Correct *env*, *aux*, and *ctrl*

$$env \triangleq x_c := *; ? \left(x_c - x \geq \frac{v^2}{2a_n^{min}} \right) \quad (27)$$

$$aux \triangleq a_n := *; ? \left(-a_n^{min} \leq a \leq a_n^{max} \right) \quad (28)$$

$$ctrl \triangleq \text{if } \neg ok(x, v, x_c, a) \text{ then } a := *; ? C(x, v, x_c, a) \text{ fi} \quad (29)$$

$$ok \triangleq x_c - x \geq vT + \frac{a_n^{max} T^2}{2} + \frac{(v + a_n^{max} T)^2}{2a_n^{min}} \quad (30)$$

$$C(x, v, x_c, a) \triangleq a = -a_s^{min} \quad (31)$$

expressed from (16) for Model 3 with $\zeta(x, v, x_c) \equiv \zeta_1$ and $R(x_c, x_c^+) \equiv x_c \leq x_c^+$ is proven valid in KeYmaera X, thereby confirming that Model 3 is susceptible to Problem 1.

As $\models \neg\rho_1$, it follows that a stronger loop invariant is needed to not verify an exploiting *ctrl*. A possible candidate is the *env* assumptions themselves, so let $\zeta_2 \equiv v^2 \leq 2a_n^{min}(x_c - x)$. For this choice of loop invariant, ρ_2 is valid with $\zeta(x, v, x_c) \equiv \zeta_2$ and $R(x_c, x_c^+) \equiv x_c \leq x_c^+$. However, γ cannot be proven with ζ_2 since the *ctrl* actions do not maintain ζ_2 , as already illustrated in Fig. 2. Hence, the exploiting *ctrl* cannot be proven to fulfill *guarantee*. These results are summarized in the first three rows of Table 2.

The next two rows of Table 2 summarize the results of the dL formula (3) with the definitions in Model 2 and Model 4 which is proved using the loop invariant ζ_1 . Therefore it follows from loop (ii) that $\models \gamma$. By Theorem 2, $\models \neg\chi$ (22) must hold to ensure that *ctrl* is indeed verified safe. However the dL formula χ_1 (26) with ζ_1 and *env*, *aux*, *plant* defined by (11), (20), and (10), respectively, is proven in KeYmaera X and thus, it follows that Model 4 is vulnerable to Problem 2.

$$\chi_1 \equiv (x \leq x_c) \rightarrow [env; aux; plant](x \leq x_c) \quad (26)$$

The last three rows of Table 2 summarize the results of the dL formula (3) with the definitions in Model 2 and Model 5, where all three parts conjuncted together is denoted by κ . Based on the insights about Model 3 and Model 4 from Table 2, Model 5 rectifies Problem 1 and Problem 2. Similar to the previous models, the *env* assumption (27) assigns x_c such that it is possible to brake to stop before the obstacle and *aux* (28) is a black box. Unlike the previous models, the *ctrl* test *ok* in (30) not only checks whether the worst-case acceleration is safe in the current execution but also checks whether, in doing so *guarantee* is fulfilled in the next loop execution.

The dL formula κ is proved in KeYmaera X using the loop invariant $\zeta_1 \equiv x \leq x_c$. Since $R(x_c, x_c^+) \equiv x_c \leq x_c^+$ is also applicable for Model 5, it follows from $\models \neg\rho_1$ (25) that ζ_1 is not sufficiently strong to solve Problem 1. The stronger invariant candidate $\zeta_2 \equiv v^2 \leq 2a_n^{min}(x_c - x)$ is used to prove κ and since $\models \rho_2$, it is concluded that ζ_2 is sufficiently strong to solve Problem 1 for Model 5.

Finally, to confirm that Model 5 is not susceptible to Problem 2, ψ from Theorem 2 must hold. The dL formula ψ_2 (32) is proven in KeYmaera X:

$$\psi_2 \equiv \exists x. \exists v. \exists x_c. \left(\zeta(x, v, x_c, a_n^{min}) \wedge \langle env; aux \rangle \left(\neg \zeta(x, v, x_c, a) \wedge \langle plant \rangle \neg \zeta(x, v, x_c, a_n^{min}) \right) \right), \quad (32)$$

where *env*, *aux* and *plant* are as defined in (27), (28) and (10) respectively, and the loop invariant $\zeta_2 \equiv \zeta(x, v, x_c, a_n^{min})$ is a specific instantiation of the dL formula $\zeta(x, v, x_c, a)$ given by:

$$\begin{aligned} \zeta(x, v, x_c, a) \equiv & (v + aT \geq 0) \rightarrow (v + aT)^2 \leq 2a_n^{min} \left(x_c - x - vT - \frac{aT^2}{2} \right) \wedge \\ & (v + aT < 0) \rightarrow v^2 \leq 2a_n^{min}(x_c - x). \end{aligned}$$

With this result, it holds that $\models \psi_2$, and therefore it follows from Theorem 2 that $\models \neg\chi_2$ for the choice of ζ_2 . Thus, it entails that Model 5 is bereft of Problem 1 and Problem 2, as summarized in the last row of Table 2.

6 Related Work

The models considered in this paper are similar to the models used to verify the European Train Controller System (ETCS) [10]. Though not explicitly stated, the modeling pitfalls are avoided for the ETCS models by the use of an *iterative refinement process* that determines a loop invariant based on a controllability constraint. The process is used to design a correct controller rather than to verify one.

An alternative to guarantee CPS correctness is *runtime validation* [6], where runtime monitors are added to the physical implementation, monitoring whether the system deviates from its model. If it does, correctness is no longer guaranteed, and safe fallbacks are activated. However, for Model 3, the safe fallback would be activated too late since *ctrl* had already taken an unsafe action when the violation of the *env* assumptions are detected. Furthermore, the safe fallbacks might cause spurious braking for Model 4.

The issue in Model 3 is not unique to dL; the issue manifests itself similarly in reactive synthesis [1, 5]. The cause of the issue, in both paradigms, stems from the logical implication from the *env* assumptions to the *ctrl* actions and requirements. Instead of taking actions to fulfill the consequent, an exploiting *ctrl* can invalidate the premise to fulfill the implication. However, Bloem et al. [1] conclude that none of the existing approaches completely solve the problem and emphasize the need for further research.

Theorems 1 and 2 put conditions on individual components, but these conditions, in the form of the loop invariant, stem from the same global requirement. Müller et al. [7] take the other approach and start with separate requirements for each of the components to support the global requirement. The goal of the decomposition is to ease the modeling and verification effort, and not directly to validate the model. However, these methods would likely be beneficial in tandem.

The contributions of this paper give additional constraints, apart from the three implications of the loop rule, that can aid the construction of invariants. This might be useful in automatic invariant inference, which is a field of active research where loop invariants are synthesized. Furia and Meyer [3] note that the automatic synthesis of invariants based on the implementation (or the model) might be self-fulfilling, and go on to argue that the postconditions and the global requirements must be considered in the invariant synthesis. This paper, however, suggests that, for certain models, it might not be sufficient to consider only the postconditions in the invariant synthesis.

7 Conclusion

Modeling errors present a risk of unsound conclusions from provably safe erroneous models, if used in the safety argument of safety-critical systems. This paper formulates and proves conditions in Theorem 1 and Theorem 2 that, when fulfilled, help identify and avoid two kinds of modeling errors that may result in a faulty controller being proven safe. Furthermore, the formulated conditions aid in finding a loop invariant which is typically necessary to verify the safety of hybrid systems.

Using a running example of an automated driving controller, the problematic cases are shown to exist in practical CPS designs. The formulated conditions are then applied to the erroneous models to show that the errors are captured. Finally, the errors are rectified to obtain a correct model, which is then proved using a loop invariant that satisfies the formulated conditions, thus ensuring absence of the two modeling errors discussed in this paper.

A natural extension of this work will be to investigate also other kinds of modeling errors that might arise in the verification of complex CPS designs. Moreover, it would also be beneficial to investigate the connection between loop invariants and differential invariants, which are used to prove properties about hybrid systems with differential equations without their closed-form solutions.

References

- [1] Bloem, R., Ehlers, R., Jacobs, S., Könighofer, R.: How to handle assumptions in synthesis. In: Chatterjee, K., Ehlers, R., Jha, S. (eds.) Proceedings 3rd Workshop on Synthesis, SYNT. EPTCS, vol. 157 (2014). <https://doi.org/10.4204/EPTCS.157.7>
- [2] Fulton, N., Mitsch, S., Quesel, J.D., Völz, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: International Conference on Automated Deduction. Springer (2015)
- [3] Furiá, C.A., Meyer, B.: Inferring Loop Invariants Using Postconditions, pp. 277–300. Springer (2010). https://doi.org/10.1007/978-3-642-15025-8_15
- [4] Koopman, P., Kane, A., Black, J.: Credible autonomy safety argumentation. In: 27th Safety-Critical Systems Symposium (2019)
- [5] Majumdar, R., Piterman, N., Schmuck, A.K.: Environmentally-friendly GR(1) synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 229 – 246 (2019). https://doi.org/10.1007/978-3-030-17465-1_13
- [6] Mitsch, S., Platzer, A.: ModelPlex: verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design* **49** (2016). <https://doi.org/10.1007/s10703-016-0241-z>
- [7] Müller, A., Mitsch, S., Retschitzegger, W., Schwinger, W., Platzer, A.: Tactical contract composition for hybrid system component verification. In: International Journal on Software Tools for Technology Transfer. vol. 20 (2018). <https://doi.org/10.1007/s10009-018-0502-9>
- [8] Platzer, A.: Logics of dynamical systems. In: 27th Annual IEEE Symposium on Logic in Computer Science. pp. 13–24. IEEE (2012). <https://doi.org/10.1109/LICS.2012.13>
- [9] Platzer, A.: Logical foundations of cyber-physical systems, vol. 662. Springer (2018). <https://doi.org/10.1007/978-3-319-63588-0>
- [10] Platzer, A., Quesel, J.D.: European train control system: A case study in formal verification. In: Breitman, K., Cavalcanti, A. (eds.) Formal Methods and Software Engineering. Springer (2009). https://doi.org/10.1007/978-3-642-10373-5_13
- [11] Selvaraj, Y., Ahrendt, W., Fabian, M.: Formal development of safe automated driving using differential dynamic logic. arXiv:2204.06873 (2022)
- [12] Selvaraj, Y., Krook, J.: model-pitfalls-dl (Jul 2022). <https://doi.org/10.5281/zenodo.6821673>