On proving that an unsafe controller is not proven safe

Citation for the original published paper (version of record):
Selvaraj, Y., Krook, J., Ahrendt, W. et al (2024). On proving that an unsafe controller is not proven safe. Journal of Logical and Algebraic Methods in Programming, 137.
http://dx.doi.org/10.1016/j.jlamp.2023.100939

J. Log. Algebraic Methods Program. 137 (2024) 100939

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp

On proving that an unsafe controller is not proven safe ☆

Yuvaraj Selvaraj [a,b,*], Jonas Krook [a,b], Wolfgang Ahrendt [b], Martin Fabian [b]

[a] Zenseact, Lindholmspiren 2, 417 56, Göteborg, Sweden
[b] Chalmers University of Technology, Chalmersplatsen 4, 412 96, Göteborg, Sweden

ARTICLE INFO

Keywords:
Hybrid systems
Automated driving
Formal verification
Loop invariant
Theorem proving

ABSTRACT

Cyber-physical systems are often safety-critical and their correctness is crucial, such as in the case of automated driving. Using formal mathematical methods is one way to guarantee correctness and improve safety. Although these methods have shown their usefulness, care must be taken because modelling errors might result in proving a faulty controller safe, which is potentially catastrophic in practice. This paper deals with two such modelling errors in *differential dynamic logic*, a formal specification and verification language for *hybrid systems*, which are mathematical models of cyber-physical systems. The main contributions are to provide conditions under which these two modelling errors cannot cause a faulty controller to be proven safe, and to show how these conditions can be proven with help of the interactive theorem prover KeYmaera X. 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. Furthermore, the relation between such a loop invariant and the characterisation of the *maximal control invariant set* is discussed.

1. Introduction

Cyber-physical systems (CPS) [1] typically consist of a digital *controller* that interacts with a physical dynamic system and is 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 is to use formal verification [2,3]. Formal verification requires a formal model of the CPS, and an increasingly popular family of models of CPS are *hybrid* systems [4–6], which are mathematical models that combine discrete and continuous dynamics.

To reason about the correctness of a CPS, hybrid systems can model the different components of the CPS and their interactions [7], thus capturing the overall *closed-loop* behaviour. 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 behaviours of the surrounding world in which the controller operates, thereby

☆ 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.

* Corresponding author at: Zenseact, Lindholmspiren 2, 417 56, Göteborg, Sweden.
E-mail addresses: yuvaraj.selvaraj@zenseact.com (Y. Selvaraj), jonas.krook@zenseact.com (J. Krook), ahrendt@chalmers.se (W. Ahrendt), fabian@chalmers.se (M. Fabian).

https://doi.org/10.1016/j.jlamp.2023.100939
Received 25 February 2023; Received in revised form 8 December 2023; Accepted 14 December 2023

Available online 19 December 2023

defining the *operational domain*. The goal for the controller is to choose control actions such that the requirements are fulfilled for *all* possible behaviours of the hybrid system.

Typically, the environment is modelled using nondeterminism to capture all possible behaviours of the surrounding world. However, assumptions on the environment behaviour are necessary to limit the operational domain and remove behaviours 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 [8]. 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 behaviours 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 modelling errors. In such cases, if a controller is verified to be safe, this leads to unsafe conclusions that might be catastrophic in practice.

This paper deals with two such modelling 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 the controller that exploits the friendliness. Then, a faulty controller may be proven safe since the environment reacts to accommodate 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 behaviours in which any action by the controller is needed. In this case, the assumptions over-constrain the allowed behaviours. For example, if the assumptions restrict the behaviour 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* behaviours, 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 modelling errors, thus resulting in potentially catastrophic operation of the CPS in practice. Modelling 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 modelling 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 behaviour of the closed-loop system within the operational domain and are expressed in some logical formalism to apply formal verification. *Differential dynamic logic* (dL) [9,10] is a specification and verification language that can be used to formally describe and verify hybrid systems. The interactive theorem prover KeYmaera X [6] implements a sound proof calculus [9,10] for dL and can thus mathematically prove that the models fulfil 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 modelling 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 modelling errors. It is also shown that this loop invariant is equivalent to the characterisation of the *maximal control invariant set*, an important property within control engineering for which closed-form expressions are given by [11]. 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 modelling errors.

This paper extends our previous work [12] with the following contributions:

- extended preliminaries in Section 2 including descriptions about proof rules of dL sequent calculus [9];
- extensive discussion on rigorous modelling concepts and vacuous truth in Section 4.1;
- Section 5.1 shows that the theorems can be applied to models other than the running example using the verified European Train Control System models [13];
- Section 6 shows equivalence between the sufficiently strong loop invariant, and the characterisation of the maximal control invariant set [11];
- extensive coverage of related works in Section 7 including a discussion on the similarities of exploiting controllers in dL to those in reactive synthesis [14].

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 term[1], and P and Q are formulas of first-order logic of real arithmetic (FOL)[2]:

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

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

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

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

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

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

The sequential composition $\alpha; \beta$ expresses that β starts executing after α has finished. The *nondeterministic repetition* α^* expresses that α repeats n times for any $n \in \mathbb{N}_0$. The *nondeterministic choice* operation expresses that the HP $\alpha \cup \beta$ can nondeterministically choose to follow either α or β. The *nondeterministic assignment* $x := *$ assigns an arbitrary (real) number to x. 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. However, if 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 behaviours* of the HP. Pragmatically, nondeterministic assignment and the test action are often combined in HPs, because this allows a declarative style of HP programming. For instance, $x := *; ?Q$ intuitively means "choose x such that Q holds". Test actions can also be combined with sequential composition and the choice operation to define (deterministic) *if-statements* as:

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

Here, the two test actions are used to discard (by aborting) unwanted computations, like choosing to not execute α even if P is true. In a similar fashion, test actions can be combined with sequential composition and nondeterministic repetition to define (deterministic) *while-statements* as:

$$\text{while}\,(P)\ \text{do}\ \alpha\ \text{od} \equiv (?P; \alpha)^*; (?\neg P) \tag{2}$$

Here, $?\neg P$ discards executions where the chosen number of iterations is too small, whereas $?P$ discards executions where the chosen number of iterations is too large.

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 is a first-order formula that restricts 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 [9,10]. 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):

$$\theta ::= e = \tilde{e} \mid e \geq \tilde{e} \mid \neg\phi \mid \phi \wedge \psi \mid \forall x\, \phi \mid [\alpha]\, \phi \tag{3}$$

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 $(\!|[\alpha]\,\phi|\!) = \{\omega \in \mathcal{S} : \forall \nu \in \mathcal{S}.\, (\omega, \nu) \in [\![\alpha]\!] \to \nu \in (\!|\phi|\!)\}$ and $(\!|\forall x\, \phi|\!) = \{\omega \in \mathcal{S} : \nu \in (\!|\phi|\!) \text{ for all } \nu \in \mathcal{S} \text{ that agree with } \omega \text{ except on } x\}$. 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, $>, \leq, <, \vee, \to, \leftrightarrow, \exists x$ are defined using combinations of the operators in (3). A dL formula θ is *valid*, denoted $\vDash \theta$, if $(\!|\theta|\!) = \mathcal{S}$.

The logic dL and the interactive theorem prover KeYmaera X [6] support the specification and verification of hybrid systems. The dL formula $(\mathit{init}) \to [\,\alpha\,](\mathit{guarantee})$ can 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 attempts to construct a proof for the

formula, successively decomposing it into several sub-goals according to the sound proof rules of dL [9,10], until all leaves of the proof tree are trivially true and can thus be "closed". As this proof technique for dL, and its realisation in KeYmaera X, play a central role for this work, a brief introduction to some of its principles is given in the following.

KeYmaera X uses a *sequent calculus*, which is a set of (sequent) rules used to construct proofs. A (dL) sequent has the form $\Gamma \vdash \Delta$ (where Γ and Δ are finite sets of dL formulas), and is semantically equivalent to the formula $\bigwedge_{\phi\in\Gamma}\phi \rightarrow \bigvee_{\psi\in\Delta}\psi$. To prove a sequent $\Gamma \vdash \Delta$ means to prove one of the formulas in Δ, under the assumption that all formulas in Γ are true. If the overall objective is to prove a single (dL) formula ϕ, then the initial goal will be the sequent $\vdash \phi$, i.e., Γ is empty, and Δ only contains ϕ. This goal will be successively decomposed by rules during proof construction. Some of the rules only exploit the propositional, or first-order, structure of a formula. One example of such a rule is:

$$\wedge\mathrm{R}\ \frac{\Gamma \vdash P, \Delta \qquad \Gamma \vdash Q, \Delta}{\Gamma \vdash P \wedge Q, \Delta}$$

Other rules of the sequent calculus for dL are concerned with decomposing hybrid programs appearing in the modalities of dL (like α in $[\alpha]\phi$ or $\langle\alpha\rangle\phi$). Concerning nondeterministic choice and sequential composition, for brevity the sequent rules here are not given here, but just the equivalences that are exploited by the rules (in a left-to-right fashion):

$$[\alpha \cup \beta]\phi \leftrightarrow [\alpha]\phi \wedge [\beta]\phi, \tag{4}$$

$$[\alpha; \beta]\phi \leftrightarrow [\alpha][\beta]\phi. \tag{5}$$

Through sufficiently many applications of these equivalences ((4), (5)), the leading modal operator "$[\cdot]$" will sooner or later contain elementary statements that can be resolved by other rules. For instance, if the elementary statement is a test action of the form $?P$, the following equivalence applies:

$$[?P]\phi \leftrightarrow (P \rightarrow \phi). \tag{6}$$

The rules for resolving all elementary statements are not presented here, but it is worth noting that deterministic and nondeterministic assignment, $x := e$ and $x := *$, are resolved by substitution and quantification, respectively. More details about the proof rules are found in [9] and their implementation in KeYmaera X in [15].

Note that in all of the above equivalences, ϕ may well contain further modal operators. For instance, a formula of the form $[?P;\delta]\psi$ would first be transformed to $[?P][\delta]\psi$, using (5), and then transformed to $(P \rightarrow [\delta]\psi)$, using (6). In such a fashion, proofs of arbitrary HPs can be constructed by decomposing the HP into elementary statements, which are then resolved individually.

The treatment of nondeterministic repetition (*loops*) deserves special attention. One of the important rules for handling an HP of the form α^* is the following:

$$\mathrm{loop}\ \frac{\Gamma \vdash \zeta, \Delta \qquad \zeta \vdash [\alpha]\zeta \qquad \zeta \vdash \phi}{\Gamma \vdash [\alpha^*]\phi, \Delta}$$

Here, ζ is a *loop invariant*, provided by the user or by some heuristic, to inductively reason about an arbitrary number of repetitions of α. Given a (candidate for a) loop invariant ζ, applying the loop invariant rule will split the current proof branch into three branches:

(i) $\Gamma \vdash \zeta, \Delta$, i.e., the invariant holds before executing the loop,
(ii) $\zeta \vdash [\alpha]\zeta$, i.e., the invariant remains true after one iteration of α, given that the invariant was true beforehand,
(iii) $\zeta \vdash \phi$, i.e., the invariant implies the post-condition of the loop, ϕ.

Finding suitable loop invariants is a well known bottle-neck of program verification. This is very true also for the verification of HPs using dL. The work presented in this article is however devoted to a different problem, namely that an unsuitable loop invariant may lead to proving an erroneous system, with a faulty controller, correct. This problem is treated in the following sections.

3. Problem scope

This paper considers hybrid systems with closed-loop feedback control as described by Model 1. The dL formula (7) models the CPS as an HP that repeatedly executes in a loop and expresses the requirement on the CPS by the formula *guarantee*. The HP in (7) 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.

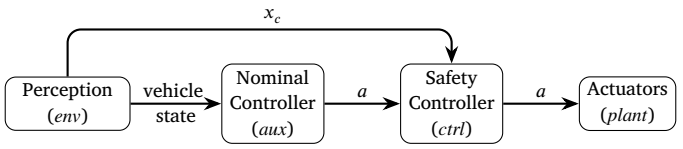


Fig. 1. Architecture of the automated driving feature.

Model 1: The general model considered.

$$(init) \rightarrow [(env; aux; ctrl; plant)^*](guarantee) \tag{7}$$

$$env \triangleq e := *; ?P(s,e,a) \tag{8}$$

$$aux \triangleq a := *; ?Q(s,e,a) \tag{9}$$

$$ctrl \triangleq \text{if } \neg ok(s,e,a) \text{ then } a := *; ?C(s,e,a) \text{ fi} \tag{10}$$

$$plant \triangleq \tau := 0; s' = f(s,e,a), \tau' = 1 \,\&\, F(s,e,a,\tau) \wedge \tau \leq T \tag{11}$$

The environment (*env*) in (8) describes the environment behaviour 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* (9) 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.

The controller's (*ctrl*) task is to ensure that the requirement *guarantee* is fulfiled and is modelled as an if-statement as seen in (10). 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* (11), 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 [3].

In the most abstract setting, the parameterised 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 modelled as an HP of the general form in Model 1. The safety requirement is for the ego-vehicle to safely stop for obstacles, even stationary, 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 (9) 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 modelled as a 2nd order system, a double integrator with position x and velocity v of the ego-vehicle as the dynamic states, and the acceleration a as the single control input, as seen in (14) 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 (7) is refined with concrete descriptions of *env*, *aux*, and *ctrl* to illustrate the modelling errors where a faulty controller can be proven safe. However, *init*, *plant*, and *guarantee* remain unchanged in the subsequent models and are shown in Model 2. The initial condition *init* (12) 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 $[(env; aux; ctrl; plant)^*]$, 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 *guarantee* (13), which says that the obstacle's position may not be exceeded.

Model 2: Example hybrid system.

$$init \triangleq v = 0 \land x \leq x_c \land a_s^{min} > 0 \land a_n^{max} > 0 \land a_n^{min} > 0 \land a_s^{min} > a_n^{min} \land T > 0 \tag{12}$$

$$guarantee \triangleq (x \leq x_c) \tag{13}$$

$$plant \triangleq \tau := 0; x' = v, v' = a, \tau' = 1 \;\&\; v \geq 0 \land \tau \leq T \tag{14}$$

4. Discovering modelling errors

This section presents two erroneous models to illustrate how a faulty *ctrl* can be proven safe with respect to *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 5, 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 (15) and (16) 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* (18) 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 (19) sets the deceleration to its maximum. This maximum deceleration is a symbolic value, parameterised over the other model variables.

Model 3: *ctrl* is exploiting.

$$env \triangleq x_c := *; ?\left(x_c - x \geq \frac{v^2}{2a_n^{min}}\right) \tag{15}$$

$$aux \triangleq a := *; ?(-a_n^{min} \leq a \leq a_n^{max}) \tag{16}$$

$$ctrl \triangleq \text{if } \neg ok(x, v, x_c, a) \text{ then } a := *; ?C(x, v, x_c, a) \text{ fi} \tag{17}$$

$$ok(x, v, x_c, a) \triangleq \left(x_c - x \geq vT + \frac{a_n^{max} T^2}{2}\right) \tag{18}$$

$$C(x, v, x_c, a) \triangleq a = -a_s^{min} \tag{19}$$

Denote by θ the dL formula (7) together with the definitions of Model 2 and Model 3. θ is proved [16] 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.

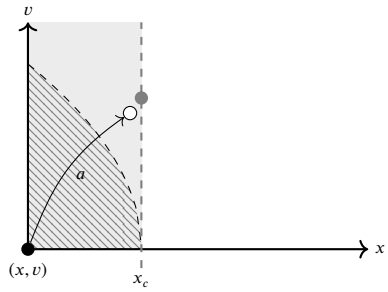
The *env* assumption (15) discards executions where the distance between the obstacle position x_c and the ego-vehicle position x is less than the minimum possible braking distance of the ego-vehicle. This assumption is reasonable as it only discards situations where it is physically impossible for *ctrl* to safely stop the vehicle. Still, infinitely many *env* behaviours are possible since x_c is nondeterministically assigned any value that fulfils the assumption. Among other behaviours, this allows x_c to remain unchanged between subsequent executions, 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. In a scenario where x_c marks an obstacle, this would mean that, in the model, *env* can be forced by *ctrl* to move the obstacle when ego-vehicle comes too close.

Consider a state $\omega_0 \in \langle\!|\zeta_1|\!\rangle$, illustrated in Fig. 2a, such that

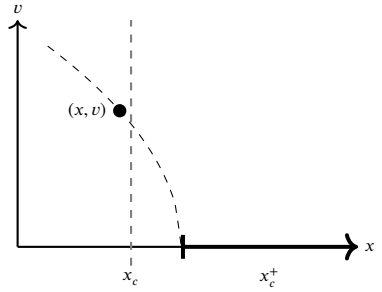
$$\omega_0(x) = 0 \qquad \omega_0(x_c) = 1 \qquad \omega_0(T) = 1 \qquad \omega_0(v) = 0 \qquad \omega_0(a) = 1.8 \qquad \omega_0(a_n^{max}) = 2 \qquad \omega_0(a_n^{min}) = 3\,.$$

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 $(\omega_0, \omega_0) \in [\![env]\!]$ 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 labelled 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 $(\omega_0, \omega_0) \in [\![aux]\!]$. 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 grey 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 (18) that $\omega_0 \in \langle\!|ok(x, v, a_n, a)|\!\rangle$. Thus, $(\omega_0, \omega_0) \in [\![ctrl]\!]$. Let $\omega_1 = \omega_0(x := 0.9, v := 1.8)$. Now it holds that $(\omega_0, \omega_1) \in [\![plant]\!]$, 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.

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 (15)



(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.

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 (15). Semantically, as $x_c - x = 0.1 < 2^2/(2 \times 3) = v^2/(2a_n^{min})$, it follows that $(\omega_1, \omega_1) \notin [\![env]\!]$ so x_c cannot be kept constant between iterations.

To summarise, it holds that $\omega_0 \in (\!|\zeta_1|\!)$, $(\omega_0, \omega_1) \in [\![env; aux; ctrl; plant]\!]$, and $\omega_1 \in (\!|\zeta_1|\!)$. 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 fulfils ζ_1, and therefore also fulfils *guarantee*. Since there exists no control action allowed by the system dynamics in the assumed operational domain that can fulfil *guarantee* from ω_1, the decision made by *ctrl* is unsafe in this case. However, since $(\omega_1, \omega_1) \notin [\![env]\!]$, Model 3 can be proven to fulfil *guarantee* with this faulty *ctrl*.

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

Problem 1. *How can a requirement be formalised that guarantees, together with the dL formula* (7)*, that the controller does not enforce particular environment actions to establish safety?*

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 (15) 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. So why was it possible to prove Model 3 with an invariant as weak as ζ_1? The reason is that this weak invariant is preserved only because all environment behaviours which would violate a more reasonable invariant are discarded. 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., *a good loop invariant should imply the env assumptions*, such that the loop invariant does not force the environment to react.

The assumption $x_c - x \geq \frac{v^2}{2a_n^{min}}$ in (15) corresponds to P in the generalised Model 1. Therefore, it can be hypothesised 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 behaviour does not characterise friendly behaviour in all models. In order to generalise the above to a larger class of behaviours environments should be allowed to expose, first express the condition $\zeta \rightarrow P$ in a different way, as follows: $\forall s.\, \forall e.\, \forall e_1.\, \Big(\zeta(s,e) \wedge e = e_1 \rightarrow \langle env \rangle\, (e = e_1)\Big)$. Note that the diamond operator is used here ("$\langle\alpha\rangle$"), instead of the box operator, thereby stating that "there exists a run", instead of "for all runs". The formula says that, if the invariant holds, then *env* has the *option* to leave the old and the new e identical. Now, it is possible to generalise the option to not move $(e = e_1)$ to a general relation R of environment values, modelling actions which the environment should always be allowed to take, i.e., the model should not force the environment out of actions according to R.

In general, the relation between two *env* values (e_0, e_1) can be any relation $R \subseteq \mathbb{R} \times \mathbb{R}$. Note that R only defines certain behaviours in the assumed operational domain. In Model 3, the exploiting controller could be proven safe because the environment behaves *friendly* by discarding some behaviours characterised by R. This is illustrated in Fig. 2b where x_c cannot be kept constant as $(\omega_1, \omega_1) \notin [\![env]\!]$.

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 $(\omega_0, \omega_1) \notin [\![env]\!]$, then the environment *env* is ***friendly*** w.r.t. the relation R. Thus, *env* is ***unfriendly*** if $(e_0, e_1) \in R \rightarrow (\omega_0, \omega_1) \in [\![env]\!]$ 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 generalised to include the relation R to describe the existence of an unfriendly *env* as:

$$\rho \equiv \forall s.\, \forall e.\, \forall e_1.\, \Big(\zeta(s,e) \land R(e,e_1) \rightarrow \langle env \rangle\, (e = e_1)\Big)\,, \tag{20}$$

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

The loop invariant $\zeta_1 \equiv x \leq x_c$ is used to prove the dL formula (7) with the definitions of Model 2 and Model 3. Thus, it follows that $\vDash \zeta_1 \rightarrow [env; aux; ctrl; plant]\, \zeta_1$ holds by (ii). But, ζ_1 is not strong enough to prevent control actions that exploit friendly *env* behaviours. 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 behaviours from ω_1 for ζ_1 to be preserved later on. These discarded behaviours 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 there exists a loop invariant ζ that is preserved by the loop body, i.e. $\vDash \gamma$, with

$$\gamma \equiv \forall s.\, \forall e.\, \Big(\zeta(s,e) \rightarrow [env; aux; ctrl; plant]\, \zeta(s,e)\Big)\,, \tag{21}$$

while at the same time the following holds

$$\exists s.\, \exists e_0.\, \exists e.\, \Big(\zeta(s,e_0) \land R(e_0,e) \land \langle aux; ctrl; plant \rangle \neg\, \zeta(s,e)\Big)\,. \tag{22}$$

(Note that *aux*, *ctrl*, and *plant* can read s and e, as well as modify s, but not modify e.)

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 (21) be the dL formula from the inductive step (ii) of the loop invariant proof rule, and let ρ be as defined by (20). If $\gamma \land \rho$ 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 [16] in KeYmaera X:

$$\gamma \land \rho \rightarrow \forall s.\, \forall e_0.\, \forall e.\, \Big(\zeta(s,e_0) \land R(e_0,e) \rightarrow [aux; ctrl; plant]\, \zeta(s,e)\Big)\,. \tag{23}$$

This asserts that the loop invariant is strong enough to prevent *ctrl* from exploiting *env*'s friendly behaviour because the clause implied by $\gamma \land \rho$ in (23) is the negation of (22). □

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.

On Rigorous Modelling and Vacuous Truth

For the specific model instance considered in this section, it is paramount that the environment may be unchanged, or else be too friendly. As shown, a controller could otherwise exploit the environment. The environment is too friendly because, in certain states, the dL formula is vacuously true. From a modelling perspective, the formula may become vacuously true because there is a test without a default case. A default case is a branch of a nondeterministic choice that may never become false, and it is considered good modelling practice to always include a default case to avoid vacuously true statements [17]. Adding a default case works by ensuring that there is always at least one execution that the environment may execute.

A simple default case is the *skip* action. It is modelled by adding $\cup ?\text{true}$ to the test that might be vacuously true. In the case of (8), adding a skip action, as shown in Model 4, ensures that there is always an execution where the environment variable e is not changed by the environment. That is, adding a skip action always allows the environment to be constant. Hence, adding the skip action as a default case for the environment in Model 5 would not allow the controller to cheat.

Model 4: Default case for *env*.

$$(init) \rightarrow [((env \cup ?\,\text{true}); aux; ctrl; plant)^*]\,(guarantee) \tag{24}$$

In this particular case, adding the skip action prevents the environment from being too friendly. Other models might require a more complicated default case. Mistakes might be introduced in complicated models, and it might be difficult to identify whether best practice is being followed or not. On the contrary, the method presented in this section is helpful because it will ensure that the complete model cannot be proven if the environment is too friendly, and it is much more general than adding a single default case.

The method presented in this section can completely capture the effect of adding a skip action. If the relation R is reflexive, then (20) and (21) imply that the model resulting from adding skip to *env*, as in (24), is valid.

Lemma 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 reflexive relation over the domain of e. Let γ (21) be the dL formula from the inductive step (ii) of the loop invariant proof rule, and let ρ be as defined by (20). If $\gamma \wedge \rho$ is valid, then it follows that a skip action can be added in parallel to the environment, as in (24), without affecting the satisfiability of the dL formula γ.*

Proof. The following dL formula is proved [18] in KeYmaera X:

$$\gamma \wedge \rho \wedge \forall e.\, R(e,e) \rightarrow \forall s.\, \forall e.\, \Big(\zeta(s,e) \rightarrow [(env \cup ?true);\, aux;\, ctrl;\, plant]\, \zeta(s,e)\Big)\,.$$

This asserts that the addition of a skip action does not alter the satisfiability of the loop step if R is reflexive. □

Lemma 1 shows that R being reflexive is a sufficient condition to imply a skip action. Lemma 2 shows that proving the model with the skip action implies that R can be extended to be reflexive.

Lemma 2. *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_0 be a relation over the domain of e. Let γ be the dL formula from the inductive step (ii) of the loop invariant proof rule applied to the general model with skip Model (24), i.e.,*

$$\gamma \equiv \forall s.\, \forall e.\, \Big(\zeta(s,e) \rightarrow [(env \cup ?true);\, aux;\, ctrl;\, plant]\, \zeta(s,e)\Big)\,.$$

If γ is valid, for a controller ctrl guaranteed to not exploit a friendly environment w.r.t. R_0, then ctrl is also guaranteed to not exploit a friendly environment w.r.t. $R = R_0 \cup I$.

Proof. Let the negation of (22) be denoted by ξ. That is,

$$\xi \equiv \forall s.\, \forall e_0.\, \forall e.\, \Big(\zeta(s,e_0) \wedge R(e_0,e) \rightarrow [aux;\, ctrl;\, plant]\, \zeta(s,e)\Big)\,.$$

It is to be shown that γ implies ξ for all $(e,e) \in R \subseteq R_0 \cup I$. This is done by taking an arbitrary $\omega \in (\!|\gamma|\!)$ and showing that $\omega \in (\!|\xi|\!)$.

Assume that $\omega \in \mathcal{S}$ satisfies γ, i.e., $\omega \in (\!|\gamma|\!)$. By the semantics of dL, this means that for all $\nu \in \mathcal{S}$ that coincide with ω except on s, e_0, and e, it holds that $\nu \in (\!|\zeta(s,e) \rightarrow [(env \cup ?true);\, aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$. Consider an arbitrary such ν. Two cases will be distinguished; either $\nu \in (\!|R(e,e_0)|\!)$ or $\nu \notin (\!|R(e,e_0)|\!)$. In the first case, it trivially follows that $\nu \in (\!|\zeta(s,e_0) \wedge R(e_0,e) \rightarrow [aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$.

In the second case, $\nu \notin (\!|R(e,e_0)|\!)$. Again, two cases can be distinguished; either $\nu \in (\!|R_0(e_0,e)|\!)$ or $\nu \in (\!|I(e_0,e)|\!)$. If $\nu \in (\!|R_0(e_0,e)|\!)$, it follows that $\nu \in (\!|\zeta(s,e_0) \wedge R(e_0,e) \rightarrow [aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$, because *ctrl* is guaranteed to not exploit a friendly environment w.r.t. R_0. If $\nu \in (\!|I(e_0,e)|\!)$, it follows that $e_0 = e$ in the state ν. Recall that $\nu \in (\!|\zeta(s,e) \rightarrow [(env \cup ?true);\, aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$. If $\nu \notin (\!|\zeta(s,e)|\!)$, then $\nu \notin (\!|\zeta(s,e_0)|\!)$, and it entails that $\nu \in (\!|\zeta(s,e_0) \wedge R(e_0,e) \rightarrow [aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$. If, on the other hand, $\nu \in (\!|\zeta(s,e)|\!)$, then $\nu \in (\!|[(env \cup ?true);\, aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$. As $(\nu,\nu) \in [\![(env \cup ?true)]\!]$, it follows that $\nu \in (\!|[aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$. Recall also that $e_0 = e$ in ν. Thus, $\nu \in (\!|\zeta(s,e_0)|\!)$, $\nu \in (\!|R(e_0,e)|\!)$ and $\nu \in (\!|[aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$, which means that $\nu \in (\!|\zeta(s,e_0) \wedge R(e_0,e) \rightarrow [aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$.

In all cases, $\nu \in (\!|\zeta(s,e_0) \wedge R(e_0,e) \rightarrow [aux;\, ctrl;\, plant]\, \zeta(s,e)|\!)$, so it follows that $\omega \in (\!|\xi|\!)$. As ω was chosen arbitrarily, it follows that *ctrl* is guaranteed to not exploit a friendly environment w.r.t. R. □

It is now clear that adding a skip action as a default case in *env* is in some sense equivalent to ensuring that R is reflexive.

4.2. Unchallenged controller

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

Consider Model 5, identical to Model 3, except for *aux* ((25) and (26)). As before, *aux* is a black box. However, in addition to the acceleration bounds, *aux* also fulfils a design requirement *req* given by (26). *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* (18) is true; if not, the controller action C (19) sets the maximal possible deceleration.

Model 5: *ctrl* is unchallenged.

$$aux \triangleq a := *; ?\left(-a_n^{min} \leq a \leq a_n^{max} \wedge req\right) \tag{25}$$

$$req \triangleq \left((v + aT \geq 0) \rightarrow vT + \frac{aT^2}{2} \leq \frac{v^2}{2a_n^{min}}\right) \wedge \left((v + aT < 0) \rightarrow a \leq -a_n^{min}\right) \tag{26}$$

To verify that *ctrl* fulfils *guarantee* (13), the dL formula (7) together with the definitions in Model 2 and Model 5 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 irrespective of *ctrl*'s actions, and hence *ctrl* is never verified. This manifests itself in Model 5; *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 fulfils *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 a requirement be formalised that guarantees, together with the dL formula (7), that safety is not established due to 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 (7) 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 fulfils *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 5.

Observe that if *ctrl* is removed from the dL formula (7) 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.,

$$\neg\chi \equiv \exists s.\, \exists e.\, \exists a.\, \zeta \wedge \langle env; aux; plant\rangle \neg\zeta\ , \tag{27}$$

can be proved to ascertain the absence of Problem 2 in the proof of (7).

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 $\neg\chi$ (27) might not be beneficial in practice. While failed attempts to prove $\neg\chi$ might illuminate modelling 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*

$$\psi \equiv \exists s.\, \exists e.\, \exists a_1.\, \left(\zeta(s, e, a_1) \wedge \langle env; aux\rangle\left(\neg\zeta(s, e, a)\ \wedge \langle plant\rangle \neg\zeta(s, e, a_1)\right)\right)\ . \tag{28}$$

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

Proof. The following dL formula is proved [16] in KeYmaera X:

$$\psi \rightarrow \exists s.\, \exists e.\, \exists a_1.\, \zeta(s, e, a_1) \wedge \langle env; aux; plant\rangle \neg\zeta(s, e, a_1)\ . \tag{29}$$

The dL formula ψ (28) 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 (29) asserts that *ctrl* is challenged by Definition 3. □

By the conjunction of ψ (28) to a dL formula of the form (7), 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*

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

Model	Loop invariant	Conjuncts	Valid	Reason
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 fulfil requirement
5	ζ_1	–	Yes	Unchallenged controller
5	ζ_1	$\neg\chi_1$	No	Invariant preserved without controller
6	ζ_1	–	Yes	
6	ζ_1	$\rho_1 \wedge \neg\chi_1$	No	Invariant not strong enough
6	ζ_2	$\rho_2 \wedge \neg\chi_2$	Yes	

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 5 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 6 where the errors are corrected.

The HPs and the KeYmaera X proofs are available from [16].

The dL formula (7) 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 (ii) that $\vDash \gamma$, 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

$$\neg\rho_1 \equiv \exists x.\, \exists v.\, \exists x_c.\, \exists x_c^+.\, \neg\Big(x \leq x_c \wedge x_c \leq x_c^+ \rightarrow \big\langle x_c := *;\ ?(v^2 \leq 2a_n^{min}(x_c - x))\big\rangle\,(x_c = x_c^+)\Big)\,, \tag{30}$$

expressed from (20) 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 $\vDash \neg\rho_1$, it follows that a stronger loop invariant is needed to not verify an exploiting *ctrl*. A possible candidate is the *env* assumption (15), 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 fulfil *guarantee*. These results are summarised in the first three rows of Table 2.

Model 6: Correct *env*, *aux*, and *ctrl*.

$$env \triangleq\ x_c := *;\ ?\left(x_c - x \geq \tfrac{v^2}{2a_n^{min}}\right) \tag{31}$$

$$aux \triangleq\ a_n := *;\ ?\left(-a_n^{min} \leq a \leq a_n^{max}\right) \tag{32}$$

$$ctrl \triangleq \text{if } \neg ok(x, v, x_c, a) \text{ then } a := *;\ ?C(x, v, x_c, a) \text{ fi} \tag{33}$$

$$ok \triangleq x_c - x \geq vT + \frac{a_n^{max}T^2}{2} + \frac{\left(v + a_n^{max}T\right)^2}{2a_n^{min}} \tag{34}$$

$$C(x, v, x_c, a) \triangleq a = -a_s^{min} \tag{35}$$

The next two rows of Table 2 summarise the results of the dL formula (7) with the definitions in Model 2 and Model 5 which is proved using the loop invariant ζ_1. Therefore it follows from (ii) that $\vDash \gamma$. By Theorem 2, $\vDash \neg\chi$ (27) must hold to ensure that *ctrl* is indeed verified safe. However the dL formula χ_1 (36) with ζ_1 and *env*, *aux*, *plant* defined by (15), (25), and (14), respectively, is proven in KeYmaera X and thus, it follows that Model 5 is vulnerable to Problem 2.

$$\chi_1 \equiv (x \leq x_c) \rightarrow [env; aux; plant]\,(x \leq x_c) \tag{36}$$

The last three rows of Table 2 summarise the results of the dL formula

$$\kappa \equiv (init) \rightarrow [(env; aux; ctrl; plant)^*]\,(guarantee), \tag{37}$$

where the definitions of *env*, *aux*, and *ctrl* are given in Model 6 and the rest in Model 2. Based on the insights about Model 3 and Model 5 from Table 2, Model 6 rectifies Problem 1 and Problem 2. Similar to the previous models, the *env* assumption (31) assigns x_c such that it is possible to brake to stop before the obstacle and *aux* (32) is a black box. Unlike the previous models, the *ctrl* test

ok in (34) 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 (37) 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 6, it follows from $\vDash \neg\rho_1$ (30) that ζ_1 is not sufficiently strong to solve Problem 1. The stronger invariant candidate

$$\zeta_2 \equiv v^2 \leq 2a_n^{min}(x_c - x) \tag{38}$$

is used to prove κ (37) and since $\vDash \rho_2$, it is concluded that ζ_2 is sufficiently strong to solve Problem 1 for Model 6.

Finally, to confirm that Model 6 is not susceptible to Problem 2, ψ from Theorem 2 must be valid. The dL formula ψ_2 (39) is proven in KeYmaera X:

$$\psi_2 \equiv \exists x.\, \exists v.\, \exists x_c.\, \left(\zeta(x, v, x_c, a_n^{min}) \wedge \langle env; aux\rangle \left(\neg\zeta(x, v, x_c, a) \wedge \langle plant\rangle \neg\zeta(x, v, x_c, a_n^{min})\right)\right), \tag{39}$$

where *env*, *aux* and *plant* are as defined in (31), (32) and (14) respectively. Note that the stronger 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:

$$\zeta(x, v, x_c, a) \equiv (v + aT \geq 0) \rightarrow (v + aT)^2 \leq 2a_n^{min}\left(x_c - x - vT - \frac{aT^2}{2}\right) \wedge (v + aT < 0) \rightarrow v^2 \leq 2a_n^{min}(x_c - x)\,. \tag{40}$$

With this result, i.e., $\vDash \psi_2$, it follows from Theorem 2 that $\vDash \neg\chi_2$ for the choice of ζ_2. Thus, it entails that Model 6 is bereft of Problem 1 and Problem 2, as summarised in the last row of Table 2.

5.1. Analysing other verified models: the European train control system

Table 2 presents the results obtained from applying Theorem 1 and Theorem 2 to the automated driving controller example. This section shows how the theorems can be used to analyse other verified models by using the European Train Control System (ETCS) models from Platzer and Quesel [13].

Model 7 shows the hybrid system model of ETCS (see Fig. 5 in [13]). ETCS is a standard to ensure safe operation of trains in addition to their high throughput. To formally analyse the ETCS, Platzer and Quesel [13] model it as a HP with two independent and parallel components as shown in (41). The first component $train_r$ consists of three sequentially composed HPs: *drive* (46) which models the train dynamics using a double integrator model (similar to the automated driving example), a speed supervision component *spd* (43), and a supervisory train controller, the automatic train protection unit atp_r (44). The train's acceleration $\tau.a$ is dynamically adjusted by atp_r to maintain safe operation.

The second component in (41) is the radio block controller rbc_r, which determines the train's movement permissions based on the current track situation. Trains are only allowed to operate within their current movement authority, modelled using the vector $m = (d, e, r)$. The train controller atp_r must control the train's movement in such a way that beyond position $m.e$, the train's velocity $\tau.v$ must not exceed $m.d$. In addition, the train should respect the recommended speed $m.r$ in the current track segment. As shown in (47), rbc_r either updates the movement authority by assigning new values to m or uses emergency messages to initiate immediate braking response from the train controller.

For brevity, a detailed explanation about all the components of the ETCS model is not given and the reader is encouraged to refer to Platzer and Quesel [13] for more information. However, it should be noted that components rbc_r, *spd*, atp_r, and *drive* in Model 7 are considered analogous to *env*, *aux*, *ctrl*, and *plant* in Model 1 for the analysis in this section.

Model 7: ETCS model from Platzer and Quesel [13].

$$ETCS_r \triangleq (train_r \cup rbc_r)^* \tag{41}$$

$$train_r \triangleq spd;\ atp_r;\ drive \tag{42}$$

$$spd \triangleq (?\,\tau.v \leq m.r;\ \tau.a := *;\ ?\,-b \leq \tau.a \leq A) \cup (?\,\tau.v \geq m.r;\ \tau.a := *;\ ?\,-b \leq \tau.a \leq 0) \tag{43}$$

$$atp_r \triangleq SB := \tfrac{\tau.v^2 - m.d^2}{2b} + \left(\tfrac{A}{b} + 1\right)\left(\tfrac{A}{2}\epsilon^2 + \epsilon\tau.v\right);\ atp \tag{44}$$

$$atp \triangleq \text{if } (m.e - \tau.p \leq SB \vee rbc.message = emergency) \text{ then } \tau.a := -b \text{ fi} \tag{45}$$

$$drive \triangleq t := 0;\ \left(\tau.p' = \tau.v \wedge \tau.v' = \tau.a \wedge t' = 1 \wedge \tau.v \geq 0 \wedge t \leq \epsilon\right) \tag{46}$$

$$rbc_r \triangleq (rbc.message := emergency) \cup \left(m_0 := m;\ m := *;\ ?\,m.r \geq 0 \wedge m.d \geq 0 \wedge m_0.d^2 - m.d^2 \leq 2b(m.e - m_0.e)\right) \tag{47}$$

Platzer and Quesel [13] verify (see Proposition 5 in [13]) that the train controller preserves safety with respect to the movement authority, that is, that the formula

$$\zeta_e \rightarrow [ETCS_r](\tau.p \geq m.e \rightarrow \tau.v \leq m.d)$$

is valid. ζ_e, given by

$$\zeta_e \equiv \tau.v^2 - m.d^2 \leq 2b(m.e - \tau.p) \wedge m.d \geq 0 \wedge \tau.v \geq 0 \wedge b > 0, \tag{48}$$

is used as the loop invariant to prove the safety of ETCS in [13].

To analyse whether the verified Model 7 is susceptible to Problem 1 and Problem 2, ρ (20) from Theorem 1 and ψ (28) from Theorem 2 can be used. The dL formula:

$$\rho_e \equiv \forall rbc.message.\, \forall m_0.\, \forall m.\, \forall m^+.\, \Big(\zeta_e \wedge m.e \leq m^+.e \rightarrow \langle rbc_r \rangle (m.e = m^+.e)\Big),$$

and the dL formula:

$$\psi_e \equiv \exists z.\, \exists v.\, \exists m.\, \Big(\zeta_e \wedge \langle rbc_r; spd \rangle \left(\neg \zeta_e' \wedge \langle drive \rangle \neg \zeta_e\right)\Big) \tag{49}$$

are proved in KeYmaera X [16], thereby confirming that Model 7 does not suffer from Problem 1 or Problem 2. Note that ζ_e' in (49) is a specific instantiation of the loop invariant ζ_e (48), similar to (40).

Though the results from the ETCS model are unsurprising, it bears evidence that Theorem 1 and Theorem 2 can be applied to analyse verified models other than the automated driving example considered in this paper. Furthermore, the results from the ETCS model, together with the results from the automated driving example, provide some hints on the usability and the scalability of the approach. In a nutshell, to analyse a given model, Theorems 1 and 2 have to be proven. In this regard, the theorem prover KeYmaera X provides support for both interactive and automated proving [6,15,19].

6. On control and loop invariance

Control Engineering is a branch of engineering that deals with the design and implementation of a *control system* to regulate the behaviour of a given dynamic system, the *plant*. Input values to the plant are dynamically chosen by the control system to effect changes to the output of the plant, and the task is to keep the plant's output within some tolerances despite the plant being subject to disturbances.

Within control engineering there is the notion of a *control invariant set*, a subset of the state-space such that starting within it, the system can be controlled to always stay within it. This is an important notion as it defines the boundaries of the behaviour within which a given controller can confine the controlled system. As is shown in this section, the control invariant set is directly related to the loop invariant used to prove the correctness of the control system treated in this work.

6.1. General case

Formally, the dynamics of a controlled system can be described by the general differential equation[3]:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}, \mathbf{u}),$$

which updates the state of the system from the current state $\mathbf{x}$ and the current control input value $\mathbf{u}$. Here, $f : \mathcal{X} \times \mathcal{U} \rightarrow \mathbb{R}^n$ is an arbitrary function over the n-dimensional state-space $\mathcal{X} \subseteq \mathbb{R}^n$ and the p-dimensional control input $\mathcal{U} \subseteq \mathbb{R}^p$.

A set $CIS(f, \mathcal{X}, \mathcal{U}) \subseteq \mathcal{X}$ is *control invariant* if for $\mathbf{x}(0) \in CIS(f, \mathcal{X}, \mathcal{U})$ there exists some control input $\mathbf{u}(t) \in \mathcal{U}$ such that $\mathbf{x}(t) \in CIS(f, \mathcal{X}, \mathcal{U})$, $\forall t \geq 0$. We parametrize CIS by the dynamics f and the set of control values $\mathcal{U}$ to make explicit that the control invariant set depends on these parameters; different dynamics and different control abilities lead to different control invariant sets. Using a logic approach, we represent sets as (Boolean) expressions. For instance, we express $\mathbf{x} \in CIS(f, \mathcal{X}, \mathcal{U})$ as $CIS(f, \mathcal{X}, \mathcal{U})(\mathbf{x})$.

Expressed in dL, the control invariant set can be described as the invariant ($T \geq 0$):

$$CIS(f, \mathcal{X}, \mathcal{U})(\mathbf{x}) \rightarrow \langle \mathbf{u} :=*; ?\mathcal{U}(\mathbf{u}) \rangle [\, t := 0; t' = 1, \mathbf{x}' = f(\mathbf{x}, \mathbf{u}) \,\&\, t \leq T \,] CIS(f, \mathcal{X}, \mathcal{U})(\mathbf{x}). \tag{50}$$

That is, if the current state is in the control invariant set, there can be selected some control input value such that any execution from the current state under the chosen control input will remain in the control invariant set.

The union of two control invariant sets is also a control invariant set, thus, the *maximal control invariant set* exists and is unique [11]. This set is bounded above and below by the upper and lower bounds of the control signals $\underline{u} \in \mathcal{U}$ and $\bar{u} \in \mathcal{U}$, respectively. For specific cases, analytical expressions of $CIS(f, \mathcal{X}, \mathcal{U})$ are known [11].

The proof system of KeYmaera X relies on the general loop invariant rule:

$$\text{loop}\ \frac{\Gamma \vdash \zeta, \Delta \qquad \zeta \vdash [\alpha]\zeta \qquad \zeta \vdash \phi}{\Gamma \vdash [\alpha^*]\phi, \Delta}\ .$$

However, as is well-known in the program verification community, suitable loop invariants for formal program verification are notoriously difficult to find. Also, the theory of program verification does not provide guarantees of a found loop invariant being the weakest possible one, even if that is indeed desired. Both problems can potentially be addressed by results for maximal control invariant sets, which may then help to find loop invariants useful in the verification of the types of models treated in this work.

[3] The dot is used to distinguish the mathematical notation of derivation from the dL notation that uses the prime.

For the specific case of a controlled system, the dL calculus could provide a dedicated *control invariant set rule*:

$$\text{controlInvSet}\ \frac{\Gamma \vdash \mathit{CIS}(f,\mathcal{X},\mathcal{U}),\Delta \qquad \mathit{CIS}(f,\mathcal{X},\mathcal{U}) \vdash [\mathit{discr};\mathit{plant}]\,\mathit{CIS}(f,\mathcal{X},\mathcal{U}) \qquad \mathit{CIS}(f,\mathcal{X},\mathcal{U}) \vdash \phi}{\Gamma \vdash [(\mathit{discr};\mathit{plant})^*]\phi,\Delta},$$

where *discr* is the discrete part of the control loop, in our model $\mathit{discr} := \mathit{env}; \mathit{aux}; \mathit{ctrl}$.

Importantly, the middle premise of the rule is not necessarily correct by construction, see (50), as *CIS* is agnostic towards the discrete part of the control loop. In particular, though *CIS* is aware of the maximal and minimal control inputs, it is agnostic to the *behaviour* of the controller; a specific controller may choose control signal inputs that do not keep the system within *CIS*. Indeed, we must prove that any given controller always chooses control input values that respect the invariance of *CIS*. However, the advantage of using the *controlInvSet* rule would be that $\mathit{CIS}(f,\mathcal{X},\mathcal{U})$ can be computed by some algorithm for control invariant sets (such as given by [11], see below), so that the invariant does not have to be provided by the user (via interaction or program annotation), instead it can be computed as part of the rule application[4].

A systematic exploitation of this insight is future work, but in the next section we investigate a special case relevant to the material of the previous sections of this paper.

6.2. Single input 2nd order linear system

This paper concerns single input 2nd order linear integrator systems (see (14)) of the form:

$$\begin{cases} \dot{x}_1 = u \\ \dot{x}_2 = x_1 \end{cases}$$

with the state-space $\mathcal{X} = [x_1, x_2]^T \in \mathbb{R}^2$ and the control signal $\mathcal{U} = [u] \in \mathbb{R}$. The dynamics of such a controlled linear system is typically expressed in matrix form as:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t),$$

where in this particular case:

$$A = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \text{ and } B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Thus, we have that $f := A\mathbf{x}(t) + B\mathbf{u}(t)$.

Both $\mathcal{X}$ and $\mathcal{U}$ are subject to rectangular constraints:

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^2 \mid \underline{x}_i \le x_i \le \bar{x}_i,\ i = 1,2\}, \text{ and } \mathcal{U} = \{u \in \mathbb{R} \mid \underline{u} \le u \le \bar{u}\},$$

where $\underline{x}_i$ and $\underline{u}$, and $\bar{x}_i$ and $\bar{u}$, are minimum and maximum values, respectively, of x_i and u, and where it is assumed that $\underline{u} < 0 < \bar{u}$, to exclude uncontrollable systems.

Given such a 2nd order integrator system with rectangular constraints, Doeser et al. [11] give an analytical characterisation of the maximal control invariant set contained in $\mathcal{X}$ for an upper state bound $x_n \le \bar{x}_n$ and a lower input bound $\underline{u} \le u$ as:

$$\begin{cases} \bar{x}_2 - x_2 + \frac{x_1^2}{2\underline{u}} & \text{if } x_1 \ge 0 \\ \bar{x}_2 - x_2 & \text{otherwise,} \end{cases}$$

where the condition on x_1 guarantees that $t \in \mathbb{R}_{>0}$.

In the specific case of this paper (see Model 2), the control signal u is the acceleration, so that x_1 is the velocity v, and x_2 the position x. Thus:

$$\begin{cases} \dot{v} = a \\ \dot{x} = v, \end{cases}$$

with the acceleration bounded $\underline{u} = -a_n^{min} \le a \le a_n^{max} = \bar{u}$, which fulfils the constraint $\underline{u} < 0 < \bar{u}$. Also, both x and v are bounded, $\underline{x}_2 = 0 \le x \le x_c = \bar{x}_2$, and $\underline{x}_1 = 0 \le v \le v^{max} = \bar{x}_1$, so there are rectangular state constraints. Thus, since $v = x_1 \ge 0$, the maximal control invariant set can be written as:

$$C_2 = \left\{ \langle x, v \rangle \in \mathbb{R}^2 \mid x_c - x - \frac{v^2}{2a_n^{min}} \ge 0 \right\}.$$

The characterisation of C_2 can be re-arranged so that C_2 can be written as:

$$C_2 = \left\{ \langle x, v \rangle \in \mathbb{R}^2 \mid v^2 \le 2a_n^{min}(x_c - x) \right\},$$

[4] The presented rule is idealised in so far as it is not pattern-matching friendly.

which is the exact same expression as the loop invariant ζ_2 (38), so that C_2 can be given as:

$$C_2 = \{\langle x, v\rangle \in \mathbb{R}^2 \mid \zeta_2(x, v)\}.$$

Thus, the loop invariant necessary to correctly prove the considered type of hybrid systems correct can be – and in this example is – identical to the characterisation of the maximal control invariant set. In fact, [11] gives closed-form expressions for maximal control invariant sets up to 4th order dynamics (with single inputs), which can serve as candidates for loop invariants for the verification of such systems.

7. Related work

The European Train Controller System (ETCS) [13], Model 7, is very similar to the other models of this paper. Though not explicitly stated, the modelling 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. However, as shown in Section 5.1, the approach presented in this paper can be used to analyse the ETCS model to increase confidence in the design.

An alternative to guarantee CPS correctness is *runtime validation* [20], 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 5.

The issue in Model 3 is not unique to dL; the issue manifests itself similarly in reactive synthesis [21,22]. 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 fulfil the consequent, an exploiting *ctrl* can invalidate the premise to fulfil the implication. However, Bloem et al. [21] conclude that none of the existing approaches completely solve the problem and emphasize the need for further research.

Essentially, the loop step (ii) $\zeta \vdash [\,\alpha\,]\zeta$ for (7) of Model 1 becomes $\zeta \wedge env \wedge aux \wedge ctrl \wedge plant \rightarrow \zeta$ during the proof calculus. Evidently, whenever *env* is false, the entire formula is true. So, if in one execution *ctrl* can force the hybrid system into a state in which ζ is true but *env* is false, then the loop step becomes trivially true in the next execution.

In reactive synthesis, the formula that must be fulfilled is abstractly stated as $A \rightarrow G$, where A represents the formula that the environment is assumed to fulfil, whereas G is the formula which the controller must guarantee to fulfil. As can be seen, the formula is trivially true if the environment does not fulfil its assumptions. Like for the problem with the cheating controller in dL, the reactive synthesis controller may also exploit the environment and force the system into a state from which the environment cannot fulfil its assumptions. See for instance Majumdar et al. [22] for a concrete example.

One way to solve the reactive synthesis problem is to convert the formula and the interactions between the environment and the controller into a two-player game, which is constructed such that if the controller wins a play the implication is true, and if the environment wins a play the implication is false. The game consists of states, where each state belongs to either the controller or the environment. In a play, the controller and the environment choose the next state depending on who owns the current state, and a player's choice of next state from each state is called its strategy. Whether a play is won by the controller or the environment is determined based on which states are visited during the play. A state is winning for a player if there exists a strategy such that, no matter what strategy the other player is using, the first player wins all plays starting in that state that are played according to that strategy. The set of winning states of a player is its winning region. The other states are the winning region for the other player. Thus, to be guaranteed to win, a player's strategy must always choose the next state from its winning region. With such a strategy for the first player, the other player will never have the choice to leave the first player's winning region.

Any state from which the controller can force the game into a dead end or into a set of states where the environment assumptions are not fulfilled is part of the winning region of the controller. The consequence becomes that the controller can "cheat" and instead of winning by fulfilling its own guarantees, it prevents the environment from fulfilling its assumptions.

The situation can be viewed in a similar way in the loop step. $\zeta \rightarrow [env; ctrl; plant](\zeta)$ essentially means that, if starting in a state set defined by ζ, then *env* and *ctrl* must choose actions such that the same state set defined by ζ is reached after *plant* has been applied. If *env* is viewed as an adversarial and *ctrl* as the entity for which a strategy shall be found, then the loop step requires that, for the controller to win the game and fulfil the guarantee, if starting in a state in which ζ holds, then, no matter what *env* chooses to do, *ctrl* must be able to choose a next state in which ζ holds. Thus, it is possible to view ζ as a definition of a winning region for *ctrl*. If the loop step formula is valid, then the model of *ctrl* is a winning strategy with respect to *env* and ζ.

The problem in Section 4.1 occurs because the "winning region" for *ctrl* includes states that lead to dead ends for *env*. Those dead ends arise because *env* cannot fulfil its assumptions in the entire winning region. So, similar to the reactive synthesis case, *ctrl* can play by a strategy that forces *env* into dead ends.

The solution put forth in Section 4.1 restricts ζ and removes states from the winning region of *ctrl*; states from which it is impossible for *env* to fulfil its assumptions are removed, thus preventing *ctrl* to completely restrict *env*. In other words, the set of states in which the assumptions are fulfilled is always reachable. This is very similar to the statement "Satisfaction [...] requires checking whether the set [of states fulfilling the assumptions of the environment] remains reachable from any reachable state in the game graph realising [all plays induced by the controller's strategy]" [22]. Although reactive synthesis and dL are very different, the solution to a vacuous *env* in Theorem 1 and the solution to a falsified assumption by Majumdar et al. [22] seem, on a high level, to solve the same type of problem with the same general technique. That is, preventing falsification of the assumptions by restricting the controller's winning region to states from which it is possible to eventually reach states where the assumptions are fulfilled.

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. [23] 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 modelling 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 [24] 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.

8. Conclusions

This paper shows that modelling errors present a risk of unsound conclusion from provably safe, but erroneous models, which is a big problem not only for 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 modelling 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. Moreover, it is shown that this loop invariant coincides with the characterisation of the *maximal control invariant set* as defined for control engineering systems, which is the largest set of states such that starting within it the system can be controlled to stay within it. This is an interesting observation that suggests that to find suitable loop invariants it might be useful to look into methods from the control engineering community to find (maximal) control invariant sets.

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 modelling errors discussed in this paper.

A natural extension of this work will be to investigate also other kinds of modelling 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.

The starting point for this work is the insight that by erroneous modelling it is easy to obtain vacuous safety proofs for controlled systems without noticing. The overall problem of drawing wrong conclusions from proofs in the presence of modelling errors is well known, and typically addressed on the meta level, in an informal manner. The main contribution of this work is to show how two specific classes of modelling errors *can be excluded using formal verification technology* itself. In general, we are convinced that more attention is needed to systematically address modelling and specification errors, *with the help of tools, and with a high level of confidence*.

CRediT authorship contribution statement

Yuvaraj Selvaraj: Conceptualization, Formal analysis, Software, Writing – original draft, Writing – review & editing. **Jonas Krook:** Conceptualization, Formal analysis, Software, Writing – original draft, Writing – review & editing. **Wolfgang Ahrendt:** Conceptualization, Formal analysis, Supervision, Writing – original draft, Writing – review & editing. **Martin Fabian:** Conceptualization, Formal analysis, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E. Lee, Cyber-physical systems - are computing foundations adequate?, Position Paper for NSF Workshop on Cyber-Physical Systems: Research Motivation, Techniques and Roadmap, vol. 2, 2006.
[2] J. Michael, D. Drusinsky, D. Wijesekera, Formal verification of cyberphysical systems, Computer 54 (2021) 15–24, https://doi.org/10.1109/MC.2021.3055883.
[3] Y. Selvaraj, W. Ahrendt, M. Fabian, Formal development of safe automated driving using differential dynamic logic, IEEE Trans. Intell. Veh. 8 (2022) 988–1000, https://doi.org/10.1109/TIV.2022.3204574.
[4] R. Alur, T.A. Henzinger, E.D. Sontag (Eds.), Hybrid Systems III – Verification and Control, Lecture Notes in Computer Science, vol. 1066, Springer, 1996.
[5] R. Alur, Formal verification of hybrid systems, in: 2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT), 2011, pp. 273–278.
[6] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, A. Platzer, KeYmaera X: an axiomatic tactical theorem prover for hybrid systems, in: International Conference on Automated Deduction, Springer, 2015, pp. 527–538.
[7] A. Benveniste, Compositional and uniform modelling of hybrid systems, in: R. Alur, T.A. Henzinger, E.D. Sontag (Eds.), Hybrid Systems III, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 41–51.
[8] P. Koopman, A. Kane, J. Black, Credible autonomy safety argumentation, in: 27th Safety-Critical Systems Symposium, 2019, pp. 34–50.
[9] A. Platzer, Logical Foundations of Cyber-Physical Systems, vol. 662, Springer, 2018.
[10] A. Platzer, Logics of dynamical systems, in: 27th Annual IEEE Symposium on Logic in Computer Science, IEEE, 2012, pp. 13–24.

[11] L. Doeser, P. Nilsson, A.D. Ames, R.M. Murray, Invariant sets for integrators and quadrotor obstacle avoidance, in: 2020 American Control Conference (ACC), 2020, pp. 3814–3821.
[12] Y. Selvaraj, J. Krook, W. Ahrendt, M. Fabian, On how to not prove faulty controllers safe in differential dynamic logic, in: A. Riesco, M. Zhang (Eds.), Formal Methods and Software Engineering, Springer International Publishing, Cham, 2022, pp. 281–297.
[13] A. Platzer, J.-D. Quesel, European train control system: a case study in formal verification, in: K. Breitman, A. Cavalcanti (Eds.), Formal Methods and Software Engineering, Springer, 2009, pp. 246–265.
[14] O. Kupferman, P. Madhusudan, P.S. Thiagarajan, M.Y. Vardi, Open systems in reactive environments: control and synthesis, in: CONCUR 2000 — Concurrency Theory, in: Lecture Notes in Computer Science, vol. 1877, Springer, Berlin, Heidelberg, 2000, pp. 92–107.
[15] S. Mitsch, A. Platzer, A retrospective on developing hybrid system provers in the keymaera family: a tale of three provers, in: Deductive Software Verification: Future Perspectives: Reflections on the Occasion of 20 Years of KeY, Springer, 2020, pp. 21–64.
[16] Y. Selvaraj, J. Krook model-pitfalls-dl, https://doi.org/10.5281/zenodo.6821673, 2022.
[17] J.-D. Quesel, S. Mitsch, S. Loos, N. Aréchiga, A. Platzer, How to model and prove hybrid systems with KeYmaera: a tutorial on safety, Int. J. Softw. Tools Technol. Transf. 18 (2016), https://doi.org/10.1007/s10009-015-0367-0.
[18] Y. Selvaraj, J. Krook, model-pitfalls-dl, https://github.com/yuvrajselvam/model-pitfalls-dl, 2023.
[19] S. Mitsch, Implicit and explicit proof management in KeYmaera X, in: Proceedings of the 6th Workshop on Formal Integrated Development Environment, in: EPTCS, vol. 338, 2021.
[20] S. Mitsch, A. Platzer, ModelPlex: verified runtime validation of verified cyber-physical system models, Form. Methods Syst. Des. 49 (2016), https://doi.org/10.1007/s10703-016-0241-z.
[21] R. Bloem, R. Ehlers, S. Jacobs, R. Könighofer, How to handle assumptions in synthesis, in: K. Chatterjee, R. Ehlers, S. Jha (Eds.), Proceedings 3rd Workshop on Synthesis, SYNT, in: EPTCS, vol. 157, 2014, pp. 34–50.
[22] R. Majumdar, N. Piterman, A.-K. Schmuck, Environmentally-friendly GR(1) synthesis, in: Tools and Algorithms for the Construction and Analysis of Systems, 2019, pp. 229–246.
[23] A. Müller, S. Mitsch, W. Retschitzegger, W. Schwinger, A. Platzer, Tactical contract composition for hybrid system component verification, Int. J. Softw. Tools Technol. Transf. 20 (2018) 615–643, https://doi.org/10.1007/s10009-018-0502-9.
[24] C.A. Furia, B. Meyer, Inferring loop invariants using postconditions, in: A. Blass, N. Dershowitz, W. Reisig (Eds.), Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday, Springer, 2010, pp. 277–300.