CESAR: Control Envelope Synthesis via Angelic Refinements *

Aditi Kabra[1(✉)], Jonathan Laurent[1,2], Stefan Mitsch[1,3], and André Platzer[1,2]

[1] Carnegie Mellon University, Pittsburgh, USA
akabra@cs.cmu.edu
[2] Karlsruhe Institute of Technology, Karlsruhe, Germany
{jonathan.laurent,platzer}@kit.edu
[3] DePaul University, Chicago, USA
smitsch@depaul.edu

Abstract. This paper presents an approach for synthesizing provably correct control envelopes for hybrid systems. Control envelopes characterize families of safe controllers and are used to monitor untrusted controllers at runtime. Our algorithm fills in the blanks of a hybrid system's sketch specifying the desired shape of the control envelope, the possible control actions, and the system's differential equations. In order to maximize the flexibility of the control envelope, the synthesized conditions saying which control action can be chosen when should be as permissive as possible while establishing a desired safety condition from the available assumptions, which are augmented if needed. An implicit, optimal solution to this synthesis problem is characterized using hybrid systems game theory, from which explicit solutions can be derived via symbolic execution and sound, systematic game refinements. Optimality can be recovered in the face of approximation via a dual game characterization. The resulting algorithm, *Control Envelope Synthesis via Angelic Refinements (CESAR)*, is demonstrated in a range of safe control envelope synthesis examples with different control challenges.

Keywords: Hybrid systems · Program synthesis · Differential game logic

1 Introduction

Hybrid systems are important models of many applications, capturing their differential equations and control [27,41,3,33,4,28]. For overall system safety, the correctness of the control decisions in a hybrid system is crucial. Formal verification techniques can justify correctness properties. Such correct controllers have

* This work was funded by the Federal Railroad Administration Office of Research, Development and Technology under contract number 693JJ620C000025, a Swartz Center Innovation Commercialization Fellowship, and an Alexander von Humboldt Professorship.

B. Finkbeiner and L. Kovács (Eds.): TACAS 2024, LNCS 14570, pp. 144–164, 2024.
https://doi.org/10.1007/978-3-031-57246-3_9

been identified in a sequence of challenging case studies [34,40,12,32,19,14,22]. A useful approach to verified control is to design and verify a safe *control envelope* around possible safe control actions. Safe control envelopes are nondeterministic programs whose every execution is safe. In contrast with controllers, control envelopes define entire families of controllers to allow control actions under as many circumstances as possible, as long as they maintain the safety of the hybrid system. Safe control envelopes allow the verification of abstractions of control systems, isolating the parts relevant to the safety feature of interest, without involving the full complexity of a specific control implementation. The full control system is then monitored for adherence to the safe control envelope at runtime [29]. The control envelope approach allows a single verification result to apply to multiple specialized control implementations, optimized for different objectives. It puts industrial controllers that are too complex to verify directly within the reach of verification, because a control envelope only needs to model the safety-critical aspects of the controller. Control envelopes also enable applications like justified speculative control [17], where machine-learning-based agents control safety-critical systems safeguarded within a verified control envelope, or [36], where these envelopes generate reward signals for reinforcement learning.

Control envelope design is challenging. Engineers are good at specifying the *shape* of a model and listing the possible control actions by translating client specifications, which is crucial for the fidelity of the resulting model. But identifying the exact control conditions required for safety in a model is a much harder problem that requires design insights and creativity, and is the main point of the deep area of control theory. Most initial system designs are incorrect and need to be fixed before verification succeeds. Fully rigorous justification of the safety of the control conditions requires full verification of the resulting controller in the hybrid systems model. We present a synthesis technique that addresses this hard problem by filling in the holes of a hybrid systems model to identify a correct-by-construction control *envelope* that is as permissive as possible.

Our approach is called *Control Envelope Synthesis via Angelic Refinements (CESAR)*. The idea is to implicitly characterize the optimal safe control envelope via hybrid games yielding maximally permissive safe solutions in differential game logic [33]. To derive explicit solutions used for controller monitoring at runtime, we successively refine the games while preserving safety and, if possible, optimality. Our experiments demonstrate that CESAR solves hybrid systems synthesis challenges requiring different control insights.

Contributions. The primary contributions of this paper behind CESAR are:

- optimal hybrid systems control envelope synthesis via hybrid games.
- differential game logic formulas identifying optimal safe control envelopes.
- refinement techniques for safe control envelope approximation, including *bounded fixpoint unrollings* via a recurrence, which exploits *action permanence* (a hybrid analogue to idempotence).
- a primal/dual game counterpart optimality criterion.

2 Background: Differential Game Logic

We use hybrid games written in differential game logic (dGL, [33]) to represent solutions to the synthesis problem. Hybrid games are two-player noncooperative zero-sum sequential 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, the other one 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 paper.

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 [33] for full exposition. 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

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
$(x_1' = \theta_1, \ldots, x_n' = \theta_n \;\&\; \psi)$	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

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 dynamic logic (dL) [33]. 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. 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. We define $\alpha^{*\leq n}$ and $\alpha^{\times \leq n}$ for angelic and demonic repetitions respectively of at most n times.

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):

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

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 [33]. A formula ϕ is *valid*, written $\vDash \phi$, iff it is true in every state ω. 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.

Control Safety Envelopes by Example. In order to separate safety critical aspects from other system goals during control design, we abstractly describe the safe choices of a controller with safe control envelopes that deliberately underspecify when and how to exactly execute certain actions. They focus on describing in which regions it is safe to take actions. For example, Model 1 designs a train control envelope [34] 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 6 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

Model 1 The train ETCS model (slightly modified from [34]). 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	$(\quad (?\boxed{e - p > vT + AT^2/2 + (v + AT)^2/2B}; a := A)$
	4	$\cup\ (?\boxed{\text{true}}; a := -B)\quad);$
plant	5	$(t := 0; \{p' = v, v' = a, t' = 1\ \&\ t \leq T \land v \geq 0\})$
safe	6	$\}^*](e - p > 0)$

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 5). These assumptions are fundamentally about the physics of the problem being considered. In contrast, Line 2 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 3 and Line 4 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 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 5 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*.

Overview of CESAR. CESAR first identifies the optimal solution for the blank of Line 2. 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 3 and Line 4) 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.

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 (Section 3.7). 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 $a := -B\,;\, t := 0\,;\, \{p' = v, v' = a \;\&\; \wedge\, v \geq 0\}$, 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 2.

Intuitively, this refinement (formalized in Section 3.4) 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 (Section 3.5). 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 3 and Line 4). 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 3.

3 Approach

This section formally introduces the *Control Envelope Synthesis via Angelic Refinements (CESAR)* approach for hybrid systems control envelope synthesis.

3.1 Problem Definition

We frame the problem of *control envelope synthesis* in terms of filling in holes $\sqcup$ in a problem of the following shape:

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

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 $\sqcup_i$ to be determined in a way that ensures safety within a controllable invariant [6,18] $\sqcup$ to be synthesized also. The plant is defined by the following template:

$$\mathsf{plant} \;\equiv\; t := 0\,;\, \{x' = f(x), t' = 1 \;\&\; \mathsf{domain} \wedge t \leq T\}. \tag{2}$$

This ensures that the plant must yield to the controller after time T at most, where T is assumed to be positive and constant. 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 $\vDash \langle \mathsf{act}_i \rangle$ 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.

Definition 1. *A* solution *to the synthesis problem above is defined as a pair (I, G) where I is a formula and G maps each action index i to a formula G_i. In addition, the following conditions must hold:*

1. *Safety is guaranteed:* $\mathsf{prob}(I, G) \equiv \mathsf{prob}[\llcorner \mapsto I, \llcorner_i \mapsto G_i]$ *is valid and* $(\mathsf{assum} \wedge I)$ *is a loop invariant that proves it so.*
2. *There is always some action:* $(\mathsf{assum} \wedge I) \rightarrow \bigvee_i G_i$ *is valid.*

Condition 2 is crucial for using the resulting nondeterministic control envelope, since it guarantees that safe actions are always available as a fallback.

3.2 An Optimal Solution

Solutions to a synthesis problem may differ in quality. Intuitively, a solution is better than another if it allows for a strictly larger controllable invariant. In case of equality, the solution with the more permissive control envelope wins. Formally, given two solutions $S = (I, G)$ and $S' = (I', G')$, we say that S' is better or equal to S (written $S \sqsubseteq S'$) if and only if $\vDash \mathsf{assum} \rightarrow (I \rightarrow I')$ and additionally either $\vDash \mathsf{assum} \rightarrow \neg(I' \rightarrow I)$ or $\vDash (\mathsf{assum} \wedge I) \rightarrow \bigwedge_i (G_i \rightarrow G'_i)$. Given two solutions S and S', one can define a solution $S \sqcap S' = (I \vee I',\, i \mapsto (I \wedge G_i \vee I' \wedge G'_i))$ that is better or equal to both S and S' ($S \sqsubseteq S \sqcap S'$ and $S' \sqsubseteq S \sqcap S'$). A solution S' is called the *optimal solution* when it is the maximum element in the ordering, so that for any other solution S, $S \sqsubseteq S'$. The optimal solution exists and is expressible in dGL:

$$I^{\mathrm{opt}} \equiv [((\cap_i \mathsf{act}_i)\,;\,\mathsf{plant})^*]\,\mathsf{safe} \tag{3}$$

$$G_i^{\mathrm{opt}} \equiv [\mathsf{act}_i\,;\,\mathsf{plant}]\,I^{\mathrm{opt}}. \tag{4}$$

Intuitively, I^{opt} characterizes the set of all states from which an optimal controller (played here by Demon) can keep the system safe forever. In turn, G^{opt} is defined to allow any control action that is guaranteed to keep the system within I^{opt} until the next control cycle as characterized by a modal formula. Section 3.3 formally establishes the correctness and optimality of $S^{\mathrm{opt}} \equiv (I^{\mathrm{opt}}, G^{\mathrm{opt}})$.

While it is theoretically reassuring that an optimal solution exists that is at least as good as all others and that this optimum can be characterized in dGL, such a solution is of limited practical usefulness since Eq. (3) cannot be executed without solving a game at runtime. Rather, we are interested in *explicit* solutions where I and G are quantifier-free real arithmetic formulas. There is no guarantee in general that such solutions exist that are also optimal, but our goal is to devise an algorithm to find them in the many cases where they exist or find safe approximations otherwise.

3.3 Controllable Invariants

The fact that S^{opt} is a solution can be characterized in logic with the notion of a controllable invariant that, at each of its points, admits some control action that keeps the plant in the invariant for one round. All lemmas and theorems throughout this paper are proved in the extended preprint [21, Appendix B].

Definition 2 (Controllable Invariant). *A* controllable invariant *is a formula* I *such that* $\vDash I \to \mathsf{safe}$ *and* $\vDash I \to \bigvee_i [\mathsf{act}_i ; \mathsf{plant}]\, I$.

From this perspective, I^{opt} can be seen as the largest controllable invariant.

Lemma 1. *I^{opt} is a controllable invariant and it is optimal in the sense that $\vDash I \to I^{\text{opt}}$ for any controllable invariant I.*

Moreover, not just I^{opt}, but *every* controllable invariant induces a solution. Indeed, given a controllable invariant I, we can define $\mathcal{G}(I) \equiv (i \mapsto [\mathsf{act}_i ; \mathsf{plant}]\, I)$ for the *control guards induced by* I. $\mathcal{G}(I)$ chooses as the guard for each action act_i the modal condition ensuring that act_i, preserves I after the plant.

Lemma 2. *If I is a controllable invariant, then $(I, \mathcal{G}(I))$ is a solution (Def. 1).*

Conversely, a controllable invariant can be derived from any solution.

Lemma 3. *If (I, G) is a solution, then $I' \equiv (\mathsf{assum} \wedge I)$ is a controllable invariant. Moreover, we have $(I, G) \sqsubseteq (I', \mathcal{G}(I'))$.*

Solution comparisons w.r.t. $\sqsubseteq$ reduce to implications for controllable invariants.

Lemma 4. *If I and I' are controllable invariants, then $(I, \mathcal{G}(I)) \sqsubseteq (I', \mathcal{G}(I'))$ if and only if $\vDash \mathsf{assum} \to (I \to I')$.*

Taken together, these lemmas allow us to establish the optimality of S^{opt}.

Theorem 1. *S^{opt} is an optimal solution (i.e. a maximum w.r.t. $\sqsubseteq$) of Def. 1.*

This shows the roadmap for the rest of the paper: finding solutions to the control envelope synthesis problem reduces to finding controllable invariants that imply I^{opt}, which can be found by restricting the actions available to Demon in I^{opt} to guarantee safety, thereby *refining* the associated game.

3.4 One-Shot Fallback Refinement

The simplest refinement of I^{opt} is obtained when fixing a single fallback action to use in all states (if that is safe). A more general refinement considers different fallback actions in different states, but still only plays one such action forever.

Using the dGL axioms, any loop-free dGL formula whose ODEs admit solutions expressible in real arithmetic can be automatically reduced to an equivalent first-order arithmetic formula (in $\text{FOL}_{\mathbb{R}}$). An equivalent propositional arithmetic formula in $\mathcal{P}_{\mathbb{R}}$ can be computed via quantifier elimination (QE). For example:

$$
\begin{aligned}
& [(v := 1 \cap v := -1)\,;\, \{x' = v\}]\, x \neq 0 \\
\equiv\ & [v := 1 \cap v := -1]\, [\{x' = v\}]\, x \neq 0 && \text{by } [;] \\
\equiv\ & [v := 1]\, [\{x' = v\}]\, x \neq 0 \ \lor\ [v := -1]\, [\{x' = v\}]\, x \neq 0 && \text{by } [\cap] \\
\equiv\ & [\{x' = 1\}]\, x \neq 0 \ \lor\ [\{x' = -1\}]\, x \neq 0 && \text{by } [:=] \\
\equiv\ & (\forall t{\geq}0\, x + t \neq 0) \lor (\forall t{\geq}0\, x - t \neq 0) && \text{by } [']\text{,}[:=] \\
\equiv\ & x > 0 \ \lor\ x < 0 && \text{by QE .}
\end{aligned}
$$

Even when a formula features nonsolvable ODEs, techniques exist to compute weakest preconditions for differential equations, with conservative approximations [38] or even exactly in some cases [35,8]. In the rest of this section and for most of this paper, we are therefore going to assume the existence of a reduce oracle that takes as an input a loop-free dGL formula and returns a quantifier-free arithmetic formula that is equivalent modulo some assumptions. Section 3.7 shows how to implement and optimize reduce.

Definition 3 (Reduction Oracle). *A* reduction oracle *is a function* reduce *that takes as an input a loop-free* dGL *formula F and an assumption $A \in \mathcal{P}_{\mathbb{R}}$. It returns a formula $R \in \mathcal{P}_{\mathbb{R}}$ along with a boolean flag* exact *such that the formula $A \to (R \to F)$ is valid, and if* exact *is true, then $A \to (R \leftrightarrow F)$ is valid as well.*

Back to our original problem, I^{opt} is not directly reducible since it involves a loop. However, conservative approximations can be computed by restricting the set of strategies that the Demon player is allowed to use. One extreme case allows Demon to only use a single action act_i repeatedly as a fallback (e.g. braking in the train example). In this case, we get a controllable invariant $[(\mathsf{act}_i\,;\,\mathsf{plant})^*]\,\mathsf{safe}$, which further simplifies into $[\mathsf{act}_i\,;\,\mathsf{plant}_\infty]\,\mathsf{safe}$ with

$$\mathsf{plant}_\infty \equiv \{x' = f(x), t' = 1\ \&\ \mathsf{domain}\}$$

a variant of plant that never yields control. For this last step to be valid though, a technical assumption is needed on act_i, which we call *action permanence.*

Definition 4 (Action Permanence). *An action* act_i *is said to be* permanent *if and only if* $(\mathsf{act}_i\,;\,\mathsf{plant}\,;\,\mathsf{act}_i) \equiv (\mathsf{act}_i\,;\,\mathsf{plant})$*, i.e., they are equivalent games.*

Intuitively, an action is *permanent* if executing it more than once in a row has no consequence for the system dynamics. This is true in the common case of actions that only assign constant values to control variables that are read but not modified by the plant, such as $a := A$ and $a := -B$ in Model 1.

Lemma 5. *If* act_i *is permanent,* $\vDash [(\mathsf{act}_i\,;\,\mathsf{plant})^*]\,\mathsf{safe} \leftrightarrow [\mathsf{act}_i\,;\,\mathsf{plant}_\infty]\,\mathsf{safe}$.

Our discussion so far identifies the following approximation to our original synthesis problem, where P denotes the set of all indexes of permanent actions:

$$\begin{aligned} I^0 &\equiv [(\cap_{i\in\mathsf{P}}\,\mathsf{act}_i)\,;\,\mathsf{plant}_\infty]\,\mathsf{safe},\\ G_i^0 &\equiv [\mathsf{act}_i\,;\,\mathsf{plant}]\,I^0. \end{aligned}$$

Here, I^0 encompasses all states from which the agent can guarantee safety indefinitely with a single permanent action. G^0 is constructed according to $\mathcal{G}(I^0)$ and only allows actions that are guaranteed to keep the agent within I^0 until the next control cycle. Note that I^0 degenerates to false in cases where there are no permanent actions, which does not make it less of a controllable invariant.

Theorem 2. I^0 *is a controllable invariant.*

Moreover, in many examples of interest, I^0 and I^{opt} are equivalent since an optimal fallback strategy exists that only involves executing a single action. This is the case in particular for Model 1, where

$$\begin{aligned} I^0 &\equiv [a := -B\,;\,\{p' = v, v' = a\ \&\ v \geq 0\}]\,e - p > 0\\ &\equiv e - p > v^2/2B \end{aligned}$$

characterizes all states at safe braking distance to the obstacle and G^0 associates the following guard to the acceleration action:

$$\begin{aligned} G^0_{a:=A} &\equiv [a := A\,;\,\{p' = v, v' = a, t' = 1\ \&\ v \geq 0 \wedge t \leq T\}]\,e - p > v^2/2B\\ &\equiv e - p > vT + AT^2/2 + (v + AT)^2/2B \end{aligned}$$

That is, accelerating is allowed if doing so is guaranteed to maintain sufficient braking distance until the next control opportunity. Section 3.6 discusses automatic generation of a proof that (I^0, G^0) is an optimal solution for Model 1.

3.5 Bounded Fallback Unrolling Refinement

In Section 3.4, we derived a solution by computing an underapproximation of I^{opt} where the fallback controller (played by Demon) is only allowed to use a one-shot strategy that picks a single action and plays it forever. Although this approximation is always safe and, in many cases of interest, happens to be exact, it does lead to a suboptimal solution in others. In this section, we allow the fallback controller to switch actions a bounded number of times before it plays one forever. There are still cases where doing so is suboptimal (imagine a car on a circular race track that is forced to maintain constant velocity). But this restriction is in line with the typical understanding of a fallback controller, whose mission is not to take over a system indefinitely but rather to maneuver it into a state where it can safely get to a full stop [32].

For all bounds $n \in \mathbb{N}$, we define a game where the fallback controller (played by Demon) takes at most n turns to reach the region I^0 in which safety is guaranteed indefinitely. During each turn, it picks a permanent action and chooses a time θ in advance for when it wishes to play its next move. Because the environment (played by Angel) has control over the duration of each control cycle, the fallback controller cannot expect to be woken up after time θ exactly. However, it can expect to be provided with an opportunity for its next move within the $[\theta, \theta + T]$ time window since the plant can never execute for time greater than T. Formally, we define I^n as follows:

$$I^n \equiv [\mathsf{step}^{\times \leq n}\,;\, \mathsf{forever}]\,\mathsf{safe} \qquad \mathsf{forever} \equiv (\cap_{i \in \mathsf{P}}\, \mathsf{act}_i)\,;\, \mathsf{plant}_\infty$$
$$\mathsf{step} \equiv (\theta := *\,;\, ?\theta \geq 0)^d\,;\, (\cap_{i \in \mathsf{P}}\, \mathsf{act}_i)\,;\, \mathsf{plant}_{\theta+T}\,;\, ?\mathsf{safe}^d\,;\, ?t \geq \theta$$

where $\mathsf{plant}_{\theta+T}$ is the same as plant, except that the domain constraint $t \leq T$ is replaced by $t \leq \theta + T$. Equivalently, we can define I^n by induction as follows:

$$I^{n+1} \equiv I^n \vee [\mathsf{step}]\, I^n \qquad I^0 \equiv [\mathsf{forever}]\,\mathsf{safe}, \tag{5}$$

where the base case coincides with the definition of I^0 in Section 3.4. Importantly, I^n is a loop-free controllable invariant and so reduce can compute an explicit solution to the synthesis problem from I^n.

Theorem 3. *I^n is a controllable invariant for all $n \geq 0$.*

Theorem 3 establishes a nontrivial result since it overcomes the significant gap between the *fantasized* game that defines I^n and the *real* game being played by a time-triggered controller. The proof critically relies on the action permanence assumption along with a result [21, Lemma 6] establishing that ODEs preserve a specific form of reach-avoid property as a result of being deterministic.

Example. As an illustration, consider the example in Fig. 1 and Model 2 of a 2D robot moving in a corridor that forms an angle. The robot is only allowed to move left or down at a constant velocity and must not crash against a wall. Computing I^0 gives us the vertical section of the corridor, in which going down is a safe one-step fallback. Computing I^1 forces us to distinguish two cases. If the corridor is wider than the maximal distance travelled by the robot in a control cycle ($VT > 2R$), then the upper section of the corridor is controllable (with the exception of a dead-end that we prove to be uncontrollable in Section 3.6). On the other hand, if the corridor is too narrow, then I^1 is equivalent to I^0. Formally, we have $I^1 \equiv (y > -R \wedge |x| < R) \vee (VT < 2R \wedge (x > -R \wedge |y| < R))$. Moreover, computing I^2 gives a result that is equivalent to I^1. From this, we can conclude that I^1 is equivalent to I^n for all $n \geq 1$. Intuitively, it is optimal with respect to *any* finite fallback strategy (restricted to permanent actions).

The controllable invariant unrolling I^n has a natural stopping criterion.

Lemma 6. *If $I^n \leftrightarrow I^{n+1}$ is valid for some $n \geq 0$, then $I^n \leftrightarrow I^m$ is valid for all $m \geq n$ and $I^n \leftrightarrow I^\omega$ is valid where $I^\omega \equiv [\mathsf{step}^\times\,;\, \mathsf{forever}]\,\mathsf{safe}$.*

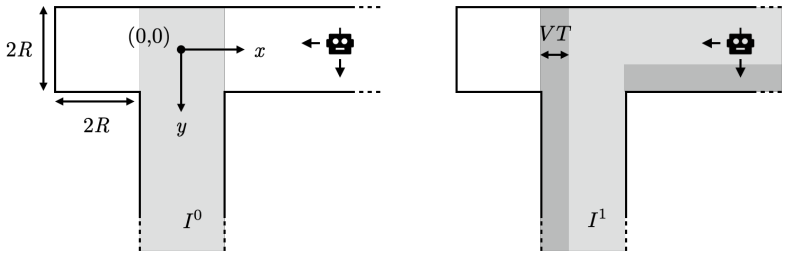

Fig. 1: Robot navigating a corridor (Model 2). A 2D robot must navigate safely within a corridor with a dead-end without crashing against a wall. The corridor extends infinitely on the bottom and on the right. The robot can choose between going left and going down with a constant speed V. The left diagram shows I^0 in gray. The right diagram shows I^1 under the additional assumption $VT < 2R$ (I^1 and I^0 are otherwise equivalent). A darker shade of gray is used for regions of I^1 where only one of the two available actions is safe according to G^1.

Model 2 Robot navigating a corridor with framed solutions of holes.

assum	1	$V > 0 \land T > 0$
ctrlable	2	$\land \boxed{(y > -R \land \lvert x\rvert < R) \lor (VT < 2R \land (x > -R \land \lvert y\rvert < R))} \rightarrow [\{$
ctrl	3	$\quad (\ ?\boxed{x > -R + VT}\,;\, v_x := -V\,;\, v_y := 0)$
	4	$\quad \cup\ (?\boxed{y < R - VT \lor x < R}\,;\, v_x := 0\,;\, v_y := V)\ \ );$
plant	5	$\quad (t := 0\,;\, \{x' = v_x, y' = v_y, t' = 1\ \&\ t \leq T\})$
safe	6	$\}^*]((x > -3R \land \lvert y\rvert < R) \lor (y > -R \land \lvert x\rvert < R))$

3.6 Proving Optimality via the Dual Game

Suppose one found a controllable invariant I using techniques from the previous section. To prove it optimal, one must show that $\vDash \mathsf{assum} \rightarrow (I^{\mathrm{opt}} \rightarrow I)$. By contraposition and $[\alpha]\, P \leftrightarrow \neg\langle\alpha\rangle\, \neg P$ ($[\cdot]$), this is equivalent to proving that:

$$\vDash \mathsf{assum} \land \neg I \rightarrow \underbrace{\langle((\cap_i \mathsf{act}_i)\,;\, \mathsf{plant})^*\rangle\, \neg\mathsf{safe}}_{\neg I^{\mathrm{opt}}}. \tag{6}$$

We define the largest *un*controllable region $U^{\mathrm{opt}} \equiv \neg I^{\mathrm{opt}}$ as the right-hand side of implication 6 above. Intuitively, U^{opt} characterizes the set of all states from which the environment (played by Angel) has a winning strategy against the controller (played by Demon) for reaching an unsafe state. In order to prove the optimality of I, we compute a sequence of increasingly strong approximations U of U^{opt} such that $U \rightarrow U^{\mathrm{opt}}$ is valid. We do so via an iterative process, in the spirit of how we approximate I^{opt} via bounded fallback unrolling (Section 3.5), although the process can be guided by the knowledge of I this time. If at any point we manage to prove that $\mathsf{assum} \rightarrow (I \lor U)$ is valid, then I is optimal.

One natural way to compute increasingly good approximations of U^{opt} is via loop unrolling. The idea is to improve approximation U by adding states from where the environment can reach U by running the control loop once, formally, $\langle(\cap_i \mathsf{act}_i)\,;\,\mathsf{plant}\rangle\, U$. This unrolling principle can be useful. However, it only augments U with new states that can reach U in time T at most. So it cannot alone prove optimality in cases where violating safety from an unsafe state takes an unbounded amount of time.

For concreteness, let us prove the optimality of I^0 in the case of Model 1. In [34] essentially the following statement is proved when arguing for optimality: $\vDash \mathsf{assum} \wedge \neg I^0 \rightarrow \langle(a := -B\,;\,\mathsf{plant})^*\rangle\, \neg\mathsf{safe}$. This is identical to our optimality criterion from Eq. (6), except that Demon's actions are restricted to braking. Intuitively, this restriction is sound since accelerating always makes things worse as far as safety is concerned. If the train cannot be saved with braking alone, adding the option to accelerate will not help a bit. In this work, we propose a method for formalizing such arguments within dGL to arbitrary systems.

Our idea for doing so is to consider a system made of two separate copies of our model. One copy has all actions available whereas the other is only allowed a single action (e.g. braking). Given a safety metric m (i.e. a term m such that $\vDash m \leq 0 \rightarrow \neg\mathsf{safe}$), we can then formalize the idea that "action i is always better w.r.t safety metric m" within this joint system.

Definition 5 (Uniform Action Optimality). *Consider a finite number of discrete* dL *programs α_i and $p \equiv \{x' = f(x)\ \&\ Q\}$. Let $V = \mathrm{BV}(p) \cup \bigcup_i \mathrm{BV}(\alpha_i)$ be the set of all variables written by p or some α_i. For any term θ and integer n, write $\theta^{(n)}$ for the term that results from θ by renaming all variables $v \in V$ to a fresh tagged version $x^{(n)}$. Using a similar notation for programs and formulas, define $p^{(1,2)} \equiv \{(x^{(1)})' = f(x^{(1)}), (x^{(2)})' = f(x^{(2)})\ \&\ Q^{(1)} \wedge Q^{(2)}\}$. We say that action j is* uniformly optimal *with respect to safety metric m if and only if:*

$$\vDash\ m^{(1)} \geq m^{(2)} \rightarrow [{\alpha_j}^{(1)}\,;\,(\cup_i {\alpha_i}^{(2)})\,;\,p^{(1,2)}]\, m^{(1)} \geq m^{(2)}.$$

$\mathsf{best}_j((\alpha_i)_i, p, m)$ denotes that action j is uniformly optimal with respect to m for actions α_i and dynamics p.

With such a concept in hand, we can formally establish the fact that criterion Eq. (6) can be relaxed in the existence of uniformly optimal actions.

Theorem 4. *Consider a finite number of discrete* dL *programs α_i such that $\vDash \langle\alpha_i\rangle\, true$ for all i and $p \equiv \{x' = f(x)\ \&\ q \geq 0\}$. Then, provided that $\mathsf{best}_j((\alpha_i)_i, p, m)$ and $\mathsf{best}_j((\alpha_i)_i, p, -q)$ (no other action stops earlier because of the domain constraint), we have:*

$$\vDash\ \langle((\cap\,\alpha_i)\,;\,p)^*\rangle\, m \leq 0 \leftrightarrow \langle(\alpha_j\,;\,p)^*\rangle\, m \leq 0\ .$$

A general heuristic for leveraging Theorem 4 to grow U automatically works as follows. First, it considers $R \equiv \mathsf{assum} \wedge \neg I \wedge \neg U$ that characterizes states that are not known to be controllable or uncontrollable. Then, it picks a disjunct $\bigwedge_j R_j$ of

the disjunctive normal form of R and computes a forward invariant region V that intersects with it: $V \equiv \bigwedge_j \{R_j : \mathsf{assum},\, R_j \vdash [(\cup_i \mathsf{act}_i)\,;\, \mathsf{plant}]\, R_j\}$. Using V as an assumption to simplify $\neg U$ may suggest metrics to be used with Theorem 4. For example, observing $\vDash V \to (\neg U \to (\theta_1 > 0 \wedge \theta_2 > 0))$ suggests picking metric $m \equiv \min(\theta_1, \theta_2)$ and testing whether $\mathsf{best}_j(\mathsf{act}, p, m)$ is true for some action j. If such a uniformly optimal action exists, then U can be updated as $U \leftarrow U \vee (V \wedge \langle(\mathsf{act}_j\,;\, \mathsf{plant})^*\rangle\, m \leq 0)$. The solution I^1 for the corridor (Model 2) can be proved optimal automatically using this heuristic in combination with loop unrolling.

3.7 Implementing the Reduction Oracle

The CESAR algorithm assumes the existence of a *reduction oracle* that takes as an input a loop-free dGL formula and attempts to compute an equivalent formula within the fragment of propositional arithmetic. When an exact solution cannot be found, an implicant is returned instead and flagged appropriately (Def. 3). This section discusses our implementation of such an oracle.

As discussed in Section 3.4, exact solutions can be computed systematically when all ODEs are solvable by first using the dGL axioms to eliminate modalities and then passing the result to a *quantifier elimination algorithm* for first-order arithmetic [9,42]. Although straightforward in theory, a naïve implementation of this idea hits two practical barriers. First, quantifier elimination is expensive and its cost increases rapidly with formula complexity [11,44]. Second, the output of existing QE implementations can be unnecessarily large and redundant. In iterated calls to the reduction oracle, these problems can compound each other.

To alleviate this issue, our implementation performs *eager simplification* at intermediate stages of computation, between some axiom application and quantifier-elimination steps. This optimization significantly reduces output solution size and allows CESAR to solve a benchmark that would otherwise timeout after 20 minutes in 26s. [21, Appendix E] further discusses the impact of eager simplification. Still, the doubly exponential complexity of quantifier elimination puts a limit on the complexity of problems that CESAR can currently tackle.

In the general case, when ODEs are not solvable, our reduction oracle is still often able to produce *approximate* solutions using differential invariants generated automatically by existing tools [38]. Differential invariants are formulas that stay true throughout the evolution of an ODE system. [4] To see how they apply, consider the case of computing $\mathsf{reduce}([\{x' = f(x)\}]\, P, A)$ where P is the post-condition formula that must be true after executing the differential equation, and A is the assumptions holding true initially. Suppose that formula $D(x)$ is a differential invariant such that $D(x) \to P$ is valid. Then, a precondition sufficient to ensure that P holds after evolution is $A \to D(x)$. For example, to compute the precondition for the dynamics of the `parachute` benchmark, our reduction oracle first uses the Pegasus tool [38] to identify a Darboux polynomial, suggesting

[4] dGL provides ways to reason about differential invariants without solving the corresponding differential equation. For example, for an invariant of the form $e = 0$, the differential invariant axiom is $[\{x' = f(x)\}]\, e = 0 \leftrightarrow (e = 0 \wedge [\{x' = f(x)\}]\, e' = 0)$.

an initial differential invariant D_0. Once we have D_0, the additional information required to conclude post condition P is $D_0 \to P$. To get an invariant formula that implies $D_0 \to P$, eliminate all the changing variables $\{x, v\}$ in the formula $\forall x \forall v\ (D_0 \to P)$, resulting in a formula D_1. D_1 is a differential invariant since it features no variable that is updated by the ODEs. Our reduction oracle returns $D_0 \wedge D_1$, an invariant that entails postcondition P.

3.8 The CESAR Algorithm

The CESAR algorithm for synthesizing control envelopes is summarized in Algorithm 1. It is expressed as a generator that yields a sequence of solutions with associated optimality guarantees. Possible guarantees include "*sound*" (no optimality guarantee, only soundness), "*k-optimal*" (sound and optimal w.r.t all k-switching fallbacks with permanent actions), "*ω-optimal*" (sound and optimal w.r.t all finite fallbacks with permanent actions) and "*optimal*" (sound and equivalent to S^{opt}). Line 11 performs the optimality test described in Section 3.6. Finally, Line 10 performs an important soundness check for the cases where an approximation has been made along the way of computing (I^n, G^n). In such cases, I is not guaranteed to be a controllable invariant and thus Case (2) of Def. 1 must be checked explicitly.

When given a problem with solvable ODEs and provided with a complete QE implementation within `reduce`, CESAR is guaranteed to generate a solution in finite time with an "*n-optimal*" guarantee at least (n being the unrolling limit).

4 Benchmarks and Evaluation

To evaluate our approach to the Control Envelope Synthesis problem, we curate a benchmark suite with diverse optimal control strategies. As Table 2 summarizes, 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 highlight a couple of benchmarks here. See [21, Appendix D] for a discussion of the full suite and the synthesized results, and [20] for the benchmark files and evaluation scripts.

`Power Station` is an example where the optimal control strategy involves two switches, corresponding to two steps of unrolling. A power station can either produce power or dispense it to meet a quota, but never give out more than it has produced. Charging is the fallback action that is safe for all time *after* the station has dispensed enough power. However, to cover all controllable states, we need to switch at least two times, so that the power station has a chance to produce energy and then dispense it, before settling back on the safe fallback. `Parachute` is an example of a benchmark with non-solvable, hyperbolic dynamics. A person jumps off a plane and can make an irreversible choice to open their parachute. The objective is to stay within a maximum speed that is greater than the terminal velocity when the parachute is open.

Algorithm 1 CESAR: Control Envelope Synthesis via Angelic Refinements

Input: a synthesis problem (as defined in Section 3.1), an unrolling limit n.
Remark: valid is defined as $\mathsf{valid}(F,\ A) \equiv (\texttt{first}(\mathsf{reduce}(\neg F, A)) = \text{false})$.
$k \leftarrow 0$
$I, e_I \leftarrow \mathsf{reduce}([\mathsf{forever}]\,\mathsf{safe},\ \mathsf{assum})$
while $k \leq n$ **do**
    $e_G \leftarrow \text{true}$
    for each i **do**
        $G_i, e \leftarrow \mathsf{reduce}([\mathsf{act}_i\,;\,\mathsf{plant}]\,I,\ \mathsf{assum})$
        $e_G \leftarrow e_G$ **and** e
    if $(e_G$ **and** $e_I)$ **or** $\mathsf{valid}(I \rightarrow \bigvee_i G_i,\ \mathsf{assum})$ **then**
        if e_G **and** $\mathsf{optimal}(I)$ **then**
            yield $((I, G)$, "optimal")
            return
        else if e_G **and** e_I **then yield** $((I, G)$, "k-optimal")
        else yield $((I, G)$, "sound")
    $I', e \leftarrow \mathsf{reduce}(I \vee [\mathsf{step}]\,I,\ \mathsf{assum})$
    $e_I \leftarrow e_I$ **and** e
    if e_G **and** e_I **and** $\mathsf{valid}(I' \rightarrow I,\ \mathsf{assum})$ **then**
        yield $((I, G)$, "ω-optimal")
        return
    $I \leftarrow I'$
    $k \leftarrow k + 1$

We implement CESAR in Scala, using Mathematica for simplification and quantifier elimination, and evaluate it on the benchmarks. Simplification is an art [25,23]. We implement additional simplifiers with the `Egg` library [45] and SMT solver `z3` [30]. Experiments were run on a 32GB RAM M2 MacBook Pro machine. CESAR execution times average over 5 runs.

CESAR synthesis is automatic. The optimality tests were computed manually. Table 2 summarizes the result of running CESAR. 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. As an extra step of validation, synthesized solutions are checked by the hybrid system theorem prover KeYmaera X [16]. All solutions are proved correct, with verification time as reported in the last column of Table 2.

5 Related Work

Hybrid controller synthesis has received significant attention [26,41,7], with popular approaches using temporal logic [5,7,46], games [31,43], and CEGIS-like guidance from counterexamples [39,1,37,10]. CESAR, however, solves the different problem of synthesizing control *envelopes* that strive to represent not one but *all* safe controllers of a system. Generating *valid* solutions is not an issue (a trivial solution always exists that has an empty controllable set). The real challenge is *optimality* which imposes a higher order constraint because it reasons

Table 2: Summary of CESAR experimental results

Benchmark	Synthesis Time (s)	Checking Time (s)	Optimal	Needs Unrolling	Non Solvable Dynamics
ETCS Train [34]	14	9	✓		
Sled	20	8	✓		
Intersection	49	44	✓		
Parachute [15]	46	8			✓
Curvebot	26	9			✓
Coolant	49	20	✓	✓	
Corridor	20	8	✓	✓	
Power Station	26	17	✓	✓	

about the relationship between possible valid solutions, and cannot, e.g., fit in the CEGIS quantifier alternation pattern $\exists\forall$. So simply adapting existing controller synthesis techniques does not solve symbolic control envelope synthesis.

Safety shields computed by numerical methods [2,13,24] 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.

On the optimality side, a systematic but manual process was used to design a safe European Train Control System (ETCS) and justify it as optimal with respect to specific train criteria [34]. Our work provides the formal argument filling the gap between such case-specific criteria and end-to-end optimality. CESAR is more general and automatic.

6 Conclusion

This paper presents the CESAR algorithm for Control Envelope Synthesis via Angelic Refinements. It is the first approach to automatically synthesize symbolic control envelopes for hybrid systems. The synthesis problem and optimal solution are characterized in differential game logic. Through successive refinements, the optimal solution in game logic is translated into a controllable invariant and control conditions. The translation preserves safety. For the many cases where refinement additionally preserves optimality, an algorithm to test optimality of the result post translation is presented. The synthesis experiments on a benchmark suite of diverse control problems demonstrate CESAR's versatility. For future work, we plan to extend to additional control shapes, and to exploit the synthesized safe control envelopes for reinforcement learning.

References

1. Abate, A., Bessa, I., Cordeiro, L.C., David, C., Kesseli, P., Kroening, D., Polgreen, E.: 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
2. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. Proceedings of the Aaai Conference on Artificial Intelligence **32** (2018). doi: 10.1609/aaai.v32i1.11797
3. Alur, R.: Principles of Cyber-Physical Systems. MIT Press, Cambridge (2015)
4. Ames, A.D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., Tabuada, P.: Control barrier functions: Theory and applications. In: 17th European Control Conference, ECC 2019, Naples, Italy, June 25-28, 2019. pp. 3420–3431. IEEE (2019). doi: 10.23919/ECC.2019.8796030
5. Antoniotti, M., Mishra, B.: Discrete event models+temporal logic=supervisory controller: automatic synthesis of locomotion controllers. In: Proceedings of 1995 IEEE International Conference on Robotics and Automation. vol. 2, pp. 1441–1446 vol.2 (1995). doi: 10.1109/ROBOT.1995.525480
6. Basile, G., Marro, G.: Controlled and conditioned invariant subspaces in linear system theory. Journal of Optimization Theory and Applications **3**, 306–315 (05 1969). doi: 10.1007/BF00931370
7. Belta, C., Yordanov, B., Gol, E.A.: Formal Methods for Discrete-Time Dynamical Systems. Springer Cham (2017)
8. Boreale, M.: Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial ODE's. In: Tjoa, A.M., Bellatreche, L., Biffl, S., van Leeuwen, J., Wiedermann, J. (eds.) SOFSEM 2018: Theory and Practice of Computer Science - 44th International Conference on Current Trends in Theory and Practice of Computer Science, Krems, Austria, January 29 - February 2, 2018, Proceedings. LNCS, vol. 10706, pp. 442–455. Springer (2018)
9. Caviness, B.F., Johnson, J.R.: Quantifier elimination and cylindrical algebraic decomposition. Springer Science & Business Media (2012)
10. Dai, H., Landry, B., Pavone, M., Tedrake, R.: Counter-example guided synthesis of neural network lyapunov functions for piecewise linear systems. 2020 59th IEEE Conference on Decision and Control (CDC) pp. 1274–1281 (2020)
11. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. **5**(1/2), 29–35 (1988)
12. Doyen, L., Frehse, G., Pappas, G.J., Platzer, A.: Verification of hybrid systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 1047–1110. Springer, Cham (2018). doi: 10.1007/978-3-319-10575-8_30
13. Fisac, J., Akametalu, A., Zeilinger, M., Kaynama, S., Gillula, J., Tomlin, C.: 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
14. Freiberger, F., Schupp, S., Hermanns, H., Ábrahám, E.: Controller verification meets controller code: A case study. In: Proceedings of the 19th ACM-IEEE International Conference on Formal Methods and Models for System Design. p. 98–103. MEMOCODE '21, Association for Computing Machinery, New York, NY, USA (2021). doi: 10.1145/3487212.3487337

15. Fulton, N., Mitsch, S., Bohrer, R., Platzer, A.: Bellerophon: Tactical theorem proving for hybrid systems. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) ITP. LNCS, vol. 10499, pp. 207–224. Springer (2017). doi: 10.1007/978-3-319-66107-0_14
16. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: CADE. pp. 527–538 (2015). doi: 10.1007/978-3-319-21401-6_36
17. Fulton, N., Platzer, A.: 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)
18. Ghosh, B.K.: Controlled invariant and feedback controlled invariant subspaces in the design of a generalized dynamical system. In: 1985 24th IEEE Conference on Decision and Control. pp. 872–873 (1985). doi: 10.1109/CDC.1985.268620
19. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Case study: Verifying the safety of an autonomous racing car with a neural network controller. In: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control. HSCC '20, Association for Computing Machinery, New York, NY, USA (2020). doi: 10.1145/3365365.3382216
20. Kabra, A., Laurent, J., Mitsch, S., Platzer, A.: Control Envelope Synthesis via Angelic Refinements (CESAR): Artifact (1 2024). doi: 10.6084/m9.figshare.24922896.v1, https://figshare.com/articles/software/Control_Envelope_Synthesis_via_Angelic_Refinements_CESAR_Artifact/24922896
21. Kabra, A., Laurent, J., Mitsch, S., Platzer, A.: Cesar: Control envelope synthesis via angelic refinements (2023). doi: https://doi.org/10.48550/arXiv.2311.02833, arXiv:2311.02833
22. Kabra, A., Mitsch, S., Platzer, A.: 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
23. Knuth, D.E.: The Art of Computer Programming. Addison Wesley Longman Publishing Co., Inc., USA (1997)
24. Kochenderfer, M.J., Holland, J.E., Chryssanthacopoulos, J.P.: Next generation airborne collision avoidance system. Lincoln Laboratory Journal **19**(1), 17–33 (2012)
25. Lara, M., López, R., Pérez, I., San-Juan, J.F.: Exploring the long-term dynamics of perturbed keplerian motion in high degree potential fields. Communications in Nonlinear Science and Numerical Simulation **82**, 105053 (2020). doi: https://doi.org/10.1016/j.cnsns.2019.105053, https://www.sciencedirect.com/science/article/pii/S1007570419303727
26. Liu, S., Trivedi, A., Yin, X., Zamani, M.: Secure-by-construction synthesis of cyber-physical systems. Annual Reviews in Control **53**, 30–50 (2022). doi: https://doi.org/10.1016/j.arcontrol.2022.03.004
27. Lunze, J., Lamnabhi-Lagarrigue, F. (eds.): Handbook of Hybrid Systems Control: Theory, Tools, Applications. Cambridge Univ. Press, Cambridge (2009). doi: 10.1017/CBO9780511807930
28. Mitra, S.: Verifying Cyber-Physical Systems: A Path to Safe Autonomy. MIT Press (2021)
29. Mitsch, S., Platzer, A.: 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

30. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
31. Nerode, A., Yakhnis, A.: Modelling hybrid systems as games. In: Decision and Control, 1992., Proceedings of the 31st IEEE Conference on. pp. 2947–2952 vol.3 (1992). doi: 10.1109/CDC.1992.371272
32. Pek, C., Althoff, M.: Fail-safe motion planning for online verification of autonomous vehicles using convex optimization. IEEE Transactions on Robotics **37**(3), 798–814 (2020)
33. Platzer, A.: Logical Foundations of Cyber-Physical Systems. Springer, Cham (2018). doi: 10.1007/978-3-319-63588-0
34. Platzer, A., Quesel, J.: 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. pp. 246–265 (2009). doi: 10.1007/978-3-642-10373-5_13
35. Platzer, A., Tan, Y.K.: Differential equation invariance axiomatization. Journal of the ACM (JACM) **67**(1), 1–66 (2020)
36. Qian, M., Mitsch, S.: Reward shaping from hybrid systems models in reinforcement learning. In: Rozier, K.Y., Chaudhuri, S. (eds.) NFM. LNCS, vol. 13903. Springer (2023)
37. Ravanbakhsh, H., Sankaranarayanan, S.: 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. pp. 8:1–8:10 (2016). doi: 10.1145/2968478.2968485
38. Sogokon, A., Mitsch, S., Tan, Y.K., Cordwell, K., Platzer, A.: Pegasus: Sound continuous invariant generation. Form. Methods Syst. Des. **58**(1), 5–41 (2022). doi: 10.1007/s10703-020-00355-z, special issue for selected papers from FM'19
39. Solar-Lezama, A.: Program sketching. STTT **15**(5-6), 475–495 (2013). doi: 10.1007/s10009-012-0249-7
40. Squires, E., Pierpaoli, P., Egerstedt, M.: Constructive barrier certificates with applications to fixed-wing aircraft collision avoidance. In: 2018 IEEE Conference on Control Technology and Applications (CCTA). pp. 1656–1661 (2018). doi: 10.1109/CCTA.2018.8511342
41. Tabuada, P.: Verification and Control of Hybrid Systems: A Symbolic Approach. Springer, Berlin (2009). doi: 10.1007/978-1-4419-0224-5
42. Tarski, A.: A decision method for elementary algebra and geometry. In: Caviness, B.F., Johnson, J.R. (eds.) Quantifier Elimination and Cylindrical Algebraic Decomposition. pp. 24–84. Springer Vienna, Vienna (1998)
43. Tomlin, C.J., Lygeros, J., Sastry, S.: A game theoretic approach to controller design for hybrid systems. Proc. IEEE **88**(7), 949–970 (2000). doi: 10.1109/5.871303
44. Weispfenning, V.: The complexity of linear problems in fields. J. Symb. Comput. **5**(1-2), 3–27 (1988)
45. Willsey, M., Nandi, C., Wang, Y.R., Flatt, O., Tatlock, Z., Panchekha, P.: Egg: Fast and extensible equality saturation. Proc. ACM Program. Lang. **5**(POPL) (jan 2021). doi: 10.1145/3434304, https://doi.org/10.1145/3434304
46. Yang, S., Yin, X., Li, S., Zamani, M.: Secure-by-construction optimal path planning for linear temporal logic tasks. In: 2020 59th IEEE Conference on Decision and Control (CDC). pp. 4460–4466 (2020). doi: 10.1109/CDC42340.2020.9304153