Thesis Proposal
Verified Control Envelope Synthesis for Hybrid Systems

Aditi Kabra

19^{th} September 2024

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
André Platzer, Co-chair
Stefan Mitsch, Co-chair
Armando Solar-Lezama
Eunsuk Kang

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

Many cyber-physical systems, such as trains, planes, and self-driving cars, are safety-critical but difficult to reason about. Formal verification can provide strong safety guarantees, but most industrial controllers are too complex to formally verify. *Safe control envelopes* characterize families of safe controllers and are used to monitor [32] untrusted controllers on verifiable *abstractions* of control systems that isolate the parts relevant to safety without the full complexity of a specific control implementation, at runtime. They can put complex controllers, even when machine learning based [18], within the reach of formal guarantees. But correct control envelopes are still hard to design because the control engineer needs to identify correct control conditions that tell the controller what to do right now to stay safe at all times in the future by anticipating the behavior of the system over complex dynamics and an uncountably infinite state space (the subject of the deep field of control theory). This thesis proposes to provide synthesis techniques to automatically synthesize provably correct control conditions, greatly reducing the manual effort required for control envelope design. It aims to scale synthesis to complex real-world problems with realistic physical dynamics.

The input of the synthesis tool is a sketch of the control envelope in a hybrid system showing what kind of control behavior is physically possible. The tool fills in the blanks of the sketch by synthesizing control conditions using hybrid system game theory. The output is a provably correct symbolic control envelope. Existing controller synthesis techniques do not solve control *envelope* synthesis because control envelopes have the higher-order constraint of permitting as many valid control solutions as possible. Completed work provides the algorithm CESAR (Control Envelope Synthesis via Angelic Refinement) [25], which solves a class of problems where a set of systematic game refinements allows automatic control envelope synthesis. Proposed work generalizes synthesis to a broad class of systems (characterized by admitting a natural representation in differential game logic) and develops a system that allows users to provide the human intuition based insights that, together with automated reasoning, can complete the control envelope synthesis process in more complex cases.

We aim to scale synthesis to control envelopes complex enough to be used in real systems. As a first step, completed work [24] manually develops a case study for the synthesis system to target: the first verified train control envelope [23] that accounts for all the forces in the realistic Federal Railroad Administration Train Kinematics Model [7]. The case study identifies generalizable techniques to deal with the challenges this system poses that appear in other real systems: cyclically interdependent system variables, underspecified dynamical influences whose exact values are unknown at proof time, and differential equation dynamics that lie outside the decidable fragment of real arithmetic.

Contents

1 Introduction

In cyber-physical systems (CPS) like trains, planes, and self-driving cars, discrete software interacts with continuous physics. Many of these systems are safety-critical, since software error can have tragic or expensive consequences. Formal verification mathematically proves that a system adheres to its safety requirements, and can provide high safety assurance. However, proving the correctness of the code is difficult, with proof complexity increasing drastically with code and system complexity. Most industrial systems are too complex to directly formally verify. *Control envelopes* provide a solution. They model simpler abstractions of the control system that focus only on safety-critical aspects. These abstractions can be verified and then checked against the full system at runtime [32].

A safe control envelope is a nondeterministic program whose every execution is safe. It is used to monitor the actual controller at runtime: if observed controller or environment behavior is not within the envelope, a safe fallback is enforced [31]. Safe control envelopes allow complex controllers, even those based on machine learning, to be used in safety-critical systems while still providing formal safety guarantees [17]. However, safe control envelopes are still very difficult to design and verify. The process requires the engineer to anticipate the influence of complicated system dynamics and the interactions of possible future controller decisions to design *control conditions* that say how the controller should act now to ensure safety at all times in the future, regardless of how environment and controller nondeterminisms play out. To design these conditions, engineers need to exercise creativity and domain expertise. Proving that the designed conditions are correct then requires careful mathematical reasoning over many engineering hours, likely resulting in the discovery that the control envelope cannot be proved correct because of a subtle bug, so that the control conditions need to be fixed and re-proved again.

This thesis aims to make it easier to design control envelopes by automating the most difficult part of the process: constructing the provably correct control conditions. We develop techniques to synthesize control conditions given a sketch of the control envelope defining system dynamics, what control decisions are physically possible, and the safety contract. Control conditions should be as permissive as possible, so that the resulting control envelope only interferes with control actions when they are genuinely unsafe. Naïvely applying existing controller synthesis techniques is insufficient for solving the control *envelope* synthesis problem, which poses the higher-order challenge of capturing *as many* valid control solutions as possible. We solve this challenge by implicitly characterizing an optimal solution to the control envelope synthesis problem using hybrid games, and then solve these games to obtain an explicit solution via symbolic execution aided by *game refinement* (transforming the game to be easier to execute symbolically while still preserving some properties).

Completed work provides the CESAR (Control Envelope Synthesis via Angelic Refinements) algorithm [25], which solves the class of problems with *time-triggered control* (where the controller repeatedly chooses an action with some maximum time latency) and *action permanence* (a hybrid analog to idempotence). For these problems, we identify a systematic game refinement approach to aid symbolic execution when it fails at loops (which can execute an unbounded number of times, failing symbolic execution). Proposed work generalizes synthesis to the class of problems where an implicit game characterization of the optimal solutions exists. In this general case, explicit solutions are again extracted with symbolic execution, but now aided by heuris-

tics that guess solutions when symbolic execution fails. The guesses are checked to recover correctness.

An important objective of our work is to be able to synthesize control envelopes complex enough to apply to real-world systems. Designing symbolic control envelopes with high fidelity to real-world dynamics even manually is so challenging that case studies from the literature still usually make simplifications while modeling the environment and controller logically [30, 33, 34, 35, 42] that leave them a few steps away from being immediately applicable as run-time monitors. So we first create as a target for the synthesis tool a verified control envelope that solves the practically important problem of train control [24] accounting for all the forces in the realistic Federal Railroad Administration Train Kinematics Model [7]. This case study poses challenges common in other safety-critical embedded systems – complex dynamics with transcendental arithmetic, competing forces with subtle interaction, and effects whose exact magnitude is unknown at proof time – and makes us seek generalizable solutions. Proposed work has the goal of scaling synthesis to the point where it can automatically generate the control conditions and proof of this case study.

In summary, this thesis proposes to design techniques to perform verified control envelope synthesis for hybrid systems. The main idea is to characterize optimal synthesis solutions using hybrid system game theory, and then extract explicit solutions using symbolic execution aided by refinements and heuristics. We plan to use LLMs to scale to greater complexity in combination with verification to maintain soundness and aim to synthesize control envelopes that are complex enough to be useful in real-world systems.

2 Completed Work

Completed work develops CESAR, an algorithm to synthesize symbolic control envelopes for a subset of time-triggered controllers with the "action permanence" property, which intuitively means that there is some control action such that repeatedly choosing it results in a period of smooth dynamical behavior (e.g., a train continuously choosing to brake). While a full explanation is elsewhere [25], here we provide an intuitive overview of CESAR by example.

Completed work also develops as a target for synthesis a verified train control envelope [24] that accounts for all the forces in the realistic Federal Railroad Administration Train Kinematics Model [7], and deals in generalizable ways with the challenges that arise with realistic modeling of forces. We provide an overview of this work as well. Much of the text in this section is based on the corresponding completed papers [24, 25].

2.1 CESAR by Example

2.1.1 Background

This section briefly introduces dL and dGL. We use hybrid games written in differential game logic (dGL, [39]) to represent solutions to the synthesis problem. Hybrid games are two-player zero-sum games with no draws that are played on a hybrid system with differential equations. Players take turns, and in their turn can choose to act arbitrarily within the game rules. At the end

of the game, one player wins and the other loses. The players are classically called Angel and Demon. *Hybrid systems*, in contrast, have no agents, only a nondeterministic controller running in a nondeterministic environment. The synthesis problem consists of filling in holes in a hybrid system. Thus, expressing solutions for hybrid *system* synthesis with hybrid *games* is one of the insights of this work.

An example of a game is $(v := 1 \cap v := -1)\,;\,\{x' = v\}$. In this game, first Demon chooses between setting velocity v to 1, or to -1. Then, Angel evolves position x as $x' = v$ for a duration of her choice. Differential game logic uses modalities to set win conditions for the players. For example, in the formula $[(v := 1 \cap v := -1)\,;\,\{x' = v\}]x \neq 0$, Demon wins the game when $x \neq 0$ at the end of the game and Angel wins otherwise. The overall formula represents the set of states from which Demon can win the game, which is $x \neq 0$ because when $x < 0$, Demon has the *winning strategy* to pick $v := -1$, so no matter how long Angel evolves $x' = v$, x remains negative. Likewise, when $x > 0$, Demon can pick $v := 1$. However, when $x = 0$, Angel has a winning strategy: to evolve $x' = v$ for zero time, so that x remains zero regardless of Demon's choice.

We summarize dGL's program notation (Table 1). See [39] for a full explanation. Assignment $x := \theta$ instantly changes the value of variable x to the value of θ. Challenge $?\psi$ continues the game if ψ is satisfied in the current state, otherwise Angel loses immediately. In continuous evolution $x' = \theta \;\&\; \psi$ Angel follows the differential equation $x' = \theta$ for some duration of her choice, but loses immediately on violating ψ at any time. Sequential game $\alpha;\beta$ first plays α and when it terminates without a player having lost, continues with β. Choice $\alpha \cup \beta$ lets Angel choose whether to play α or β. For repetition α^*, Angel repeats α some number of times, choosing to continue or terminate after each round. The dual game α^d switches the roles of players. For example, in the game $?\psi^d$, Demon passes the challenge if the current state satisfies ψ, and otherwise loses immediately.

In games restricted to the structures listed above but without α^d, all choices are resolved by Angel alone with no adversary, and hybrid games coincide with hybrid systems in differential

Table 1: Hybrid game operators for two-player hybrid systems

Game	**Effect**
$x := \theta$	assign value of term θ to variable x
$?\psi$	Angel passes challenge if formula ψ holds in current state, else loses immediately
$\big(x_1' = \theta_1, \ldots,$ $x_n' = \theta_n \;\&\; \psi\big)$	Angel evolves x_i along differential equation system $x_i' = \theta_i$ for choice of duration ≥ 0, loses immediately when violating ψ
$\alpha;\beta$	sequential game, first play hybrid game α, then hybrid game β
$\alpha \cup \beta$	Angel chooses to follow either hybrid game α or β
α^*	Angel repeats hybrid game α, choosing to stop or go after each α
α^d	dual game switches player roles between Angel and Demon
$\alpha \cap \beta$	demonic choice $(\alpha^d \cup \beta^d)^d$ gives choice between α and β to Demon
$\alpha^\times$	demonic repetition $((\alpha^d)^*)^d$ gives control of repetition to Demon

dynamic logic (dL) [39]. We will use this restriction to specify the synthesis *question*, the sketch that specifies the shape and safety properties of control envelopes. But to characterize the *solution* that fills in the blanks of the control envelope sketch, we use games where both Angel and Demon play. The notation we use includes demonic choice $\alpha \cap \beta$, which lets Demon choose whether to run α or β. Demonic repetition $\alpha^\times$ lets Demon choose whether to repeat α choosing whether to stop or go at the end of every run.

In order to express properties about hybrid games, differential game logic formulas refer to the existence of winning strategies for objectives of the games (e.g., a controller has a winning strategy to achieve collision avoidance despite an adversarial environment). The set of dGL formulas is generated by the following grammar (where $\sim \in \{<, \leq, =, \geq, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals, x is a variable, α is a hybrid game):

$$\phi := \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

Comparisons of arithmetic expressions, Boolean connectives, and quantifiers over the reals are as usual. The modal formula $\langle\alpha\rangle\phi$ expresses that player Angel has a winning strategy to reach a state satisfying ϕ in hybrid game α. Modal formula $[\alpha]\phi$ expresses the same for Demon. The fragment without modalities is first-order real arithmetic. Its fragment without quantifiers is called *propositional arithmetic* $\mathcal{P}_{\mathbb{R}}$. Details on the semantics of dGL can be found in [39].

States are functions assigning a real number to each variable. For instance, $\phi \rightarrow [\alpha]\psi$ is valid iff, from all initial states satisfying ϕ, Demon has a winning strategy in game α to achieve ψ.

2.1.2 Control Envelopes

We make the idea of a control safety envelope concrete by example. In order to separate safety-critical aspects from other system goals during control design, control envelopes abstractly describe the safe choices of a controller, deliberately underspecifying when and how to exactly execute certain actions. Control envelopes focus on describing in which regions it is safe to take actions. For example, Model 1 designs a train control envelope [42] that must stop by the train by the *end of movement authority* e located somewhere ahead, as assigned by the train network scheduler. Past e, there may be obstacles or other trains. The train's control choices are to accelerate or brake as it moves along the track. The goal of CESAR is to synthesize the framed formulas in the model, that are initially blank.

Line 8 describes the *safety property* that is to be enforced at all times: the train driving at position p with velocity v must not go past position e. Line 1 lists *modeling assumptions*: the train is capable of both acceleration ($A{>}0$) and deceleration ($B{>}0$), the controller latency is positive ($T{>}0$) and the train cannot move backwards as a product of braking (this last fact is also reflected by having $v \geq 0$ as a domain constraint for the plant on Line 6). These assumptions are fundamentally about the physics of the problem being considered. In contrast, Line 3 features a *controllability assumption* that can be derived from careful analysis. Here, this synthesized assumption says that the train cannot start so close to e that it won't stop in time even if it starts braking immediately. Line 4 and Line 5 describe a train controller with two actions: accelerating ($a := A$) and braking ($a := -B$). Each action is guarded by a synthesized formula, called an *action guard* that indicates when it is safe to use. Angel has control over which action runs, and adversarially plays with the objective of violating safety conditions. But Angel's options

Model 1 The train ETCS model (slightly modified from [42]). Framed formulas are initially blank and are automatically synthesized by our tool as indicated.

assum	1	$A > 0 \land B > 0 \land T > 0 \land v \geq 0 \land$
ctrlable	2	$\boxed{e - p > v^2/2B} \rightarrow [\{$
ctrl	3	$\quad (\quad (?\boxed{e - p > vT + AT^2/2 + (v + AT)^2/2B}; a := A)$
	4	$\quad \cup\ (?\boxed{\text{true}}; a := -B) \quad);$
plant	5	$\quad (t := 0;\ \{p' = v, v' = a, t' = 1\ \&\ t \leq T \land v \geq 0\})$
safe	6	$\}^*](e - p > 0)$

are limited to only safe ones because of the synthesized action guards, ensuring that Demon still wins and the overall formula is valid. In this case, braking is always safe whereas acceleration can only be allowed when the distance to end position e is sufficiently large. Finally, the plant on Line 6 uses differential equations to describe the train's kinematics. A timer variable t is used to ensure that no two consecutive runs of the controller are separated by more than time T. Thus, this controller is *time-triggered*.

2.1.3 Overview of CESAR

For Model 1, CESAR first identifies the optimal solution for the blank of Line 3. Intuitively, this blank should identify a *controllable invariant*, which denotes a set of states where a controller with choice between acceleration and braking has some strategy (to be enforced by the conditions of Line 4 and Line 5) that guarantees safe control forever. Such states can be characterized by the following dGL formula where Demon, as a proxy for the controller, decides whether to accelerate or brake: $[((a := A \cap a := -B);\ \mathsf{plant})^*]\mathsf{safe}$ where plant and safe are from Model 1. When this formula is true, Demon, who decides when to brake to maintain the safety contract, has a winning strategy that the controller can mimic. When it is false, Demon, a perfect player striving to maintain safety, has no winning strategy, so a controller has no guaranteed way to stay safe either. We define a solution in dGL for any given CESAR input and prove that it is optimal after first defining an ordering on solutions where more permissive solutions, allowing more actions more often are better.

This dGL formula provides an *implicit* characterization of the optimal controllable invariant from which we derive an explicit formula in $\mathcal{P}_{\mathbb{R}}$ to fill the blank with using symbolic execution. Symbolic execution solves a game following the axioms of dGL to produce an equivalent $\mathcal{P}_{\mathbb{R}}$ formula. However, our dGL formula contains a loop, for which symbolic execution will not terminate in finite time. To reason about the loop, we *refine* the game, modifying it so that it is easier to symbolically execute, but still at least as hard for Demon to win so that the controllable invariant that it generates remains sound. In this example, the required game transformation first restricts Demon's options to braking. Then, it eliminates the loop using the observation that the repeated hybrid iterations $(a := -B; \mathsf{plant})^*$ behave the same as just following the continuous dynamics of braking for unbounded time. It replaces the original game with $a := -B;\ t := 0;\ \{p' = v, v' = a\ \&\ \land v \geq 0\}$, which is loop-free and easily symbolically executed.

Symbolically executing this game to reach safety condition safe yields controllable invariant $e - p > \frac{v^2}{2B}$ to fill the blank of Line 3.

Intuitively, this refinement captures situations where the controller stays safe forever by picking a single control action (braking). It generates the optimal solution for this example because braking forever is the dominant strategy: given any state, if braking forever does not keep the train safe, then certainly no other strategy will. However, there are other problems where the dominant control strategy requires the controller to strategically switch between actions, and this refinement misses some controllable invariant states. So we introduce a new refinement: bounded game unrolling via a recurrence. A solution generated by unrolling n times captures states where the controller can stay safe by switching control actions up to n times.

Having synthesized the controllable invariant, CESAR fills the action guards (Line 4 and Line 5). An action should be permissible when running it for one iteration maintains the controllable invariant. For example, acceleration is safe to execute exactly when $[a := A; \mathsf{plant}]e - p > \frac{v^2}{2B}$. We symbolically execute this game to synthesize the formula that fills the guard of Line 4. When given a problem with solvable ODEs whose solutions are polynomial, CESAR is guaranteed to generate a solution in finite time that sound and optimal w.r.t. all k-switching fallback strategies with permanent actions where k is the number of bounded unrolling recurrences used. When ODEs are not solvable, we make approximations using continuous invariants. We design a test that can check whether the result produced is still optimal.

To evaluate CESAR, we curate a benchmark suite with diverse optimal control strategies. Some benchmarks have non-solvable dynamics, while others require a sequence of clever control actions to reach an optimal solution. Some have *state-dependent fallbacks* where the current state of the system determines which action is "safer", and some are drawn from the literature. We implement CESAR in Scala and evaluate it on the benchmarks. Despite a variety of different control challenges, CESAR is able to synthesize safe and in some cases also optimal safe control envelopes within a few minutes.

CESAR can fill in holes $\llcorner\lrcorner$ in a problem of the following shape:

$$\mathsf{prob} \equiv \mathsf{assum} \wedge \llcorner\lrcorner \rightarrow [\big((\cup_i (?\llcorner\lrcorner_i ; \mathsf{act}_i)) ; \mathsf{plant}\big)^*]\mathsf{safe}. \tag{1}$$

Here, the control envelope consists of a nondeterministic choice between a finite number of guarded actions. Each action act_i is guarded by a condition $\llcorner\lrcorner_i$ to be determined in a way that ensures safety within a controllable invariant [4, 21] $\llcorner\lrcorner$ to be synthesized also. In addition, we make the following assumptions:

1. Components assum, safe and domain are propositional arithmetic formulas.
2. Timer variable t is fresh (does not occur except where shown in template).
3. Programs act_i are discrete dL programs that can involve choices, assignments and tests with propositional arithmetic. Variables assigned by act_i must not appear in safe. In addition, act_i must terminate in the sense that $\vDash \langle \mathsf{act}_i \rangle \mathsf{true}$.
4. The modeling assumptions assum are invariant in the sense that

   $$\vDash \mathsf{assum} \rightarrow [(\cup_i \mathsf{act}_i) ; \mathsf{plant}]\mathsf{assum}.$$

   This holds trivially for assumptions about constant parameters such as $A > 0$ in Model 1 and this ensures that the controller can always rely on them being true.

Proposed work moves past this template to synthesize for a much broader variety of sketches.

2.2 Case Study: Train Control

Train control envelopes decide when to enforce braking to prevent movement authority violation and collisions. They must account for all the competing influences that govern train motion. Uphill slopes decrease velocity, for example, which decreases resistance, which permits a more rapid increase in velocity, slope and curve effect, all while the train's brake force builds gradually until saturation as air pressure propagates along brake pipes. These complex interactions make it hard to design an efficient train control envelope, and even harder to ensure it is always safe. Completed work [24] designs and verifies train controllers for the Federal Railroad Administration (FRA) freight train kinematics model [6, 7] (henceforth FRA model), contributing generalizable verified control envelope design techniques. Ultimately, our synthesis tool should automatically synthesize for this case study.

Previous case studies of formally verified train motion [34, 42, 49] do not account for at least two effects amongst track grade, track curvature, resistance, and air brake propagation time, rendering their results inapplicable to most real-world scenarios. We surmount the challenges of verification against the full dynamics of the FRA model, in which these effects interact subtly with each other. Our verification results are significant, because these parameters influence the motion of the train in safety-critical and/or performance-critical ways. Neglecting track slope profile and the gradual propagation of air pressure braking, in particular, can render otherwise verifiably safe train controllers unsafe, since their influence may diminish the train's ability to decelerate, causing collisions. Our verification is valid for realistic FRA models [6, 7].

Before verifying envelope safety, we first design the control envelopes, balancing efficiency with provable safety. Conservative control envelopes are mathematically more simplistic, and easier to design and verify, but make railway operations inefficient, violating performance objectives. We start by presenting a conservative safe control envelope and then iteratively make it more efficient by exploiting characteristics of the physical train dynamics for better but safe control. Train control envelopes are assessed relative to a (changing) destination stopping point—called *end of movement authority*: overshoot of the end of movement authority is a safety violation, because that risks collision with other trains; efficiency is measured in terms of end of movement authority undershoot. We prove absence of end of movement authority overshoot when using our controllers in the FRA model by verification and demonstrate efficiency by simulation.

To remain as general and widely applicable as possible, our envelopes are written with symbolic parameters. Verification follows a two-stage process: we first prove symbolic mathematical models of train control, and then obtain proofs of the actual physical models of train control by *uniform substitution* [38] to replace the abstract function symbols of the mathematical models with physical terms specific to the FRA model or even specific railroads. Our proof is written in differential dynamic logic [37, 38], and performed using the hybrid systems theorem prover KeYmaera X [19]. We compare the efficiency permitted by our envelope with concrete control algorithms [6] for a number of train consists (arrangement of locomotives and cars) and scenarios [6], and illustrate their behavior in simulation. The proved models are permissive and only interfere in train control operation when acting otherwise risks movement authority violation.

2.2.1 Generalizable Challenges

We briefly discuss the generalizable challenges encountered in the FRA case study, first providing the background necessary to understanding them.

Background Eq. (2) shows the mathematical abstraction of the FRA train kinematic model that we construct and use [24]. It is an ODE in time. The rate of change of position is velocity, and the rate of change of velocity is acceleration. The variables and constants involved, along with their signs, when relevant, are (i) Train position p, (ii) velocity v, (iii) velocity and position-independent component of acceleration a_l ranging from immediate braking ability $-b_{\text{max}} < 0$ to maximum train engine acceleration $a_{\text{max}} > 0$, (iv) acceleration due to air brakes a_a in range $a_{b\text{max}} < 0$ to 0, (v) rate of change m_b of air brake acceleration, which is $m_p < 0$ when brakes are ramping up and 0 otherwise, (vi) map a_s from position to acceleration due to grade, (vii) map a_c from position to acceleration due to curvature, and (viii) velocity-dependent resistance a_r. In the chosen sign convention, resistive acceleration is negative.

$$\begin{aligned} &p' = v, v' = a_l + a_a + a_s(p) + a_r(v) + a_c(p), a_b' = m_b \\ &\text{with } a_l \in [-b_{\text{max}}, a_{\text{max}}], a_a = \max(a_b, a_{b\text{max}}), m_b \in \{0, m_p\} \end{aligned} \tag{2}$$

The Davis equation resistance $a_r(v)$ has the shape $a_r = a_1 v + a_2 v^2$ when a_1 summarizes the linear coefficient of velocity, and a_2 summarizes the quadratic coefficient. Grade and curvature are represented by unspecified but bounded functions a_s and a_c that map train positions to a numeric value for acceleration due to slope and average curvature, respectively. The quantity a_l summarizes locomotive tractive effort ($a_l \geq 0$) and train deceleration ($a_l < 0$) as commanded by the train controller, with adjustment for the velocity-independent resistance.

In order to decide when free driving (as opposed to braking) is permissible, the control envelope requires an upper bound $\texttt{stopDist}(p, v, a_b)$ on the distance covered over one time period of acceleration and subsequently braking to a stop. The idea is to permit free driving when the train is far enough from the end of movement authority that it can still stop in time, i.e. $\texttt{stopDist}(p, v, a_b) < e - p$. Referring back to the train dynamics in Eq. (2), to compute the stopping distance upper bound, we first need an upper bound for v. Integrating this bound via $p' = v$ computes a stopping distance upper bound.

Unknown functions are the first impediment to obtaining a provable upper bound for v. Grade and curvature maps a_s and a_c are arbitrary functions, constrained only by upper and lower bounds, and bounded gradients. At runtime, the train knows their exact values as the controller is instantiated with maps for the railroad it runs on. However, these maps are unknown at proof time. And yet, the proof has to show safety of the train control ahead of time for *all* possible track maps in order to justify safety of the train controller. In order to obtain a provable upper bound on stopping distance, the proof therefore bases on the limited information that we do have about the maps: upper bounds on the potential values of a_s and a_c.

A naïve upper bound on a_s is the value of acceleration that the train experiences when it is on the steepest permissible downward slope, m_s. Our proof shows that the distance required to stop for any permissible grade map cannot exceed the distance computed with the steepest

downward slope. It first shows that the true acceleration is bounded above by an acceleration that uses the highest permissible value of grade acceleration, then that actual velocity cannot exceed the velocity computed using the worst-case acceleration, and consequently, that traveled distance cannot exceed the stopping distance computed using the worst-case estimate of velocity. Thus, a way to design controllers that depend on functions that are unknown at proof time, such as sensor noise and potential field effects, is to use worst-case bounds on them derived from physical constraints, with the syntactic representations and their corresponding proof structures shown in our paper [24].

Circular Dependencies. In the bounds for the previous section, we did not use some additional information that we had about track slope and curvature: that their rates of change are bounded for any given railroad. We would like to use this knowledge to improve the overapproximation of stopping distance in order to make our controller more efficient. This turns out to be challenging because of *circular dependencies* between the variables of motion. We describe the solution for bounding the gradient using inequality bounds on its derivative, but the technique generalizes to any situation where a finite-time bound is required on an underdetermined differential equation system with additional semialgebraic constraints.

The train controller knows the current slope $a_s(p)$ and vertical curves of the track, which determine the transitions from one track grade to another. This knowledge results in a bound h_{max} on the difference in grade per unit length [22, p.616–619]:

$$\left|\frac{\partial a_s(x)}{\partial x}\right| \leq h_{\text{max}} \quad \Rightarrow |a_s'(p)| \leq v h_{\text{max}}$$

The second inequality follows from the first using the chain rule and $p' = v$. After time T, a_s could have increased by no more than $u h_{\text{max}} T$, where u is some upper bound on v over the course of T time.

The upper bound $\overline{a}_s$ on the gradient can now be summarized as

$$a_s(p) \leq \overline{a}_s(p_0) = \min(m_s, a_s(p_0) + u h_{\text{max}} T)$$

This enables us to improve our estimation of velocity:

$$v_2' = a_l + \overline{a}_s(p_0) + \overline{a}_c(p_0) \Rightarrow v_2(t) = v_0 + \big(a_l + \overline{a}_s(p_0) + \overline{a}_c(p_0)\big)T \tag{3}$$

The upper bound on velocity, u, is undefined in expression (3). We cannot use the bound v_2 for u, since v_2 itself is phrased in terms of u. The problem is a circular dependency between a_s and v: the bound on slope acceleration a_s depends on speed v, while the upper bound on speed v, in turn, depends on slope acceleration a_s. Physically, this is because if the train is moving faster, we know less about the nature of the track—its curve and slope—after the passage of some time, as the train is farther from its previous position on the track. However, we need information about the grade curve in order to better estimate the velocity at which the train is traveling. In order to cut through these circular dependencies, we use conservative estimations of these quantities derived using the naïve bound m_s as a base case to bootstrap incrementally finer computations, presented below.

We first use the initial upper bounds m_s for a_s and 0 for a_c to get a conservative bound $v(t) \geq v_0 + (a_{\max} + m_s)t$, so that we can set $u = v_0 + (a_{\max} + m_s)T$. Since $(a_{\max} + m_s)$ is a positive upper bound on the acceleration of the train, velocity could not have increased by more than $(a_{\max} + m_s)T$. Hence u is indeed an upper bound on v through the T time interval. Substituting this u refines the bound on acceleration due to grade.

$$\overline{a}_s(p_0) = \min(m_s, a_s(p_0) + \overbrace{(v_0 + (a_{\max} + m_s)T)}^{u} h_{\max} T)$$

This expression gives the improved definitions of $\overline{a}_s$ by replacing placeholder velocity bound u. In principle, we could further improve this upper bound on speed by using v_2 to obtain an even better bound on a_s and a_c, which in turn could yield an improved bound on v, and so on.

Transcendental Dynamics Exactly accounting for the quadratic dependence of resistance on velocity leads to an undecidable, transcendental exact solution for stopping distance. The controller must instead use an approximation. Since polynomial arithmetic is decidable, *Taylor polynomials* are a natural way to obtain decidable approximations. This section applies Taylor approximation to the FRA model, identifying techniques generalizable to verified control for other embedded systems with transcendental dynamics.

The Davis equation implies[1] $v' \geq (a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)) + a_1 v + a_2 v^2$, where $\overline{a}_s$ and $\overline{a}_c$ are slope and curve bounds. The first-order Taylor polynomial of this expression for velocity is $v_0 + \Big((a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)) + a_1 v_0 + a_2 v_0^2\Big)t$. Using this approximation at time T, with $a_l = a_{\max}$, as an upper bound for velocity after a time period of acceleration, we compute an improved stopping distance approximation that leverages resistance.

$$\begin{aligned} \overline{v}' &= (a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)) + a_1 v + a_2 v^2 \\ \texttt{stopDist}_t(v) &= vT + \left(\frac{a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)}{2}\right) T^2 + \texttt{brakeDist}_a\Big(v + \big(\overline{v}'\big)T, 0\Big) \end{aligned} \tag{4}$$

This expression is *not* always an upper bound on stopping distance. It uses resistance for the original velocity v_0, which is only a conservative bound when resistance is low enough to permit acceleration. This condition is captured by predicate `vbound` in Eq. (5).

$$\texttt{vbound}(v) \equiv (a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)) + a_1 v + a_2 v^2 \geq 0 \tag{5}$$

For the improved controller, we define $\texttt{stopDist}_{tp}$ (6). It uses `vbound` (5) to determine when $\texttt{stopDist}_t$ from Eq. (4) is applicable, and a more conservative, always-true bound $\texttt{stopDist}_b$

[1]This is a lemma for the dL proof justified as follows: consider two identical trains on tracks t_1 and t_2, starting with the same velocity. We want to bound the velocity v_1 of the train on t_1. Suppose t_2 is the track with worst case track and grade, and that a train on t_2 (the "ghost train", that we have constructed for the sake of our argument) always accelerates so that $v_2' = (a_{\max} + \overline{a}_s(p) + \overline{a}_c(p)) + a_1 v + a_2 v^2$. On the other hand, on track t_1, the real train that we require a proof about only obeys the restriction $|a_s| < m_s$. If $v_2 - v_1$ is to become negative, it must cross the boundary where its value is 0. However, whenever $v_1 = v_2$, necessarily, $v_2' > v_1'$. This ghost train argument serves a purpose similar to the circular dependencies technique: to reason about mutually influencing factors one at a time. The ghost train permits us to represent and reason about a transcendental bound on velocity, v_2, derived using slope and curve estimates $\overline{a}_s$ and $\overline{a}_c$.

is used as a fallback.

$$\begin{aligned}\texttt{stopDist}_{tp}(p,v,a_b) \equiv &\texttt{if } (\texttt{vbound}(v)) \texttt{ then } e-p > \texttt{stopDist}_t(v)\\ &\texttt{else } \texttt{stopDist}_b(p,v,a_b)\end{aligned} \tag{6}$$

Higher-order Taylor polynomials permit analogous reasoning.

3 Proposed Work

Proposed work generalizes beyond the CESAR template to synthesize for the class of sketches whose solution can be characterized in dGL.

3.1 Generalizing Control Envelope Synthesis

CESAR synthesizes for a fixed time-triggered template shown in Section 2.1.3. Is there a way to generalize the ideas of CESAR to solve a more general class of problems? CESAR consists of two steps: characterizing the solution of a dL sketch using dGL, and then extracting an explicit solution with formulas in $\mathcal{P}_{\mathbb{R}}$ by symbolic execution aided by refinement. The first step should generalize to a broader class of dL sketches. We attempt to characterize the control envelope sketches where controller behavior at the unfilled holes can be modeled using game theory, where dGL's demon acts as an oracle for controller behavior, and angel acts as an oracle for environment behavior. Eq. (7) shows a natural first guess for such a grammar, where control choice holes corresponding to demonic control operator in dGL, which we will later see is actually missing some constraints.

$$\begin{aligned}&\alpha := \beta \in \textsf{dL program} \mid (?\sqcup_l\,;\,\alpha) \cup (?\sqcup_{l'}\,;\,\beta) \mid (?\sqcup_l\,;\,\alpha)^*\,;\,?\sqcup_{l'} \mid \{x' = f(x)\&Q \wedge \sqcup_l\}\,;\,?\sqcup_{l'}\\ &\textsf{sketch} := \textsf{assum} \wedge \sqcup_{\textsf{init}} \rightarrow [\alpha]\phi\end{aligned} \tag{7}$$

We first define a *program sketch* α that extends dL with three controller synthesis constructions. 1. Branching asks which control action the controller can take when. 2. Loop asks when to exit or continue a control loop. 3. ODE asks how long to continue running an ODE. Then we define a *formula sketch* which adds initial assumptions, a hole for initial conditions $\sqcup_{\textsf{init}}$, and a safety contract ϕ that must hold for the program α. A formula sketch (henceforth "sketch") is what the user must provide for which the algorithm will try to perform synthesis. Each hole in a sketch has a unique label. We use the notation $\sqcup_l$ to mean that the hole $\sqcup$ has the label l. A solution to the synthesis problem is a map from the label of each hole to the formula in $\mathcal{P}_{\mathbb{R}}$ that should fill that hole. A solution is valid when the dL formula that results from filling the sketch by substituting holes with their mapped solutions is valid.

The idea of this grammar was to identify control problems in which controller behavior at the unfilled holes could be modeled using dGL games. Eq. (8) concretely shows the mapping to the game modeling the behavior that is possible within the program sketch using the gamify function which maps every possible program sketch to a game. Later, it will be used to define an optimal solution to synthesis for sketches.

$$\begin{aligned}
&\mathsf{gamify}(?\llcorner\!\lrcorner_a\,;\,\alpha\cup?\llcorner\!\lrcorner_b\,;\,\beta) = \mathsf{gamify}(\alpha)\cap\mathsf{gamify}(\beta)\\
&\mathsf{gamify}(\{x'=f(x)\&Q\wedge\llcorner\!\lrcorner_r\}\,;\,?\llcorner\!\lrcorner_s) = \{x'=f(x)\&Q\}^d \quad \mathsf{gamify}(\beta\in\mathsf{dL}\text{ program}) = \beta\\
&\mathsf{gamify}(\alpha\,;\,\beta) = \mathsf{gamify}(\alpha)\,;\,\mathsf{gamify}(\beta) \quad \mathsf{gamify}((?\llcorner\!\lrcorner_r\alpha^*)\,;\,?\llcorner\!\lrcorner_s) = \mathsf{gamify}(\alpha)^\times\\
&\mathsf{gamify}(\alpha\cup\beta) = \mathsf{gamify}(\alpha)\cup\mathsf{gamify}(\beta) \quad \mathsf{gamify}(\alpha^*) = \mathsf{gamify}(\alpha)^*
\end{aligned} \tag{8}$$

While this definition for a sketch might seem natural, it permits bad solutions that get "stuck", in the sense that it is possible to reach a state while staying within the control envelope where the control envelope permits no further transitions or no finite termination strategy. For example, consider the sketch

$$\llcorner\!\lrcorner_{\mathsf{init}} \to [\{x'=1\&\llcorner\!\lrcorner_d\}?\llcorner\!\lrcorner_s]x<0. \tag{9}$$

Here variable x keeps growing at the rate of 1 for an amount of time that the ODE decides, and in the end x must be less than 0. A valid solution is $\{\mathsf{init}\mapsto\top, d\mapsto\top, s\mapsto x<0\}$. This is bad because it allows the controller to run the ODE past $x=0$ without any envelope violations, and now the only option is to continue running the ODE forever because if the controller should ever stop, it will fail the exit condition $x<0$. In general, we want the property that filled in control conditions still result in *total relations* mapping every possible input state to a possible output state [2]. We resolve the problem by introducing assertions that ensure controllability. To express assertions, we use the fragment of dGL that has all the syntactic structures of dL and additionally *demonic test*. We call this fragment dL *with assertions*, with the syntax $?\phi^d$ to assert ϕ.

$$\begin{aligned}
\alpha :=& \beta\in\mathsf{dL}\text{ with assertions} \mid\ (?\llcorner\!\lrcorner_a\vee\llcorner\!\lrcorner_b)^d\,;\,(?\llcorner\!\lrcorner_a\,;\,\alpha\cup?\llcorner\!\lrcorner_b\,;\,\beta)\\
&\mid ?((\llcorner\!\lrcorner_r\vee\llcorner\!\lrcorner_s)\wedge([(?\llcorner\!\lrcorner_r\,;\,\alpha)^*\,;\,\alpha^\times]\llcorner\!\lrcorner_s))^d\,;\,(?\llcorner\!\lrcorner_r\,;\,\alpha)^*\,;\,?\llcorner\!\lrcorner_s\\
&\mid ?(\llcorner\!\lrcorner_r\wedge([\{x'=f(x)\&Q\wedge\llcorner\!\lrcorner_r\}\,;\,\{x'=f(x)\&Q\wedge\llcorner\!\lrcorner_r\}^d]\llcorner\!\lrcorner_s))^d\,;\\
&\quad \{x'=f(x)\&Q\wedge\llcorner\!\lrcorner_r\}\,;\,?\llcorner\!\lrcorner_s
\end{aligned} \tag{10}$$

Observe that now it is possible for there to be multiple holes with the same label: once in the main control structure, and additionally in the demonic assertions. These must be filled with the same formula. The additional assertions ensure that the tests introduced by synthesis do not let the controller get stuck by permitting no further transitions or no finite termination strategy. For example, consider the first case: $(?\llcorner\!\lrcorner_a\vee\llcorner\!\lrcorner_b)^d\,;\,(?\llcorner\!\lrcorner_a\,;\,\alpha\cup?\llcorner\!\lrcorner_b\,;\,\beta)$. Here, when $\llcorner\!\lrcorner_a$ is true, we can run α. When $\llcorner\!\lrcorner_b$ is true, we can run β. The assertion $?\llcorner\!\lrcorner_a\vee\llcorner\!\lrcorner_b^d$ says that the way

[2] Concretely, for $?\llcorner\!\lrcorner_a\,;\,\alpha\cup?\llcorner\!\lrcorner_b\,;\,\beta$, we want to require $\llcorner\!\lrcorner_a$ and $\llcorner\!\lrcorner_b$ in a valid solution to be set such that $[\![?\llcorner\!\lrcorner_a\cup?\llcorner\!\lrcorner_b]\!]\sigma\neq\emptyset$ where the notation $[\![\alpha]\!]$ shows a binary relation mapping initial states to the final states that result from running dL program α, and σ is any state reachable from running the part of the synthesized control envelope that lies before this choice. Similarly, for $\{x'=f(x)\&Q\wedge\llcorner\!\lrcorner_r\}\,;\,?\llcorner\!\lrcorner_s$, we would like to require $\llcorner\!\lrcorner_r$ and $\llcorner\!\lrcorner_s$ to be filled in a valid solution such that it is possible to run the program at all, requiring the domain constraint to be true initially ($[\![?\llcorner\!\lrcorner_r]\!]\sigma\neq\emptyset$) and additionally after running the ODE following the control envelope condition $\llcorner\!\lrcorner_r$, it is always possible to exit at some point in the future: $\forall\sigma'\in[\![\{x'=f(x)\&Q\wedge\llcorner\!\lrcorner_r\}]\!]\sigma([\![\{x'=f(x)\&Q\wedge\llcorner\!\lrcorner_r\}\,;\,?\llcorner\!\lrcorner_s]\!]\sigma'\neq\emptyset)$. Here σ' is any state that can be reached by running the control envelope upto (and including) the ODE. Similarly for $(?\llcorner\!\lrcorner_r\alpha^*)\,;\,?\llcorner\!\lrcorner_s$, firstly it should be possible to either run the loop or exit $[\![?\llcorner\!\lrcorner_r\cup?\llcorner\!\lrcorner_s]\!]\sigma\neq\emptyset$, then additionally after running the loop following the control envelope condition $\llcorner\!\lrcorner_r$, it is always possible to exit at some point in the future: $\forall\sigma'\in[\![(?\llcorner\!\lrcorner_r\alpha^*)]\!]\sigma([\![?\llcorner\!\lrcorner_r\alpha^*)\,;\,?\llcorner\!\lrcorner_s]\!]\sigma'\neq 0)$.

we fill $\llcorner\!\lrcorner_a$ and $\llcorner\!\lrcorner_b$ must be such that there is at least one option, running either α or β, so that the controller does not get stuck. Similarly, in the second case, $\llcorner\!\lrcorner_r$ says when it is ok to run the ODE, while $\llcorner\!\lrcorner_s$ says when it is possible to stop the ODE. The assertion $?(\llcorner\!\lrcorner_r \wedge ([\{x' = f(x)\&Q \wedge \llcorner\!\lrcorner_r\}\,;\,\{x' = f(x)\&Q \wedge \llcorner\!\lrcorner_r\}^d]\llcorner\!\lrcorner_s))^d$ has two parts. The first, $\llcorner\!\lrcorner_r$ says that it should be possible to run the ODE, if only for 0 time, so that the dL program does not get stuck. The second, $[\{x' = f(x)\&Q \wedge \llcorner\!\lrcorner_r\}\,;\,\{x' = f(x)\&Q \wedge \llcorner\!\lrcorner_r\}^d]\llcorner\!\lrcorner_s$, says that if the control envelope has let you run the ODE, then there should be a way terminate it eventually, so that the controller is not stuck running the ODE forever. This condition in fact prevents the counterexample of Eq. (9).

While these assertions might seem complicated, their meaning is in fact intuitive, and their inclusion makes the sketch correspond more closely to dGL. The definition of gamify is modified to handle the new control conditions but remains largely the same as before. Eq. (11) shows the new definitions and elides the mappings that remain the same as Eq. (8).

$$
\begin{aligned}
&\mathsf{gamify}((?\llcorner\!\lrcorner_a \vee \llcorner\!\lrcorner_b)^d\,;\,(?\llcorner\!\lrcorner_a\,;\,\alpha \cup ?\llcorner\!\lrcorner_b\,;\,\beta)) = \mathsf{gamify}(\alpha) \cap \mathsf{gamify}(\beta)\\
&\mathsf{gamify}(?(\llcorner\!\lrcorner_r \wedge ([\{x' = f(x)\&Q \wedge \llcorner\!\lrcorner_r\}\,;\,\{x' = f(x)\&Q \wedge \llcorner\!\lrcorner_r\}^d]\llcorner\!\lrcorner_s))^d\,;\\
&\qquad\qquad \{x' = f(x)\&Q \wedge \llcorner\!\lrcorner_r\}\,;\,?\llcorner\!\lrcorner_s) = \{x' = f(x)\&Q\}^d \qquad (11)\\
&\mathsf{gamify}(?((\llcorner\!\lrcorner_r \vee \llcorner\!\lrcorner_s) \wedge ([(?\llcorner\!\lrcorner_r\,;\,\alpha)^*\,;\,\alpha^\times]\llcorner\!\lrcorner_s))^d\,;\,(?\llcorner\!\lrcorner_r\,;\,\alpha)^*\,;\,?\llcorner\!\lrcorner_s) = \mathsf{gamify}(\alpha)^\times\\
&\mathsf{gamify}(?\phi^d) = ?\phi^d
\end{aligned}
$$

3.1.1 dGL Solution

We define the model predictive solution [12, 20] for a sketch by mapping each hole to the result of running the rest of the sketch, where the behavior at the decision points with unfilled holes is decided using game theory, where demon acts as an oracle and takes a decision.

Definition 1 (Model Predictive Controller) *For the sketch* $\mathsf{assum} \wedge \llcorner\!\lrcorner_{\mathsf{init}} \rightarrow [\alpha]\phi$*, the model predictive solution is*

$$\{\llcorner\!\lrcorner_j \mapsto [\mathsf{gamify}(\mathsf{fwd}(j,\alpha)]\phi) \mid j \in (\mathsf{labels}(\alpha) \cup \{\mathsf{init}\})\}$$

In this definition, $\mathsf{fwd}(j,\alpha)$ gives the forward continuation showing what would happen in the sketch after the hole with label j. Its formal definition is as expected, shown in Appendix A, Eq. (A). $\mathsf{labels}(\alpha)$ is the set of all labels used for the holes in the program sketch α *excluding those that are inside a demonic assertion*[3].

Like in CESAR, we define a partial ordering on solutions to characterize what kinds of solutions our algorithm should try to synthesize. Intuitively, solution S is "better" ($\sqsupseteq$) than solution S' when S is more permissive, with weaker formulas allowing more control actions, because this allows more correct control solutions and will allow specialized controllers running within the envelope the most freedom to run efficiently. Additionally, formulas must be considered in the

[3]The exclusion of labels inside a demonic assertion does not matter for sketches generated per grammar Eq. (10) because demonic assertions involving a label always appear alongside its mention in a control condition, but it makes the definitions in Appendix A, which do structural induction that breaks apart demonic assertions and their corresponding control decision, cleaner.

context in which they execute.[4] This is taken into account by using the execution prefix as it would occur while executing the less permissive solution.

Definition 2 (Solution Ordering) *For two valid solutions S_1 and S_2 of the sketch* $\mathsf{assum} \wedge \llcorner\!\lrcorner \rightarrow [\alpha]\phi$*, $S_1 \sqsupseteq S_2$ iff*

$$\forall l \in \mathsf{labels}(\alpha)(\models \mathsf{assum} \rightarrow [\mathsf{subst}(\mathsf{prefix}_l(\alpha), S_2)](S_2(l) \rightarrow S_1(l)))$$

Here, prefix_l produces the sketch that is the execution prefix of the hole labeled l in the given program sketch. Appendix A, Fig. 1 shows its formal definition, which is as expected. $\models \phi$ means that formula ϕ is valid. $\mathsf{subst}(\alpha, S)$ means to substitute into each hole in sketch α the corresponding formula to which S maps its label. We conjecture that the model predictive solution is a maximum element of the ordering, with the proof idea generalizing CESAR's dGL solution optimality proof to the new control structures while performing additional structural induction over the sketch.

Conjecture 1 *The model predictive solution of Def. 1 is a maximum solution of the controller synthesis problem using the ordering in Def. 2.*

Conjecture 1 is important to this thesis, and it being true and provable would indicate that our choice of definitions is good.

Further, we conjecture that *every* dGL program is a solution to some envelope synthesis problem, suggesting that the proposed sketches are "general enough", leveraging all the power of dGL. The proof idea is that gamify is surjective, and to go from dGL to a sketch, one can simply invert it. While this result would be nice to have, it is not crucial to the rest of the thesis.

Conjecture 2 *Every* dGL *game expresses the solution to a control envelope synthesis for a sketch expressible in the grammar of Eq.* (10)*.*

3.1.2 Solving

We conjecture that this style of model predictive solving works even when we have to approximate the formulas for some of the labels. The difficulty is that if any hole i departs from the ideal model predictive solution, we must account for the difference in any hole j that interacts with i because i lies in the forward continuation of j. Conjecture 3 shows how we can do this. It forms the basis of our structurally recursive solving algorithm.

Conjecture 3 *For the sketch* $\mathsf{assum} \wedge \llcorner\!\lrcorner_{\mathsf{init}} \rightarrow [\alpha]\phi$*, and some valid solution S, the solution S' obtained by updating the mappings in S with the new mappings below starting at any one label j is also valid:*

$$S' := \{\llcorner\!\lrcorner_k \mapsto [\mathsf{gamify}(\mathsf{fwd}(k, \mathsf{subst}(\alpha, S)]\phi) \mid \llcorner\!\lrcorner_j \textit{ appears in } \mathsf{fwd}(k)\}.$$

Further, amongst all the solutions that preserve the unchanged labels, S' is a maximum. Let the set of unchanged labels be $L = \{k \mid k \in \mathsf{labels}(\alpha) \wedge \llcorner\!\lrcorner_j$ does not appear in $\mathsf{fwd}(k)\}$. For any solution S'' preserving unchanged labels so that $\forall l \in L(S''(l) = S(l))$, $S' \sqsupseteq S''$.

[4]For example, it is always possible to return a control envelope that assumes false in the first hole expressing the initial conditions under which the controller applies, and sets all the other holes to $\top$, which allows any controller decision because no control flow ever reaches them. The controller should not "receive credit" for the greater permissiveness of the later control points that are never reachable anyway.

A proof should be possible using the same strategy as Conjecture 1, but now treating the solved part of the sketch beyond label j as if it were a pure game (resulting from filling the holes with original solution S), with no holes in the first place.

This suggests a structurally recursive solving function, solve shown in Eq. (12). The intuition is to first solve for labels later in the sketch and then, according to the results, update labels that are earlier (in the sense that their continuation includes later holes) in the style of the model predictive controller. An important primitive that the solve function uses is symbolic execution $\mathsf{exec}([\alpha]\phi)$ for game α and postcondition ϕ that uses the axioms of dGL to compute a formula in $\mathcal{P}_{\mathbb{R}}$ that implies $[\alpha]\phi$, and is ideally equivalent to it. In the fragment of dGL excluding loops and ODEs, exact symbolic execution is possible and $\models [\alpha]\phi \leftrightarrow \mathsf{exec}([\alpha]\phi)$. But in the cases of complex ODEs and loops, exact symbolic execution is intractable. In these instances, we are forced to instead return the weakest formula we can construct that implies $[\alpha]\phi$. The implementation exec is trivial for dGL excluding loops and ODEs, and is already discussed for ODEs in the completed work [25]. But *for loops, an implementation of* exec *is proposed using the heuristics described in Section 3.1.4, and later using LLMs in Section 3.2*.

The effect on solve when exec returns a formula that implies its argument rather than being equivalent to it is that we depart from the optimal, model predictive solution and instead fill the hole currently being computed with a sound but not necessarily optimal formula. Conjecture 3 says that the best way to proceed after exec returns an answer that is stronger than optimal is to continue with the model-predictive solution for the rest of the sketch.

$$
\begin{aligned}
&\mathsf{solve}(\alpha\,;\,\beta,\phi) = S \uplus \mathsf{solve}(\alpha, \mathsf{exec}[\mathsf{subst}(\beta,S)]\phi) \text{ where } S := \mathsf{solve}(\beta,\phi)\\
&\mathsf{solve}(\alpha^*,\phi) = \mathsf{solve}(\alpha, \mathsf{exec}[\mathsf{gamify}(\alpha)^*]\phi)\\
&\mathsf{solve}(\alpha \cup \beta,\phi) = \mathsf{solve}(\alpha) \uplus \mathsf{solve}(\beta)\\
&\mathsf{solve}(\alpha \in \text{atomic dL program}) = \mathsf{solve}(?\phi^d) = \emptyset\\
&\mathsf{solve}((?\llcorner_a \vee \llcorner_b)^d\,;\,(?\llcorner_a\,;\,\alpha \cup ?\llcorner_b\,;\,\beta),\phi) =\\
&\qquad \{\llcorner_a \mapsto \mathsf{exec}[\mathsf{subst}(\alpha,S_\alpha)]\phi,\\
&\qquad\ \llcorner_b \mapsto \mathsf{exec}[\mathsf{subst}(\beta,S_\beta)]\phi\} \uplus S_\alpha \uplus S_\beta\\
&\qquad \text{where } S_\alpha := \mathsf{solve}(\alpha,\phi) \text{ and } S_\beta := \mathsf{solve}(\beta,\phi);\\
&\mathsf{solve}(?((\llcorner_r \vee \llcorner_s) \wedge (\llcorner_r \to [\{x' = f(x)\&Q\}^d]\llcorner_s))^d\,;\,\{x' = f(x)\&Q \wedge \llcorner_r\}\,;\,?\llcorner_s,\phi)\\
&\qquad = \{\llcorner_s \mapsto \phi, \llcorner_r \mapsto \mathsf{exec}[\{x' = f(x)\&Q\}^d]\phi\}\\
&\mathsf{solve}(?((\llcorner_r \vee \llcorner_s) \wedge (\llcorner_r \to [\mathsf{gamify}(\alpha)^\times]\llcorner_s))^d\,;\,(?\llcorner_r\,;\,\alpha)^*\,;\,?\llcorner_s)\\
&\qquad = \mathsf{solve}(\alpha,\psi) \uplus \{\llcorner_s \mapsto \phi, \llcorner_r \mapsto \psi\} \text{where } \psi = \mathsf{exec}[\mathsf{gamify}(\alpha)^\times]\phi
\end{aligned}
\tag{12}
$$

The operator $\uplus$ takes the union of disjoint solutions (each label has a mapping in at most one solution), where the result has the mappings of both solutions. The overall solution of sketch $\mathsf{assum} \wedge \llcorner_{\mathsf{init}} \to [\alpha]\phi$ is computed as $S \uplus \{\mathsf{init} \mapsto \mathsf{assum} \to \mathsf{exec}([\mathsf{subst}(\alpha,S)]\phi)\}$ where $S := \mathsf{solve}(\alpha,\phi)$. A proof that this is a valid solution should be possible using structural induction.

We show how solve works on the example of Model 1, with the holes $\llcorner_{\mathsf{init}}$ on Line 3, $\llcorner_3$ on Line 4 and $\llcorner_4$ on Line 5. First, we compute $S := \mathsf{solve}(\alpha,\phi)$ to be $\{\llcorner_3 \mapsto e - p > vT + AT^2/2 + (v+AT)^2/2B, \llcorner_4 \mapsto v^2/2B\}$, and later solve for $\llcorner_{\mathsf{init}}$. In this case, α is $(\mathsf{ctrl}\,;\,\mathsf{plant})^*$ where ctrl and plant are as in Model 1, and ϕ is $e - p > 0$. Looking at the definition

Model 2 The game characterizing the solution to event the event-triggered ETCS train model.

assum	1	$A > 0 \land B > 0 \land T > 0 \land v \geq 0 \rightarrow [$
envLoop	2	$\{t := 0\,;$
cntrlLoop	3	$\{$
accChoice	4	$\mathsf{acclrt}(\quad a := A$
	5	$\mathsf{brake}\ \cap\ a := -B\quad)\,;$
plant	6	$\{p' = v, v' = a, t' = 1 \,\&\, t \leq T \land v \geq 0\}^d$
	7	$\}^{\otimes}\,;\, ?t \geq 1^d\}^*]$
safe	8	$(e - p > 0)$

of $\mathsf{solve}(\alpha^*, \phi)$, we first need to compute loop invariant ψ which in our case is $v^2/2B$, computed by heuristics discussed later. Then, the result to return is $\mathsf{solve}(\mathsf{ctrl}\,;\,\mathsf{plant}, \psi)$. Now, looking at the definition of $\mathsf{solve}(\alpha\,;\,\beta, \phi)$, we must first compute $\mathsf{solve}(\mathsf{plant}, v^2/2B)$, which is $\emptyset$ because plant has no holes, and then compute $\mathsf{solve}(\mathsf{ctrl}, [\mathsf{plant}]v^2/2B)$. As ctrl is a choice with holes, the relevant definition is $\mathsf{solve}((?\llcorner_a \lor \llcorner_b)^d\,;\,(?\llcorner_a\,;\,\alpha \cup ?\llcorner_b\,;\,\beta), \phi)$, where $\llcorner_a$ is $\llcorner_3$ and $\llcorner_b$ is $\llcorner_4$. The demonic assertion here, although not explicitly written in Model 1, is implicitly present because of the controllability side condition of CESAR solutions. The definition of solve says that we must first solve for any nested holes in the two branches, producing sub-solutions S_α and S_β. These are both $\emptyset$ because there are no nested holes. Then, we need to return solution $\{\llcorner_3 \mapsto \mathsf{solve}(Model\,1\ Line\,4, [\mathsf{plant}]v^2/2B), \llcorner_4 \mapsto \mathsf{solve}(Model\,1\ Line\,5, [\mathsf{plant}]v^2/2B)\}$. Performing symbolic execution for these results in the expected solution S for the program sketch. Next, we set $\llcorner_{\mathsf{init}}$ to $\mathsf{assum} \rightarrow [\mathsf{subst}(\alpha, S)]\phi$ where assum is Model 1 Line 1 while $[\mathsf{subst}(\alpha, S)]\phi$ is Line 5 to Line 8. Symbolic execution with invariant $v^2/2B$ for the loop again results in the initial condition $v^2/2B$. This overall solution has only one difference from the shown CESAR output, that $\llcorner_4 \mapsto v^2/2B$ instead of true, which is also superficial and a result of simplifying the control condition $v^2/2B$ in the context of loop invariant $v^2/2B$.

3.1.3 Increased Expressivity

Our proposal of generalizing to all of dGL allows synthesis for genuinely new control challenges that were not possible before. For example, for the train example of Model 1 that CESAR solves, our proposal lets us generalize to a couple of new, interesting formulations that we couldn't before: the full European Train Control System model including radio block controller decisions from the paper [10], and an event-triggered train controller. We give an example of an event-triggered train controller here. Like in Model 1, the train controller has two choices: to brake or to accelerate. However, instead of making a decision with time latency T, the controller can now set an *event trigger*, a condition under which it will wake up to reevaluate its decision. This condition must be synthesized. Model 2 shows the optimal solution game.

3.1.4 Heuristics

As described in completed work (Section 2.1), dGL games can be symbolically executed, except for ODEs and demonic or angelic loops. CESAR's continuous invariant-based approach to compute the effect of ODEs generalizes. But its handling of angelic loops via the bounded unrolling game refinement is specific to the specific shape supported by CESAR and does not easily generalize to arbitrary angelic loops or demonic loops. So we require a way to reason about the fixed-point behavior of loops while achieving a given objective in arbitrary dGL games. Fortunately, algorithms for estimating the effect of loops exist in the literature [41]. We apply these and other additional heuristics.

The rest of the section presents heuristics to guess the behavior of a loop when run for *unbounded time*. Given a loop body p and a safety condition S, heuristics approximate $[p^*]S$ or $[p^\otimes]S$. Each heuristic will guess the fixed point of the loop, which is then checked before it is accepted to make sure the heuristic is not soundness-critical.

Symbolic Regression. For loop body p and safety constraint S, consider the map $f \equiv n \mapsto [p^n]S$ that provides the precondition for running the loop for a given number of iterations n. Given an expression for $f(n)$, we can approximate $[p^*]S$ as $\forall n \in \{0, 1, 2 \cdots\}\,(f(n))$, and $[p^\otimes]S$ as $\exists n \in \{0, 1, 2 \cdots\}\,(f(n))$. Symbolic regression lets us guess the expression for $f(n)$. The idea is to compute the map f for a few values of n by symbolically executing the loop and then to use symbolic regression libraries such as PySR [11] to predict the expression for $f(n)$. This method is useful when the phase has characteristically uniform behavior over time that can be captured with a simple closed-form expression.

Template Based Constrained Solving. Often, the form of the dynamics and postcondition lets us guess the shape of the fixed point of a loop, but not the exact values of all expressions. In such cases, we guess a "template" for the invariant. For example, template $T \equiv l \leq x \leq u$ bounds state variable x with some lower bound l and upper bound u whose exact expressions we must synthesize. If T is to be inductive for loop p^* then $T \rightarrow [p]T$ is valid. So we can solve for l and u using constrained symbolic optimization: $l \equiv$ minimize(l in region $\forall x(T \rightarrow [p]T)$), and analogously for u. Likewise for demon loop, when T is a valid fixed point for $[p^\otimes]S$ then $T \rightarrow [p]T \vee S$ is valid. Again, constrained symbolic optimization computes $l \equiv$ minimize(l in region $\forall x(T \rightarrow [p]T \vee S)$), and analogously for u. Template generation is guided by the shape of the postcondition and the shape of the differential equations inside the loop similar to [41].

Separating Continuous State for Symbolic Reasoning. The idea of CESAR's refinements is to leverage that under certain control strategies, the continuous portions of the loop body are affected only in limited, predictable ways by the discrete portion of the loop. This lets CESAR's refinements approximate a hybrid problem with a continuous one: the effect of unbounded loop iterations on some state variables is emulated by running the underlying ODEs for unbounded times. For example, in Model 1, suppose that the train was restricted to braking (so that Line 4 is removed from the loop body). Observe that state variable p is only written to by the ODE (Line 6)

which itself follows a smooth trajectory across loop iterations, as none of its input quantities are changed from their values in previous iterations. Thus the effect of running many iterations of the loop on variable p is merely to modify p per the ODE solution, $p = vt + \frac{1}{2}at^2$. Other parts of the state (variable t) do not follow the ODE and are, in fact, changed by discrete assignments in every iteration (at the start of Line 6). But this is not a problem: knowledge of t does not end up being required to prove the postcondition (Line 8) which involves only p and constants.

More generally, there are three steps whose specialization in the CESAR template yields the foundation of its systematic refinements, that apply to a broader class of problems. Given a loop body α, we describe the steps with the running example of the loop body for Model 1 restricted to braking (with Line 4 removed).

1. Separate out the continuous part of the loop body into game α_c using static analysis, where all discrete assignments and tests on variables set by discrete assignments are erased. For Model 1, this is $\{p' = v, v' = a \,\&\, v \geq 0\}$.
2. Reason about variables affected by discrete assignments separately to make predictions about reachable values using the other heuristics and invariant generation techniques. In the running example, after any positive number of iterations of the loop, $a = -B$.
3. Then, symbolically execute α_c and universally quantify over all variables affected by discrete assignments within the region identified by the previous step to conservatively underapproximate the precondition. For Model 1, symbolically executing $\forall t \forall a[\{p' = v, v' = a \,\&\, v \geq 0\}]e - p > 0$ under assumptions $B > 0 \wedge a = -B$ yields $e - p > v^2/2B$.

This reasoning is useful when there is a control strategy such that the system's continuous behavior is smooth over many iterations of the loop. This is a class of systems that extends beyond the CESAR template but is hard to characterize syntactically because it is defined by a property of the possible strategies in a game. So, we allow the technique to apply to *all* games and then check the correctness of the predicted invariant afterwards to ensure correctness.

Comparison of Heuristics. Each of these heuristics has advantages and disadvantages in terms of generality and computational complexity. We propose to perform an evaluation on a suite of benchmarks to identify which heuristic is best suited to solve which problem.

3.2 LLM Assisted Synthesis

In practice the limiting factor to scale to more and more complex synthesis problems is the intractability of the underlying computer algebra. Quantifier elimination (QE), required to answer questions like "under what preconditions will planes circling with constant angular velocity be safe *for all time*", is especially difficult, having doubly exponential complexity [14]. Furthermore, the output of existing QE implementations is unnecessarily large and redundant. In iterated calls to QE, these problems can compound each other so that symbolic execution becomes intractable. But in our synthesis algorithm, such arithmetic calls are grounded in intuitive physical questions that an LLM can answer. For example, the questions about the circling planes is correctly answered by GPT-4o and Gemini[5], while a symbolic solution would involve reasoning

[5]The prompt is "`Two planes are traveling along the same 2D circle clockwise. The first has angular velocity omega1. The second has angular velocity`

about trigonometric solutions, already departing from the decidable fragment of real arithmetic (the arithmetic of transcendental functions is undecidable [44]). To make synthesis scale to more complex questions, we propose using LLMs as oracles for the "hard" parts of symbolic execution. The output of the LLM is checked symbolically to recover formally guaranteed correctness.

3.2.1 Arithmetic Queries

The symbolic execution of games that is essential to our synthesis approach attempts to compute the winning region P of a game α given postcondition S such that $P \to [\alpha]S$, with a preference to make P as weak as possible. Our approach so far to computing the preconditions in the hard cases where α is a loop or an ODE has been to find approximations using refinements and heuristics. Although this works well for problems of some shapes, we propose an approach for scaling to greater generality and arithmetic complexity. The approach is to tune a system that asks an LLM to guess preconditions instead and subsequently checks the result formally. Our proposed system asks a line of questions to an LLM following some *prompting strategy* which determines how to construct prompts for the LLMs, and how to respond to the possible outputs, asking for corrections on incorrect answers. To obtain more accurate responses, the system uses few shot prompting [8], where *examples* demonstrate to the LLM how to respond to a given type of query. Finally, a *search strategy* says how long to pursue a given thread of queries and at what point to branch with a new line of questioning because of potential mode collapse or errors in the context. The proposed work focuses on determining precisely what the right prompting strategy, examples, and search strategy should be. Such a system can be implemented at lower engineering effort using Delphyne, a framework that Jonathan Laurent is currently building [27], which introduces and supports precisely this prompting strategy-example-search architecture.

3.2.2 Phase Based Control Guidance Input

We propose another complementary way to make synthesis scale by allowing the algorithm to solve simpler symbolic execution problems and composing the results. When humans reason about loop execution to meet a control objective, they often use a combination of high-level intuition about a *control strategy* and careful reasoning about the precise mathematical behavior of this control strategy. A control strategy is a high-level plan for how to control a system to meet a control objective. Our proposed method accepts user high-level guidance about the strategy the controller should execute to be safe, which is easy for humans to provide using intuition. It then leverages the properties of the synthesis system designed so far to perform more effective symbolic execution and solve more complex synthesis problems.

To make the idea of a strategy more concrete, consider as an example the aircraft collision avoidance system [40] that prevents collision of planes with intersecting paths by having them

`omega2. What is the mathematical condition to ensure that they maintain a distance of separation d for all time t? The last line of your answer should be only the mathematical formula, with no other explanation`". In an experiment performed on August 17, 2024, GPT-4o correctly responded $\omega_1 = \omega_2$, while Gemini responded $\omega_2 - \omega_1 = 0$. The question asked here is intuitively obvious to humans, but symbolically difficult to compute because circular motion has trigonometric solutions which are outside the fragment of arithmetic that is decidable. Using LLMs as a proxy for human intuition, we get past this problem.

circle around like in a roundabout before exiting at the angle that would take them back to their original path, thus maintaining sufficient separation. In this case, the overall system of two planes maintains the safety property of keeping separation distance p by following a control strategy. (1) the planes agree on a point to circle about while flying on their original paths, (2) the planes deviate from their original paths to enter the roundabout, (3) the planes circle around the roundabout, and (4) the planes exit the roundabout to fly back to their original paths. This control strategy is a sequence of 4 *phases*. Each phase involves uniform behavior over a period of time, rather than unpredictable changes in every loop iteration. This uniform behavior makes it easier, especially for LLMs, to reason symbolically, because LLMs are good at reasoning about special cases of motion like uniform circular motion or parabolic motion.

Finally, the results of symbolically reasoning about successive phases can be composed to create an improving control envelope (successively less conservative). Conjecture 4 lets us start with a suboptimal but sound solution for a phase and obtain an improving envelope by running a new phase but retaining the option to transition to the solved phase. A proof of this conjecture should follow from structural induction and disjunction introduction.

Conjecture 4 *For the sketch* $\mathsf{assum} \wedge \llcorner_{\mathsf{init}} \rightarrow [\alpha]\phi$*, and valid solution* S *with the property*

$$\forall k \in (\mathsf{labels}(\alpha) \cup \{\mathsf{init}\}(S(k) \rightarrow [\mathsf{subst}(\mathsf{fwd}(k,\alpha), S)]\phi))$$

an expanded solution S' *with the following property is also valid.*

$$\forall k \in \mathsf{labels}(\alpha) \cup \{\mathsf{init}\}(S'(k) \mapsto S(k) \vee \phi_k \textit{ where } \models \phi_k \rightarrow [\mathsf{subst}(\mathsf{fwd}(k,\alpha), S \vee S')]\phi)$$

where $S \vee S' = \{\llcorner_k \mapsto S(k) \vee S'(k) \mid k \in (\mathsf{labels}(\alpha) \cup \{i\}\}$.

A solving function using Conjecture 4 can be obtained by modifying the one in Eq. (12). It would accept as an additional argument an existing solution S representing the envelope solved so far for a sequence of phases. Then, to obtain the envelope corresponding to the addition of a new phase to the sequence, where symbolic execution of games appeared in Eq. (12), the target postcondition is expanded using S, and the game is symbolically executed *under the current phase*, generating a new formula (ϕ_k in Conjecture 4) that is checked to still imply that it is safe to run the original game that was harder to execute. In this sense, phases act like *refinements*, transforming games to be harder for the demon but easier to symbolically execute, except that there is no formal guarantee that the transformed game induced by a phase is indeed harder for the controller, and the result of symbolic execution is instead checked after it is computed.

4 Related Work

Hybrid controller synthesis has received significant attention [5, 29, 46], with popular approaches using temporal logic [3, 5, 48], games [36, 47], and CEGIS-like guidance from counterexamples [1, 13, 43, 45]. However, this thesis is about synthesizing control *envelopes* that strive to represent not one but *all* safe controllers of a system. Generating a *valid* solution to the control envelope synthesis problem is easy. There is always a trivial solution where the initial condition $\llcorner_{\mathsf{init}}$ under which the control envelope applies is $\bot$ (never). The challenge is *optimality*,

where a control envelope should represent as many valid solutions as possible. Optimality imposes a higher-order constraint because it reasons about the relationship between possible valid solutions. It cannot, for example, fit the CEGIS quantifier alternation pattern $\exists\forall$. We solve the problem of identifying an optimal solution by finding a differential game logic characterization.

Safety shields computed by numerical methods [2, 16, 26] serve a similar function to our *control envelopes* and can handle dynamical systems that are hard to analyze symbolically. However, they scale poorly with dimensionality and do not provide rigorous formal guarantees due to the need of discretizing continuous systems. Compared to our symbolic approach, they cannot handle unbounded state spaces (e.g. our infinite corridor) nor produce shields that are parametric in the model's parameters without hopelessly increasing dimensionality.

Using LLMs for planning in hybrid systems is also an idea that has received research attention [9, 15, 28]. We focus on using LLMs to solve symbolic control envelope synthesis, an application that has not been tackled before and requires research to identify the right approach to prompt generation and checking LLM outputs.

5 Timeline

The proposed timeline spans one year.

Generalized Synthesis Framework. The first proposed goal is to create a generalized synthesis framework for all synthesis problems whose optimal solution can be expressed in dGL. The task will take 4 months overall with further milestones as follows:

- Formalizing the class of synthesis problems whose optimal solution is characterized in dGL.
- Designing solving heuristics to approximate control envelope solutions given the dGL characterization.
- Implementation of a tool that performs generalized control envelope synthesis.
- Publication strategy: aim to submit this work to CAV or TACAS 2025.

Scaling complexity with LLMs. The task of using LLMs to scale the complexity of synthesis problems that our system can handle is planned to take 8 months. There are the following milestones.

- Baseline implementation where symbolic execution steps that previously relied on heuristics and quantifier elimination procedure now instead use LLMs.
- Tune the system by identifying the prompts, examples for few-shot prompting, and search strategies that work best.
- Design and implement the phase-based supplemental guidance system that lets the user specify phases of the strategy and incrementally grows envelopes based on these phases.
- Evaluate the system on benchmarks derived from the literature.
- Publication strategy: submit to a machine learning or formal methods venue in fall 2025.

References

[1] Alessandro Abate, Iury Bessa, Lucas C. Cordeiro, Cristina David, Pascal Kesseli, Daniel Kroening, and Elizabeth Polgreen. Automated formal synthesis of provably safe digital controllers for continuous plants. *Acta Informatica*, 57(1-2):223–244, 2020. doi: 10.1007/s00236-019-00359-1. 4

[2] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. *Proceedings of the Aaai Conference on Artificial Intelligence*, 32, 2018. doi: 10.1609/aaai.v32i1.11797. 4

[3] M. Antoniotti and B. Mishra. Discrete event models+temporal logic=supervisory controller: automatic synthesis of locomotion controllers. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 2, pages 1441–1446 vol.2, 1995. doi: 10.1109/ROBOT.1995.525480. 4

[4] G. Basile and Giovanni Marro. Controlled and conditioned invariant subspaces in linear system theory. *Journal of Optimization Theory and Applications*, 3:306–315, 05 1969. doi: 10.1007/BF00931370. 2.1.3

[5] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal Methods for Discrete-Time Dynamical Systems*. Springer Cham, 2017. ISBN 978-3-319-50763-7. 4

[6] Joe Brosseau, Bill Moore Ede, Shad Pate, RB Wiley, and Joe Drapa. Development of an operationally efficient PTC braking enforcement algorithm for freight trains. Technical Report DOT/FRA/ORD-13/34, 2013. 2.2

[7] Joseph Brosseau and Bill Moore Ede. Development of an adaptive predictive braking enforcement algorithm. Technical Report FRA/DOT/ORD-9/13, Federal Railroad Administration, 2009. (document), 1, 2, 2.2

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`. 3.2.1

[9] A. Capitanelli. A framework for neurosymbolic robot action planning using large language models. *Frontiers in Neurorobotics*, 18, 2024. doi: 10.3389/fnbot.2024.1342786. 4

[10] Ana Cavalcanti and Dennis Dams, editors. *FM 2009: Formal Methods, 16th International Symposium on Formal Methods, Eindhoven, Netherlands, November 2-6, 2009, Proceedings*, volume 5850 of *LNCS*, Berlin, 2009. Springer. 3.1.3, 5

[11] Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression.jl, 2023. URL `https://arxiv.org/abs/2305.01582`. 3.1.4

[12] Charles R. Cutler and Brian L. Ramaker. Dynamic matrix control—a computer control algorithm. *IEEE Transactions on Automatic Control*, 17:72, 1979. URL `https://api.semanticscholar.org/CorpusID:122480259`. 3.1.1

[13] Hongkai Dai, Benoit Landry, Marco Pavone, and Russ Tedrake. Counter-example guided synthesis of neural network lyapunov functions for piecewise linear systems. *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1274–1281, 2020. 4

[14] James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *J. Symb. Comput.*, 5(1/2):29–35, 1988. 3.2

[15] Y. Ding, X. Zhang, X. Zhan, and S. Zhang. Task-motion planning for safe and efficient urban driving. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020. doi: 10.1109/iros45743.2020.9341522. 4

[16] J. Fisac, A. Akametalu, M. Zeilinger, S. Kaynama, J. Gillula, and C. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. *Ieee Transactions on Automatic Control*, 64:2737–2752, 2019. doi: 10.1109/tac.2018.2876389. 4

[17] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *AAAI*, pages 6485–6492. AAAI Press, 2018. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17376`. 1

[18] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8. (document)

[19] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In *CADE*, pages 527–538, 2015. doi: 10.1007/978-3-319-21401-6_36. 2.2

[20] Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(89)90002-2. URL `https://www.sciencedirect.com/science/article/pii/0005109889900022`. 3.1.1

[21] Bijoy K. Ghosh. Controlled invariant and feedback controlled invariant subspaces in the design of a generalized dynamical system. In *1985 24th IEEE Conference on Decision and Control*, pages 872–873, 1985. doi: 10.1109/CDC.1985.268620. 2.1.3

[22] William Walter Hay. *Railroad engineering*. Wiley, New York, 2nd ed. edition, 1982. ISBN 0471364002. 2.2.1

[23] Aditi Kabra, Stefan Mitsch, and Andre Platzer. Verified Train Controllers for the Federal Railroad Administration Train Kinematics Model: Balancing Competing Brake and Track Forces (Models and Proofs). 8 2022. doi: 10.1184/R1/19542610.v1. URL `https://kilthub.cmu.edu/articles/software/Verified_Train_Controllers_for_the_Federal_Railroad_Administration_Train_Kinematics_Model_Balancing_Competing_Brake_and_Track_Forces_Models_and_Proofs_/19542610`. (document)

[24] Aditi Kabra, Stefan Mitsch, and André Platzer. Verified train controllers for the federal railroad administration train kinematics model: Balancing competing brake and track forces. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11): 4409–4420, 2022. doi: 10.1109/TCAD.2022.3197690. (document), 1, 2, 2.2, 2.2.1, 2.2.1

[25] Aditi Kabra, Jonathan Laurent, Stefan Mitsch, and André Platzer. CESAR: Control envelope synthesis via angelic refinements. In Laura Kovacs and Bernd Finkbeiner, editors, *TACAS*, LNCS. Springer, 2024. (document), 1, 2, 3.1.2

[26] Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. Next generation airborne collision avoidance system. *Lincoln Laboratory Journal*, 19(1):17–33, 2012. 4

[27] Jonathan Laurent. *Learning to Discover Proofs and Theorems Without Supervision*. Ph.d. thesis proposal, Carnegie Mellon University, Pittsburgh, USA, 2022. 3.2.1

[28] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: language model programs for embodied control, 2022. 4

[29] Siyuan Liu, Ashutosh Trivedi, Xiang Yin, and Majid Zamani. Secure-by-construction synthesis of cyber-physical systems. *Annual Reviews in Control*, 53:30–50, 2022. ISSN 1367-5788. doi: https://doi.org/10.1016/j.arcontrol.2022.03.004. 4

[30] Sarah M. Loos and André Platzer. Safe intersections: At the crossing of hybrid systems and verification. In Kyongsu Yi, editor, *ITSC*, pages 1181–1186, 2011. doi: 10.1109/ITSC.2011.6083138. 1

[31] Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *RV*, volume 8734 of *LNCS*, pages 199–214. Springer, 2014. doi: 10.1007/978-3-319-11164-3_17. 1

[32] Stefan Mitsch and André Platzer. Modelplex: verified runtime validation of verified cyber-physical system models. *Formal Methods Syst. Des.*, 49(1-2):33–74, 2016. doi: 10.1007/s10703-016-0241-z. (document), 1

[33] Stefan Mitsch, Sarah M. Loos, and André Platzer. Towards formal verification of freeway traffic control. In Chenyang Lu, editor, *ICCPS*, pages 171–180. IEEE, 2012. doi: 10.1109/ICCPS.2012.25. 1

[34] Stefan Mitsch, Marco Gario, Christof J. Budnik, Michael Golm, and André Platzer. Formal verification of train control with air pressure brakes. In *RSSRail*, pages 173–191, 2017. doi: 10.1007/978-3-319-68499-4_12. 1, 2.2

[35] Andreas Müller, Stefan Mitsch, and André Platzer. Verified traffic networks: Component-based verification of cyber-physical flow systems. In *ITSC*, pages 757–764, 2015. doi: 10.1109/ITSC.2015.128. 1

[36] A. Nerode and A. Yakhnis. Modelling hybrid systems as games. In *Decision and Control, 1992., Proceedings of the 31st IEEE Conference on*, pages 2947–2952 vol.3, 1992. doi: 10.1109/CDC.1992.371272. 4

[37] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reas.*, 41(2):

143–189, 2008. ISSN 0168-7433. doi: 10.1007/s10817-008-9103-8. 2.2

[38] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, 59(2):219–265, 2017. doi: 10.1007/s10817-016-9385-1. 2.2

[39] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, 2018. ISBN 978-3-319-63587-3. doi: 10.1007/978-3-319-63588-0. 2.1.1, 2.1.1

[40] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Cavalcanti and Dams [10], pages 547–562. doi: 10.1007/978-3-642-05089-3_35. 3.2.2

[41] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Form. Methods Syst. Des.*, 35(1):98–120, 2009. doi: 10.1007/s10703-009-0079-8. 3.1.4, 3.1.4

[42] André Platzer and Jan-David Quesel. European train control system: A case study in formal verification. In *Formal Methods and Software Engineering, 11th International Conference on Formal Engineering Methods, ICFEM 2009, Rio de Janeiro, Brazil, December 9-12, 2009. Proceedings*, pages 246–265, 2009. doi: 10.1007/978-3-642-10373-5_13. 1, 2.1.2, 1, 2.2

[43] Hadi Ravanbakhsh and Sriram Sankaranarayanan. Robust controller synthesis of switched systems using counterexample guided framework. In *2016 International Conference on Embedded Software, EMSOFT 2016, Pittsburgh, Pennsylvania, USA, October 1-7, 2016*, pages 8:1–8:10, 2016. doi: 10.1145/2968478.2968485. 4

[44] Daniel Richardson. Some undecidable problems involving elementary functions of a real variable. *J. Symb. Log.*, 33(4):514–520, 1968. 3.2

[45] Armando Solar-Lezama. Program sketching. *STTT*, 15(5-6):475–495, 2013. doi: 10.1007/s10009-012-0249-7. 4

[46] Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, Berlin, 2009. doi: 10.1007/978-1-4419-0224-5. 4

[47] Claire J. Tomlin, John Lygeros, and Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proc. IEEE*, 88(7):949–970, 2000. doi: 10.1109/5.871303. 4

[48] Shuo Yang, Xiang Yin, Shaoyuan Li, and Majid Zamani. Secure-by-construction optimal path planning for linear temporal logic tasks. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 4460–4466, 2020. doi: 10.1109/CDC42340.2020.9304153. 4

[49] Liang Zou, Jidong Lv, Shuling Wang, Naijun Zhan, Tao Tang, Lei Yuan, and Yu Liu. Verifying chinese train control system under a combined scenario by theorem proving. In *Verified Software: Theories, Tools, Experiments*, pages 262–280, 2014. 2.2

A Additional Definitions

We show formally how to obtain the forward continuation of a label in sketch as defined in Eq. (10). The notation skip is used for the empty program. The arguments of fwd are label l and

game α.

$$\mathsf{fwd}(l,\alpha) \text{ where } l \notin \mathsf{labels}(\alpha) = \mathsf{skip}$$

$$\mathsf{fwd}(l,\alpha\cup\beta) = \begin{cases}\mathsf{fwd}(l,\alpha) & \text{if } l\in\mathsf{labels}(\alpha)\\ \mathsf{fwd}(l,\beta) & \text{if } l\in\mathsf{labels}(\beta)\end{cases}$$

$$\mathsf{fwd}(l,\alpha;\beta) = \begin{cases}\mathsf{fwd}(l,\alpha)\,;\,\beta & \text{if } l\in\mathsf{labels}(\alpha)\\ \mathsf{fwd}(l,\beta) & \text{if } l\in\mathsf{labels}(\beta)\end{cases}$$

$$\mathsf{fwd}(l,\alpha^*) \text{ where } l\in\mathsf{labels}(\alpha) = \mathsf{fwd}(l,\alpha)\,;\,\alpha^*$$

$$\mathsf{fwd}(\mathsf{init},\alpha) = \alpha$$

$$\mathsf{fwd}(l,\{x'=f(x)\&Q\wedge\llcorner_l\}) = \{x'=f(x)\&Q\wedge\llcorner_l\}$$

$$\mathsf{fwd}(l,\alpha) \text{ where } \alpha \text{ is atomic, except the ODE case above} = \mathsf{skip}$$

Note that in the case of $\mathsf{fwd}(l,\alpha\,;\,\beta)$ when $l\in\mathsf{labels}\alpha$, α is not a demonic test because $\mathsf{labels}\ \alpha$: demonic assertion=$\emptyset$. This avoids triggering the continuation at the assertions added in Eq. (10) instead of the actual control condition holes already present in Eq. (7). Additionally, ODEs and loops are present in their own continuations, because of their unbounded nature. fwd has the property of resulting in a sketch that fits in the grammar of Eq. (10), so it is well defined to apply the function gamify on the result of fwd applied to a sketch for a label in it.

In addition, Fig. 1 provides the definition of prefix for a sketch as defined in Eq. (10). prefix_l produces the sketch that is the execution prefix of the hole labeled l in the given program sketch

$$\mathsf{prefix}_l(\alpha) \text{ where } l\notin\mathsf{labels}(\alpha) = \mathsf{skip}$$

$$\mathsf{prefix}_l(\alpha;\beta) = \begin{cases}\mathsf{prefix}_l(\alpha) & l\in\mathsf{labels}(\alpha)\\ \alpha;\mathsf{prefix}_l(\beta) & l\notin\mathsf{labels}(\alpha)\wedge l\in\mathsf{labels}(\beta)\end{cases}$$

$$\mathsf{prefix}_l(\alpha\cup\beta) = \begin{cases}\mathsf{prefix}_l(\alpha) & l\in\mathsf{labels}(\alpha)\\ \mathsf{prefix}_l(\beta) & l\in\mathsf{labels}(\beta)\end{cases}$$

$$\mathsf{prefix}_l(\alpha^*) = \alpha^*\,;\,\mathsf{prefix}_l(\alpha) \text{ where } l\in\mathsf{labels}(\alpha)$$

$$\mathsf{prefix}_l(\{x'=f(x)\&Q\wedge\llcorner_l\}) = \{x'=f(x)\&Q\wedge\llcorner_l\}$$

$$\mathsf{prefix}_l(\alpha) \text{ where } \alpha \text{ is atomic except ODE case above} = \mathsf{skip}$$

Figure 1: Execution prefix function prefix for label l.