LLM-Powered Automatic Theorem Proving and Synthesis for Hybrid Systems and Games

Aditi Kabra*, Jonathan Laurent†, Ruben Martins*, Stefan Mitsch‡, André Platzer†
*Carnegie Mellon University, Pittsburgh, USA
{akabra, rubenm}@cs.cmu.edu
†Karlsruhe Institute of Technology, Karlsruhe, Germany
{jonathan.laurent, platzer}@kit.edu
‡DePaul University, Chicago, USA
smitsch@depaul.edu

***Abstract*—Hybrid games model cyber-physical systems (CPS), like cars, trains, and airplanes, where discrete control decisions interact with continuous physical dynamics. We use Large Language Models (LLMs) to scale formal verification and synthesis for hybrid systems and games in a high-level symbolic logic, differential game logic (dGL). This combination of a logic with the right expressivity and automation of the interactive theorem proving process using LLMs brings within reach a challenging class of CPS verification/synthesis problems that were previously well out of range of automatic theorem proving. We demonstrate it on five challenging case studies, all beyond the reach of existing automatic techniques. Verification succeeds for all five, and the synthesis of control solutions succeeds for four of the five.**

***Index Terms*—hybrid systems, synthesis, verification, large language models**

I. Introduction

Systems like cars, trains, and airplanes, where discrete control software interacts with continuous physical dynamics, are modeled by *hybrid systems* and *hybrid games*. Their verification and formally justified synthesis are important because they are often safety-critical, but mathematically complex, and thus difficult to get exactly right. As a consequence, extensive research has gone into developing formal methods for hybrid systems and games, with different approaches making different tradeoffs. Some tools gain impressive scalability by limiting the systems that they handle to mathematically simpler dynamics such as piecewise constant [1] or affine [2], [3]. Other tools very effectively handle nonlinear dynamics over short time horizons [4], [5] using numerical techniques, but lose precision of prediction over longer time horizons, struggling when time is unbounded. The general trend is that scalability is gained at the cost of expressiveness and precision. On the most expressive end are deductive, symbolic techniques in high-level logics [6], [7], [8], [9]. They accept nonlinear dynamics interacting with discrete control in rich, program-like structures, permit reasoning in terms of *symbolic parameters* that make theorems adaptable and reusable, and cleanly handle infinite state space as well as unbounded domains including unbounded time horizons, all with precise reasoning free of discretization or approximation. However, they have one big disadvantage: despite efforts towards automation [10], [11], they generally require effort-intensive interactive theorem proving by an expert[1], which becomes a bottleneck for the complexity of systems handled and broader adoption.

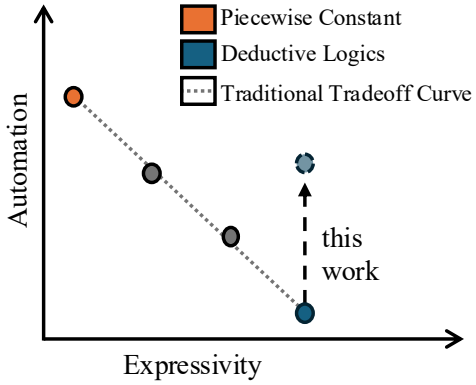


Fig. 1: Schematic representation of tradeoffs in formal methods applied to Hybrid Systems.

LLMs have recently shown promise in automating (and therefore scaling) interactive theorem proving for mathematics and (purely discrete) software [15], [16], [17]. This paper explores the same premise for the data-scarce, mathematically complex domain of hybrid systems and games. We identify five challenging case studies in verification and control synthesis and evaluate the ability of LLMs to prove and synthesize for them. The findings enable a change in the tradeoff landscape of formal methods for hybrid systems, as the application of LLMs brings unprecedented automation to symbolic, deductive verification and synthesis, illustrated schematically in Fig. 1.

We create a pipeline where the LLM interacts with the interactive theorem prover KeYmaera X [7] over multiple rounds, analyzing the problem and fixing its own mistakes. With this verification engine, downstream applications of verification can be automated. We target the problem of soundly *synthesizing* the constraints characterizing the space of

[1]Hybrid systems are fundamentally undecidable, and reasoning about real arithmetic is expensive: polynomial arithmetic is doubly exponential [12]. Still, using the deductive proving system of differential dynamic logic (dL), it is possible to prove most theorems since dL is complete for semialgebraic invariants [13] and for open properties of compact initial value problems [14].

https://doi.org/

correct control for a given hybrid game and control objective. These control constraints are represented as *symbolic control envelopes*, inducing families of controllers that all verifiably meet the control objectives[2]. A *symbolic* control envelope is parametric in symbols in its input, e.g., a train control solution in terms of symbolic train weight w that can be any real number. We use LLMs and the automatic theorem proving system strategically within a formal envelope synthesis framework [18], [19] to automatically synthesize verified *symbolic control envelopes*, applying LLMs to this problem for the first time. As a consequence, we solve problems out of the reach of existing tools, and support, for the first time, *informal descriptions* of the desired control solution, for which our tool automatically identifies mathematical symbolic control constraints.

The tool verifies all five and synthesizes four of the five case studies, none of which is solved by existing automatic tools. Our key contributions are the *creation* and *evaluation* of an *automatic theorem proving system* and a *synthesis pipeline* that scale symbolic logic based verification and synthesis for hybrid systems and games, *bringing a new class of problems within the reach of automatic verification and computer-aided design*. We *propose challenging case studies* and analyze performance on them. Our system, including the case studies and the experiments reported in the paper (LLM responses cached and re-playable) is available online at https://doi.org/10.1184/R1/32248389 [20], together with a supplementary website at https://aditink.github.io/llm-powered-atp-and-synthesis.

The remainder of this paper is organized as follows. Section II discusses related work, and Section III introduces our five case studies. Section IV presents the automated theorem-proving pipeline and its quantitative results, while Section V analyzes the resulting proofs and common failure modes. Section VI presents the control-synthesis pipeline and its evaluation, and Section VII concludes.

II. Related Work

LLMs have been applied to formal theorem proving through several forms of interaction with proof assistants. Prior work uses language models to generate proof steps [15], combines generation with premise retrieval and proof search [21], and generates complete proofs followed by repair using prover feedback [22]. Agentic systems further combine informal reasoning, recursive decomposition, formal proof generation, and verifier feedback [17]. Most closely related to our verification pipeline, COPRA [23] uses a general-purpose LLM with a stateful backtracking loop, repeatedly proposing tactics in Lean and Rocq and using feedback from the proof environment to guide subsequent attempts.

LLMs have also been explored for software verification, including the generation of machine-checkable proofs for system software [16]. Verification-aware languages such as Verus [24] and Dafny [25] provide environments for checking implementations, specifications, and proof annotations.

The dGL/KeYmaera X setting has relatively few publicly available proofs, tactics, or synthesis examples compared with widely studied theorem-proving and software-verification settings. Our verification pipeline shares the broad prover-in-the-loop architecture of prior work, but addresses proof obligations that combine adversarial hybrid-game structure, nonlinear differential equations, loop and differential invariants, real arithmetic, and Bellerophon tactic construction. We support this reasoning with an initial game-analysis step and domain-specific guidance for dGL. To our knowledge, this is the first work to explore LLM-based verification and synthesis for hybrid systems and games using a deductive, high-level symbolic logic like differential game logic [26]. Unlike works focused on LLM-based *autoformalization* for hybrid systems and games [27], [28], this paper studies the separate problem of *verification and synthesis assuming correct specifications*.

Theorem provers for hybrid systems traditionally focus on a mix of interactive proof techniques and proof heuristics for automation [29]. HHL Prover [9] and HHLPy [30] use proof scripts with annotated invariants for loops and differential equations [29]. IsaVODEs [31] uses scripted proofs with automation provided by Isabelle [32]. Plaidypvs [33] embeds differential dynamic logic in PVS. KeYmaera X [7] selects proof tactics based on the syntactic shape of proof obligations; proof heuristics provide a manually selected priority between tactic alternatives, attempted in order with timeouts. These proof heuristics achieve automation in hybrid systems, nonlinear continuous systems, and hybrid games, outperforming even proof scripts of other hybrid systems theorem provers [34], [29]. This paper introduces a reasoning pipeline to push automation beyond the reach of KeYmaera X heuristics.

III. Case Studies

Five case studies are carefully developed to pose *challenging verification and synthesis problems*. None of them are solved automatically by existing tools alone. The synthesis variants of the problems consist of *hybrid games* defining the control options and environment behaviors that are physically possible, the formal desired control objectives, and *informal descriptions* of the desired control behavior within the many possibilities. The synthesis question consists of identifying the exact symbolic constraints that characterize which control action can be taken when. Multiple control decisions may be permitted at any given time, and following different decisions leads to different controllers, each of which is correct. Thus a synthesized solution identifies a *set* of control solutions, each of which is guaranteed to be correct. In the *verification* variants of the problems, the input is the same hybrid game, restricted to a set of (uncountably infinitely many) initial states for which

[2]The reason for seeking *families* or *sets* of controllers is, when developing a formally verified controller, (e.g., a train control system) one generally has to consider primary, mission-critical requirements, (e.g., safety) along with secondary or evolving requirements (e.g., smoothness of the ride). An approach to synthesizing a controller that is formally correct with respect to the primary requirements is: (1) first identify a maximal *set* of controllers formally verified to satisfy the primary requirement, i.e., a *verified control envelope* for the requirement, and then (2) optimize within this set for the controller choice that best meets the secondary requirements.

we think there exists a correct solution. The task is to *prove* that there exists a control solution that works for all of these initial states using the theorem prover KeYmaera X. These case studies could serve as the starting point for a benchmark suite for hybrid systems and games verification and synthesis.

1) Lotka-Volterra: The first problem (Model 1), which serves as a running example, is about maintaining safe populations under the *nonlinear* predator-prey *Lotka-Volterra* model given the ability to introduce additional predators and prey at unpredictable intervals of *unbounded duration*. Section V-A presents this example in further detail, demonstrating how a combination of high-level understanding of the dynamical behavior of the problem as well as low-level knowledge of the dGL calculus is required to solve it.

2) Train: The second case study, *Train*, is about verifying the ability of a *train* control system to maintain safety over train dynamics with *Davis resistance (nonlinear) and time-dependent airbrakes*. This case study poses structural complexity with a *nested inner loop* over *different modes* of air brake operation. The structural complexity demands a long proof tactic during verification, and the generation of multiple interdependent control constraints/invariants during synthesis.

3) Chemical Reaction: The third problem, *Chemical Reaction*, is about maintaining safe temperatures during a *chemical reaction*. This system again requires reasoning about complex dynamics and different, arbitrarily alternating control modes over unbounded iterations. The solution to the dynamics lies *outside the decidable (polynomial) fragment of real arithmetic*, requiring non-trivial *deductive reasoning about differential equations via invariants*.

4) Coolant: The fourth problem, *Coolant*, is about the ability of a *coolant* system to meet heat absorption quotas in a nuclear power plant while limiting coolant discharge, with a *complex, timed, reach-avoid objective*, different discrete modes, and dynamics that behave very differently depending on initial conditions of variables. Anecdotally, it took a human expert three days to manually prove this case study.

5) Van der Pol: The fifth problem, *Van der Pol*, is about choosing the right initial conditions for a *Van der Pol* oscillator to maintain a safety property, which again requires reasoning about *nonlinear dynamics* over unbounded iterations, and therefore, effectively, *unbounded time*. The *chaotic* nature of the Van der Pol oscillator dynamics makes the *choice of initial conditions* delicate.

The artifact [20] presents these case studies in detail.

IV. Automated Theorem Proving

We design an automated theorem proving (ATP) framework that uses LLMs to prove hybrid game/systems theorems in KeYmaera X [7]. A strength of our framework is its generality: we design the pipeline and prompts to focus on generic information about all parts of dGL [26], and the pipeline works across all five case studies without any case-study-specific engineering.

A. Background

Hybrid *systems* extend *discrete* software by also modeling *continuous* dynamics which allows them to model control systems governed by discrete code in a continuous physical environment. Hybrid *games* extend systems by adding nondeterministic choice points resolved *adversarially* by players. They can be used to specify the *space* of control within which a controller player must find a strategy to achieve its control objectives despite potentially adversarial environmental conditions, e.g., a car that must reach the destination while remaining safe despite other cars driving aggressively. In this paper, hybrid games are represented in differential game logic (dGL) [26]. Differential dynamic logic (dL) [35] is the single-player subset of dGL that models hybrid *systems*. dGL can be read like an imperative programming language. This paper does not require a detailed understanding of dGL, but Appendix B provides a brief overview.

B. Pipeline Description

Our automated theorem proving system follows the pipeline in Fig. 2. It uses the natural design of a conversational agent that iteratively proposes tactics that KeYmaera X checks. With access to the full conversation history, it corrects mistakes based on KeYmaera X output. To make the LLM aware of the high-level meaning of the theorem, we add an initial problem analysis step.

In the *Analyze Game* step, given information about the syntax of dGL, the LLM produces an analysis of the input formula that contains three parts: what actions are controlled by which player, what modes (regions of different qualitative behavior) the game has, and comments about the overall game play pattern. Then the proving loop begins. This consists of two steps repeated until the proof is complete. First, the LLM is prompted to propose a tactic. The prompt includes a *detailed guide on writing tactics in Bellerophon*, the tactic language of KeYmaera X [36]. The LLM is given the formula to prove, and a record of past proof attempts. The second step is to run the proposed tactic in KeYmaera X. If the tactic completes the proof, the pipeline ends successfully. Otherwise, KeYmaera X produces some output indicating where the proof failed. This output feeds back into the next tactic proposal step, continuing the loop. The artifact [20] contains the detailed prompts for each query. The pipeline is implemented using the Delphyne framework [37] for oracular programming with LLMs.

A useful variant of this pipeline is one where after each iteration of the proving loop, the LLM is given a chance to reflect and decide if in fact the original theorem it is trying to prove is not valid. This variant is a useful component in our synthesis system later in the pipeline, where it helps identify and repair incorrect control conditions. Additionally, when it is important to limit input context length, making long histories of proof attempts impractical, a variant of the pipeline summarizes the proof state, mistakes so far, and global strategy after each iteration of the proving loop.

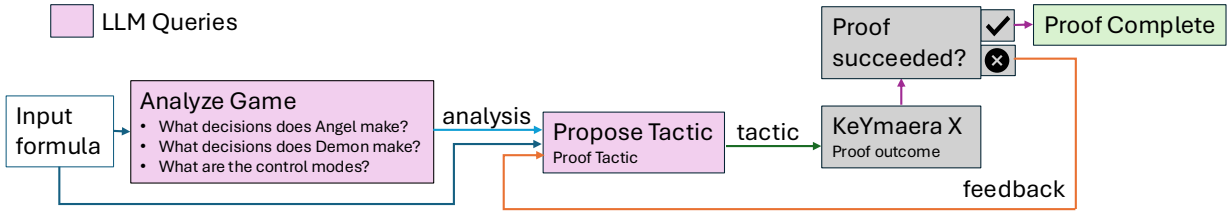


Fig. 2: Automated Theorem Proving Pipeline. The artifact [20] shows prompts.

C. Experimental Results

Table I summarizes the outcomes of running the tool. We try multiple models, running three times on each case study (except for GPT-5.5 with no reasoning, where we run only once per benchmark because of high dollar costs for failure runs[3]). We place an output limit of three hundred thousand tokens; successful runs are generally much shorter than this, and a failed run occurs when the tool fails to find a proof within this token limit. A case study is *solved* when at least one run finds a proof. *Pass@1* measures the average, over case studies, of the probability that a single run succeeds. None of the case studies can be solved by KeYmaera X's existing automation alone [10].

GPT-5.5 with high reasoning effort achieves the best pass@1 over all tested configurations, succeeding in all runs. GPT-5.4 with high reasoning effort also solves all problems, albeit with some failed runs. GPT-5.4-mini, a much smaller model, with high reasoning effort solves only 2/5 case studies. GPT-5.4-mini without reasoning, as well as Qwen3.5 32B A3B with thinking enabled, run locally, solve 1/5 case studies. The pipeline for the Qwen experiment is modified to include a summarization step after each iteration of the proving loop instead of retaining the entire conversation history in order to avoid exceeding the input context window. Without reasoning, models perform substantially worse. Table II shows a detailed per-benchmark breakdown of the results.

The results suggest that our tool is general relative to the LLM used. As smaller and open-source models catch up in capability with large frontier models, the tool will become easier and cheaper to use. Section V discusses the successes and failures in-depth.

V. Analysis of Verification Results

We next discuss the results in depth. Section V-A first sets up the background knowledge and context for this discussion by walking through an example.

A. Verification Impact by Example

This section illustrates how the combination of a symbolic logic with the right level of expressivity and LLMs to automate reasoning resolves challenges beyond the reach of existing automatic techniques. For clarity, we focus the explanation on one case study, but the style of challenges and their resolutions generalize to all case studies and most dGL theorems.

1) Case Study Lotka-Volterra: We introduce the first case study as a running example. A forest department must maintain predator population `y` and prey population `x` above minimum thresholds y_{mn} and x_{mn} respectively, for arbitrarily long. Because of unpredictable funding cycles, they can introduce additional prey and predators at unpredictable intervals, that could last arbitrarily long. This case study is inspired by an ARCH-COMP benchmark [38], but is modified by the addition of discrete control choices and adversarial game play.

2) Model 1 Explanation: Model 1 shows the dGL formula modeling this problem via a *game*, with two players, canonically called Angel (representing the forest department) and Demon (representing uncertainties)[4]. Angel chooses the number of animals to introduce (Lines 4, 5 and 6)[5], while Demon adversarially chooses the length of the interval until the next opportunity to introduce animals (Line 7), as well as how many iterations the game runs for (Line 8), forcing Angel to maintain safe populations for arbitrarily long[6]. Angel wins if in the end populations remain above the minimum thresholds (Line 8). The overall modal formula of the form $\langle\alpha\rangle\phi$ is true in states where Angel has a strategy to play in α so that regardless of how Demon plays in α, she will win (i.e., postcondition ϕ holds). Model 1 asks us to prove that, under the assumptions of Lines 1, 2, and 3, Angel can always win.

This case study is symbolic, with different behavior depending on the values of the parameters a, b, d, g, and variables `x` and `y`. Our reasoning must account for all uncountably infinite possible values that these symbols can take. To verify this formula, we will necessarily have to reason about the *nonlinear dynamics of the Lotka-Volterra equation over unbounded time horizons* (Line 7). While only a few automatic solvers can reason about nonlinear dynamics, even these cannot reason about reachability over unbounded time

[3]An observation that may not be immediately apparent from Table I is that failed runs that exhaust the token budget can be very costly. For example, although the successful runs of GPT-5.5 without reasoning cost only $17.85, the total cost of all five runs was $127.03 because the failed runs cost $63.43 and $45.75 (with the difference between these two caused by different input token counts).

[4]We use the typographical convention of monospaced font for `variables` that change over the course of a game and math font for each *symbolic parameter* P that can represent any real number which remains constant over the course of a game.

[5]In the game `x_add` $:= *$, Angel freely chooses any real number to assign to `x_add`. Then, ?`x_add` ≥ 0 checks that `x_add` must be non-negative.

[6]In the game $\{x' = f(x)\}^d$, the state evolves per the flow of the differential equation $x' = f(x)$ for a duration chosen by Demon. In the game $(\alpha)^\times$, Demon chooses how many times to repeat the game α.

Model	Reasoning	Solved	Pass@1	Avg Cost ($)	Avg Calls	Avg Output (k Tokens)
Qwen3.5	✓	1/5	0.13 ± 0.13	—	88.0 ± 18.0	254 ± 17
GPT-5.4-mini		1/5	0.13 ± 0.13	1.07 ± 1.05	138.5 ± 130.5	41 ± 39
GPT-5.4-mini	✓	2/5	0.33 ± 0.21	0.70 ± 0.17	11.0 ± 2.7	148 ± 36
GPT-5.4	✓	5/5	0.93 ± 0.07	2.21 ± 0.44	9.4 ± 1.4	92 ± 18
GPT-5.5		3/5	0.60 ± 0.24	5.95 ± 5.25	47.3 ± 33.7	35 ± 30
GPT-5.5	✓	**5/5**	$\mathbf{1.00 \pm 0.00}$	$\mathbf{2.05 \pm 0.56}$	$\mathbf{6.1 \pm 1.3}$	$\mathbf{63 \pm 17}$

TABLE I: Verification results across models. *Reasoning* indicates whether `reasoning_effort` is set to `high` or `none` (OpenAI), or enabled (Qwen). *Solved* indicates the number of case studies where at least one run succeeded. *Pass@1* is the mean, over case studies, of the empirical success rate of a single run $\pm$ standard error across case studies. Cost, calls, and output tokens are averaged *only over successful runs* $\pm$ standard error. Output token counts are reported in thousands. As Qwen3.5 runs are local, they do not have costs.

Configuration	Chem. Reaction		Coolant		Train		Lotka-Volterra		Van der Pol	
	Succ.	Cost	Succ.	Cost	Succ.	Cost	Succ.	Cost	Succ.	Cost
Qwen3.5, reas.	2/3	—	0/3	—	0/3	—	0/3	—	0/3	—
GPT-5.4-mini, no reas.	2/3	1.07	0/3	—	0/3	—	0/3	—	0/3	—
GPT-5.4-mini, reas.	2/3	0.33	0/3	—	0/3	—	0/3	—	**3/3**	**0.95**
GPT-5.4, reas.	3/3	0.81	3/3	2.09	**3/3**	**1.22**	**3/3**	**4.38**	2/3	2.74
GPT-5.5, no reas.	**1/1**	**0.17**	0/1	—	1/1	16.44	0/1	—	1/1	1.24
GPT-5.5, reas.	3/3	0.88	**3/3**	**1.80**	3/3	1.89	3/3	4.68	3/3	1.01

TABLE II: Per-benchmark verification results. *Succ.*: successful runs out of total runs. *Cost*: average dollar cost of *successful runs*.

Model 1 Lotka-Volterra Population Control. Gray text is part of the verification problem but is excluded during synthesis, where it is synthesized automatically.

assume	1	$\mathtt{x} \leq \frac{g}{d} \wedge \mathtt{y} \leq \frac{a}{b} \wedge \frac{g}{d} \geq x_{\text{mn}} \wedge \frac{a}{b} \geq y_{\text{mn}} \wedge$
	2	$\mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge a > 0 \wedge b > 0 \wedge$
	3	$d > 0 \wedge g > 0 \wedge \mathtt{x} \geq x_{\text{mn}} \wedge \mathtt{y} \geq y_{\text{mn}} \rightarrow$
ctrl	4	$\langle ($ $\mathtt{x\_add} := *; ?\mathtt{x\_add} \geq 0;$
	5	$\mathtt{y\_add} := *; ?\mathtt{y\_add} \geq 0;$
	6	$\mathtt{x} := \mathtt{x} + \mathtt{x\_add};\ \mathtt{y} := \mathtt{y} + \mathtt{y\_add};$
plant	7	$\{\mathtt{x}' = a\mathtt{x} - b\mathtt{xy},\ \mathtt{y}' = d\mathtt{xy} - g\mathtt{y}\}^d$
contract	8	$)^\times \rangle (\mathtt{x} \geq x_{\text{mn}} \wedge \mathtt{y} \geq y_{\text{mn}})$

horizons[7]. Then we must consider the discrete and adversarial dynamics: Angel chooses *some* non-negative real number of animals to introduce (Line 4 and Line 5) and we must verify whether *any* of her uncountably infinite choices will work over the nonlinear, unbounded-time dynamics. Additionally, we must consider what happens if this entire process repeats *arbitrarily many times* in a loop (Line 8). This explosion in possible behaviors is hard for traditional automatic techniques to deal with.

Yet a symbolic proof is possible by translating a high-level physical understanding of the problem to a clever choice of deductive proof rules. Looking closely at the qualitative behavior of the Lotka-Volterra dynamics reveals a way for Angel to win. The dynamics have an *equilibrium point* at $(\mathtt{x}, \mathtt{y}) = (g/d, a/b)$, which is above the minimum populations threshold per assumptions $a/b \geq y_{\text{mn}}$ and $g/d \geq x_{\text{mn}}$ (Model 1, Line 1). If Angel manages to set the populations to this equilibrium point at the start of each iteration, then she will satisfy her winning condition at the end of the iteration, regardless of how long Demon runs the dynamics. Since Angel can introduce any non-negative number of animals, all she needs at the start of each iteration is that predator and prey populations are no greater than the equilibrium point. Finally, many iterations of this maneuver string together correctly over loop iterations because at the end of each iteration, populations are at the equilibrium point, satisfying the requirement that populations are no greater than the equilibrium point.

To translate this high-level physical reasoning to a formal proof, the right choice of dGL proof rules[8] is critical, and traditionally provided by a human expert who understands what proof rule is applicable when. To show the first step in our reasoning above, that $(\mathtt{x}, \mathtt{y}) = (g/d, a/b)$ is an equilibrium point of the dynamics, the rule most commonly used to prove invariants of differential equations (the *differential invariant* rule) does not immediately apply. Instead, a quick proof is possible using the more rarely used *differential radical invariant* [39], [13] rule (Eq. (1)). It allows us to conclude that polynomial equation $h_j = 0$ is an invariant of the dynamics if we can show that its Lie derivatives $\mathcal{L}_d(h_j), \mathcal{L}_d^2(h_j), \ldots \mathcal{L}_d^N(h_j)$ are zero, where N is the finite derivative order at which the chain of ideals $\langle h \rangle \subseteq \langle h, \mathcal{L}_p(h) \rangle \subseteq \cdots$ reaches a fixed point.

$$\text{(dRI)} \quad \frac{\Gamma, Q \vdash \wedge_i^N \wedge_j \mathcal{L}_p^i(h_j) = 0}{\Gamma \vdash [x' = p \ \&\ Q] \wedge_j h_j = 0} \tag{1}$$

Eq. (2) shows how this applies to our case. Read from bottom to top, the differential radical invariant rule (dRI) applies

[7]For instance, at ARCH-COMP 2025, where benchmark 5 involved Lotka-Volterra dynamics, the aim was only to solve for 3.64 time units.

[8]Fig. 3 lists all the rules referenced in this discussion; the intent is not to dwell on their individual details, but to highlight that the proof requires a sequence of complex rule applications that resist straightforward (traditional) automation.

$$\text{(loop)}\quad \frac{\Gamma \vdash J, \Delta \quad J \vdash [\alpha]J \quad J \vdash P}{\Gamma \vdash [\alpha^*]P, \Delta} \qquad (\exists\text{R})\quad \frac{\Gamma \vdash p(e), \Delta}{\Gamma \vdash \exists x\, p(x), \Delta}\ \text{(arbitrary term } e\text{)}$$

$$\text{(dC)}\quad \frac{\Gamma \vdash [x' = f(x)\,\&\,Q]C, \Delta \quad \Gamma \vdash [x' = f(x)\,\&\,(Q \wedge C)]P, \Delta}{\Gamma \vdash [x' = f(x)\,\&\,Q]P, \Delta} \qquad \text{(dI)}\quad \frac{Q \vdash [x' := f(x)](F)'}{F \vdash [x' = f(x)\,\&\,Q]F}$$

Fig. 3: dGL Proof Rules and Axioms relevant to the proof of Model 1. The rules are as follows: differential invariant (dI), differential cut (dC), loop (loop), and exists right (∃R).

to produce a sequent that is propositional in the decidable fragment of real arithmetic, that an automatic prover like Z3 [40] can discharge, establishing an equilibrium point of these nonlinear dynamics.

$$\text{dRI}\frac{\dfrac{*}{\begin{array}{c}\mathrm{x} - g/d = 0 \\ \wedge \mathrm{y} - a/b = 0\end{array} \vdash \begin{array}{c} d(a\mathrm{x} - b\mathrm{xy}) = 0 \\ \wedge b(d\mathrm{xy} - g\mathrm{y}) = 0\end{array}}}{\begin{array}{c}\mathrm{x} - g/d = 0 \\ \wedge \mathrm{y} - a/b = 0\end{array} \vdash \begin{array}{c}[\mathrm{x}' = a\mathrm{x} - b\mathrm{xy},\ \mathrm{y}' = d\mathrm{xy} - g\mathrm{y}] \\ (\mathrm{x} - g/d = 0 \wedge \mathrm{y} - a/b = 0)\end{array}} \tag{2}$$

This reasoning step must be carefully combined with the other reasoning steps to complete the proof. In this example we also need to apply rules such as the *loop* rule with the correct invariant $J \equiv \mathrm{x} \geq x_{\text{mn}} \wedge \mathrm{y} \geq y_{\text{mn}} \wedge \mathrm{x} \leq g/d \wedge \mathrm{y} \leq a/b \wedge g/d \geq x_{\text{mn}} \wedge a/b \geq y_{\text{mn}} \wedge a > 0 \wedge b > 0 \wedge d > 0 \wedge g > 0$ to demonstrate the inductiveness of Angel's strategy over many iterations, and the *exists right* rule ∃R with a correct expression for Angel's choice of `x_add` and `y_add`. Identifying and applying these rules requires both an understanding of the high-level solution strategy of the problem as well as dGL's proof rules, leading to difficulties in automation using traditional techniques which lack the former and need careful, case-by-case encoding of the latter.

Proofs can be expressed succinctly as *tactic* scripts, which are sequences of proof rules with their relevant parameters that when applied in order, complete the proof. Our pipeline was able to automatically generate a 24-line tactic script that completed the proof for this case study, listed in Appendix A.

B. Verification Experimental Outcomes Discussion

This section provides some insight into the qualitative behavior of the pipeline, and the common failure modes.

1) Comparison to Manual Proofs: The proofs generated by the tool are generally longer than human-written proofs. For example, a proof for the Coolant problem written by a human consisted of 45 lines of proof tactic, compared to the 103, 96, and 104 line tactics generated by the tool in the GPT-5.5 high reasoning runs. Part of this, however, can be explained by the fact that the LLMs, because of guidance in the prompt, tend to produce more robust tactics that rely less on KeYmaera X's heuristics-based `auto` tactic.

2) Tracking State: The ability to reason about proof state accurately is important to writing correct Bellerophon tactics, and deteriorates in the absence of reasoning and with smaller models. This leads to two common failure modes: incorrect *positional arguments* and mishandled *proof branching*.

Bellerophon tactics often require *positional arguments* indicating where in the sequent a rule should be applied (e.g., suppose the sequent has the shape $\Gamma \vdash [\alpha_1^*]\phi_1, [\alpha_2^*]\phi_2$. Here, KeYmaera X needs to be told whether to apply the loop rule to the formula $[\alpha_1^*]\phi_1$ at position 1 or the formula $[\alpha_2^*]\phi_2$ at position 2). Identifying the right positional arguments requires tracking the proof state accurately, causing problems for GPT-5.5 without reasoning, GPT-5.4-mini and Qwen3.5.

Proof branching occurs when a proof rule has multiple premises, for example, applying the loop rule leads to three branches (for the base case, inductive case, and postcondition), each of which must receive their own proofs. A KeYmaera X proof tactic, upon applying the loop rule, must therefore specify three sub-tactics to handle the three branches. Getting this right syntactically requires accurate tracking of the proof state. GPT-5.4-mini and Qwen3.5 face difficulties with branching tactics, often specifying too few or too many sub-tactics.

3) Provability of Invariants: Even amongst valid loop invariants (or their continuous analogs), some are easier to prove than others. Invariants that are mathematically more compact can sometimes be harder to prove than "flat" invariants that are a long conjunction/disjunction of simple expressions. This is, for example, the reason that the GPT-5.4 model finds the Van der Pol Oscillator problem hard, failing one run and requiring a larger budget even for the successful runs. It has a tendency to propose a more complex loop invariant than GPT-5.5.

4) Cost and Token Considerations: Generally, GPT-5.4 takes slightly longer to find solutions than GPT-5.5, but often costs less because of its lower per-token cost. For easier problems, non-reasoning runs sometimes have the advantage of not "overthinking" their solutions, and can also perform more queries within the token budget because of no expenditure on reasoning tokens. However, overall, reasoning greatly improves performance for both GPT-5.5 and GPT-5.4-mini, largely because of improved state-tracking and error-recovery based on KeYmaera X feedback.

VI. Synthesis

Having presented and analyzed the verification pipeline in Sections IV and V, we now turn to the synthesis of verified control policies. Verifying Model 1 shows *there exists* some way for Angel to play to win the game. Synthesis is instead about *constructing* a concrete policy that the user desires, which is useful for developing controllers and runtime monitors/supervisors that ensure control remains compliant with a verified nondeterministic policy. The user provides the game and describes their desired solution informally, and the tool computes the corresponding formal nondeterministic policy, verified to be correct (never lose the game or get stuck).

A. Subvalue Maps

In this paper, we represent nondeterministic policies as *subvalue maps* [18], a symbolic equivalent of value functions in reinforcement learning or value tables in dynamic programming. dGL games consist of *subgames*, that compose together with operators like sequential composition (;). An Angelic subvalue map delineates a nondeterministic policy for Angel by mapping each subgame to a *formula*, and constraining Angel to only make choices that result in her entering the next subgame in a state where its mapped formula is true. Fig. 4 shows a subvalue map for the running example that induces the policy of setting populations to the equilibrium point. Here, the subgame $?\mathtt{x\_add} \geq 0$, which follows Angel's choice of $\mathtt{x\_add}$, is mapped to formula $\phi_{\mathtt{x}} \equiv \cdots \wedge d \cdot (\mathtt{x} + \mathtt{x\_add}) = g \wedge \mathtt{x\_add} \geq 0$, constraining Angel in the preceding subgame $\mathtt{x\_add} := *$ to only choose non-negative $\mathtt{x\_add}$ values that set the dynamics to their equilibrium point.

Once the LLM guesses a policy (subvalue map), we can verify if it is correct (non-losing, never stuck) by checking some dGL formulas identified by existing frameworks [18], [19]. For example, in the subvalue map of Fig. 4, to check that the subvalue of the loop, ϕ_{inv}, is correct, we make sure $\phi_{\mathsf{inv}} \rightarrow \phi_{\mathsf{contract}} \wedge \phi_{\mathtt{x}}$ holds, that is, Angel won't lose regardless of whether Demon continues the loop or not. If he ends the loop, Angel will win into ϕ_{contract}, and if he repeats the loop, Angel will still win because $\phi_{\mathtt{x}}$ holds, which inductively tells us that she has a winning strategy from the start of the loop body $\mathtt{x} := *$. While this check was simple (no modalities), for other subgames, checks can involve modal dGL formulas, for which we use the verification tool of Section IV.

For synthesis, the same five case studies are used but with some initial assumptions missing (it will be the task of the synthesis pipeline to come up with all conditions necessary to make a strategy succeed). Additionally, each is associated with an informal one-sentence natural language description of which of the many possible control solutions to synthesize (available in the artifact [20]). Given the running example and a request to synthesize the solution that sets the system to an equilibrium point as soon as possible, our tool automatically identifies the corresponding subvalue map. It computes that Angel must pick $\mathtt{x\_add}$ and $\mathtt{y\_add}$ such that $\mathtt{x} + \mathtt{x\_add} = g/d$ and $\mathtt{y} + \mathtt{y\_add} = a/b$, and that this policy only works when, initially, $\mathtt{x} \leq g/d$, $\mathtt{y} \leq a/b$, $g/d \geq x_{\mathrm{mn}}$, and $a/b \geq y_{\mathrm{mn}}$.

B. Pipeline Description

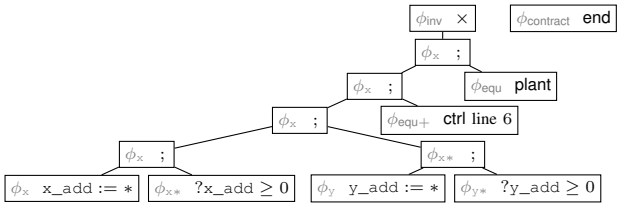
Weakest precondition calculus for normal programs [41] starts at the end and works backwards, requiring loop invariant guesses and retrospective checks at loops. Similarly, our pipeline executes a precondition calculus for subvalue maps [18] on the game backwards from end to start, computing subvalues as preconditions. Some subgames, such as simple assignments, are easy to symbolically execute. Computing the precondition of others: loops, continuous dynamics, and nondeterministic choices, is expensive or undecidable in general. For these, the LLM is prompted to guess subvalue map entries which encode Angel's winning strategy. A retrospective check

1. Input

1) **dGL Game**: *Model 1* (Angel freely chooses $\mathtt{x\_add}$ and $\mathtt{y\_add}$, and wins if she keeps populations $\mathtt{x}$ and $\mathtt{y}$ above the minimum thresholds).
2) **Desired Solution (Informal)**: *"Set the system to its equilibrium point."*

---

2. Synthesized subvalue map (some subgames unexpanded)



where $\phi_{\mathsf{const}} \equiv a > 0 \wedge b > 0 \wedge d > 0 \wedge g > 0$,
$\phi_{\mathsf{pre}} \equiv x_{\mathrm{mn}} \cdot d \leq g \wedge y_{\mathrm{mn}} \cdot b \leq a$,
$\phi_{\mathsf{contract}} \equiv \mathtt{x} \geq x_{\mathrm{mn}} \wedge \mathtt{y} \geq y_{\mathrm{mn}}$,
$\phi_{\mathsf{equ}} \equiv \phi_{\mathsf{const}} \wedge \phi_{\mathsf{pre}} \wedge d \cdot \mathtt{x} = g \wedge b \cdot \mathtt{y} = a$,
$\phi_{\mathtt{y}}* \equiv \phi_{\mathsf{equ}+} \wedge \mathtt{y\_add} \geq 0$,
$\phi_{\mathtt{x}}* \equiv \phi_{\mathtt{y}} \wedge \mathtt{x\_add} \geq 0$,
$\phi_{\mathtt{y}} \equiv \phi_{\mathsf{const}} \wedge \phi_{\mathsf{pre}} \wedge d \cdot (\mathtt{x} + \mathtt{x\_add}) = g \wedge b \cdot \mathtt{y} \leq a$,
$\phi_{\mathtt{x}} \equiv \phi_{\mathsf{const}} \wedge \phi_{\mathsf{pre}} \wedge d \cdot \mathtt{x} \leq g \wedge b \cdot \mathtt{y} \leq a$,
$\phi_{\mathsf{equ}+} \equiv \phi_{\mathsf{const}} \wedge \phi_{\mathsf{pre}} \wedge d \cdot (\mathtt{x} + \mathtt{x\_add}) = g \wedge b \cdot (\mathtt{y} + \mathtt{y\_add}) = a$, and $\phi_{\mathsf{inv}} \equiv \phi_{\mathsf{const}} \wedge \phi_{\mathsf{contract}} \wedge \phi_{\mathsf{pre}} \wedge d \cdot \mathtt{x} \leq g \wedge b \cdot \mathtt{y} \leq a \wedge \mathtt{x} > 0 \wedge \mathtt{y} > 0$.

ctrl line 6 and plant refer to the corresponding parts of Model 1, and end is the special node representing the end of the game. ϕ_{contract} is the same correct control condition from Model 1, that is, $\mathtt{x} \geq x_{\mathrm{mn}} \wedge \mathtt{y} \geq y_{\mathrm{mn}}$. The other formulas are synthesized by the pipeline.

---

3. Control policy derived from subvalue map

In state σ:

1) At $\mathtt{x\_add} := *$ subgame, $\mathtt{x} := e$ is permitted iff $\sigma(\mathtt{x} \mapsto e) \models \phi_{\mathtt{x}}$.
2) At $\mathtt{y\_add} := *$ subgame, $\mathtt{y} := e$ is permitted iff $\sigma(\mathtt{y} \mapsto e) \models \phi_{\mathtt{y}}$.

where $\sigma(\varphi \mapsto e)$ is the state obtained by updating σ to map variable φ to value e.

Fig. 4: A *subvalue map* for the running example.

using Z3 [40] when possible and KeYmaera X when necessary recovers soundness. Fig. 5 shows the synthesis pipeline.

Initially, the LLM analyzes the game, identifying player actions and control modes, and comes up with a detailed strategy for Angel based on the informal guideline provided by the user. The strategy is expressed as a sequence of steps that

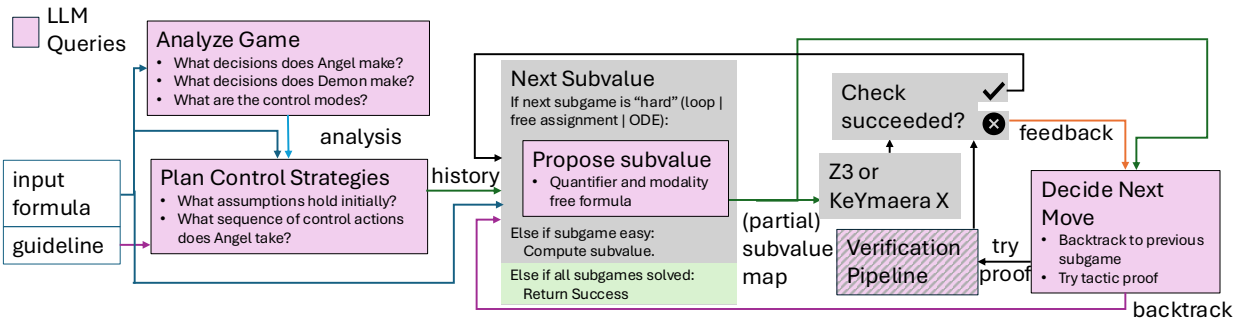


Fig. 5: Synthesis Pipeline. The artifact [20] shows prompts.

Angel will take, stored to provide context in later steps. In the main subvalue map computation phase, the *Next Subvalue* step goes backwards to the next unprocessed subgame, and either symbolically computes the subvalue if it is easy, or prompts the LLM to guess it otherwise. In the latter case, the subvalue is checked using Z3 or KeYmaera X with existing, LLM-free automation. Guesses and their outcomes are stored to provide context for future guesses[9].

If the check succeeds, the pipeline continues to the next unprocessed subgame. Otherwise, there are two possible recovery strategies, that the LLM chooses between. If the lemma was too complicated to verify using existing automation, it should try using the automatic theorem proving system from Section IV. Otherwise, the subvalue may be genuinely incorrect. The incorrectness may lie at the current subgame, or result from a bad earlier subvalue guess that makes the current subgame unsolvable (e.g., a non-inductive loop invariant causing problems in the loop body subvalues). The LLM should choose to backtrack to the subgame where the bad guess was made, and subvalue computation resumes from there.

The interactive theorem proving sub-system used in this pipeline has a modification relative to the one described in Section IV. There is an additional step at the end of every verification loop iteration where the LLM can decide whether the goal it was trying to prove was simply invalid, and it should escape the proof attempt and backtrack to guess a different subvalue.

C. Experimental Results

We apply the synthesis pipeline to all five case studies, using GPT-5 (default reasoning) to make all LLM calls. Each experimental run of the tool consists of four parallel runs of the pipeline of Fig. 5 that terminate when any one of them succeeds, or when they collectively run out of a nine hundred thousand output token budget. Table III summarizes the results of our experiments. A case study is solved when one of the runs of the synthesis tool produces a correct subvalue map. For each benchmark, we perform two runs of the tool (each run consisting of four parallel threads running the synthesis pipeline). In addition, we perform an ablation experiment where we do not allow the LLM to perform interactive theorem proving or backtracking to the subgame of its choosing. The ablation does not separately isolate the effect of backtracking and LLM-assisted proving, instead showing their combined effect. In this variant, the LLM must rely only on KeYmaera X's automation to check the guessed subvalues and on a depth-first search strategy for backtracking on subvalue guesses.

As a baseline, we compare to the tool associated with the only previous work general enough to synthesize control envelopes for our problems [18]. The baseline tool uses a combination of template-based case-by-case heuristics and exact symbolic execution to compute subvalue maps. It is not sufficiently general or scalable to solve any of the five case studies of this paper.

In contrast, our pipeline is very general, free of the need for case-by-case encoding of heuristics, and solves four out of five case studies. Further, our pipeline is able to account for the one-sentence informal guidelines describing the desired control solutions (shown in the artifact [20]), that are part of the input problem. This ability is useful in practice because control engineers often know what kind of solution they are looking for at an informal level, but getting the *exact mathematical details* right can be challenging. A control envelope that focuses on the solutions of interest (rather than as many solutions as possible) can be desirable because it avoids carrying unnecessary arithmetic complexity, making it easier to understand and use downstream (for example, simpler expressions would be more efficient to check when monitoring a control system at runtime for compliance with the envelope). The control envelope is *not formally guaranteed* to match the informal description (it is not clear what such a guarantee would mean given the ambiguity of natural language), but the synthesized map *is* guaranteed to be a correct (non-losing, never stuck) control solution of the game. In practice, in our experiments the synthesized control envelopes do match the informally described solutions.

In the ablation, without LLM-assisted proving, only two out of the five benchmarks are solved. The lower average cost/number of calls for successful benchmarks in the ablation happens since only simpler problems are solved and contribute

[9]Control flow is more complicated for loops, where the loop invariant is guessed first, then the subvalue map of the loop body is computed assuming the invariant holds at the end of the body, and finally the loop invariant is retrospectively checked for inductiveness given the computed subvalue map of the body.

Configuration	Solved	Pass@1	Avg Cost ($)	Avg Calls	Avg Output (k Tokens)
Baseline	0/5	—	—	—	—
Ablation	2/5	0.40 ± 0.24	0.80 ± 0.06	19.5 ± 2.5	76 ± 7
Full Pipeline	**4/5**	$\mathbf{0.80 \pm 0.20}$	$\mathbf{2.19 \pm 0.73}$	$\mathbf{41.0 \pm 10.0}$	$\mathbf{203 \pm 67}$

TABLE III: Synthesis results across configurations. *Solved* indicates the number of case studies where at least one run succeeded. *Pass@1* is the mean, over case studies, of the empirical success rate of a single tool attempt (each attempt having four parallel threads of the pipeline) $\pm$ standard error across case studies. Cost, calls, and output tokens are averaged *only over successful runs* $\pm$ standard error. Output token counts are reported in thousands. The ablation disables LLM-assisted proving and custom backtracking.

Configuration	Chem. Reaction		Coolant		Train		Lotka-Volterra		Van der Pol	
	Succ.	Cost	Succ.	Cost	Succ.	Cost	Succ.	Cost	Succ.	Cost
Ablation	1/1	0.86	0/1	—	0/1	—	0/1	—	1/1	0.74
Full Pipeline	**2/2**	**0.75**	0/2	—	**2/2**	**5.35**	**2/2**	**1.90**	**2/2**	**0.75**

TABLE IV: Per-benchmark synthesis results. *Succ.*: successful runs out of total runs. *Cost*: average dollar cost of *successful runs*.

to the average. Table IV shows per-benchmark outcomes.

The synthesis pipeline failed only on the Coolant problem. The coolant has a subtle loop invariant (subvalue corresponding to the main loop) that is easy to get wrong (and prove right). The LLM did reach nearly-correct loop invariants, dove into the loop body and generated subvalues (differential invariants) for the dynamics, decided to prove that its inner choice of subvalue was correct with an interactive proof since an automatic proof would generally fail, only to ultimately fail the final loop correctness check because the original loop invariant was missing something. It backtracked to make adjustments to the invariant, and got stuck in a series of expensive failed proof attempts. In the ablation experiment, the inability to do LLM-assisted tactical proofs became a bottleneck, since for benchmark problems Train and Lotka-Volterra, the nonlinear dynamics led to proof obligations that would be hard to discharge without tactics.

VII. Conclusion

This paper applies recent advances in Machine Learning (Reasoning LLMs) together with differential game logic, and recent advances in control envelope synthesis to bring a new class of CPS verification and synthesis tasks within reach of automation. Beyond advancing the tool presented in this paper to handle even more complex problems, an interesting direction of future work is applying the new verification capabilities to unlock downstream challenges in cyber-physical systems.

For example, better verification makes it possible to increase the reliability of *autoformalization* (translating natural language to formal specifications). A way to achieve increased reliability of formalizations is by generating multiple independent formalizations and then using verification tools to check that they are consistent with each other. However, verification of dGL autoformalizations [28] is often outside the reach of KeYmaera X's automation. The strengthened verification capabilities provided by this paper could provide a path forward. Another example of an application is scalable *runtime monitor synthesis* [42], where the goal is to synthesize a monitor that observes the system at runtime and intervenes if it departs from the space of admissible behaviors. dGL proofs, the output of our verification tool, can be used to synthesize runtime monitors scalably[10]. Studying the applications of the new verification capabilities to problems of practical interest in industry could lead to significant impact.

VIII. Acknowledgments

This work was partially supported by the National Science Foundation (NSF) under Award Number CCF2427581, DARPA under Agreement FA8750-24-9-1000, an Alexander von Humboldt Professorship, and by the NVIDIA Academic Grant Program using NVIDIA DGX Spark.

Appendix A
Tactic for Model 1

The tactic that the LLM discovers to prove Model 1 is as follows:

```
unfold;
loop("x <= g/d & y <= a/b & g/d >= xmin &
    a/b >= ymin & x > 0 & y > 0 & a > 0 &
    b > 0 & d > 0 & g > 0 & x >= xmin &
    y >= ymin", 1);
<(
  QE("Z3"),
  unfold;
  existsR("g/d-x", 1);
  unfold;
  <(
    QE("Z3"),
    existsR("a/b-y", 1);
    unfold;
    <(
      QE("Z3"),
      unfold;
      print("State before ODE cut after
```

[10]Indeed, for differential dynamic logic (the systems fragment of dGL) the existing runtime monitoring framework ModelPlex [42] already supports this. The ideas can be extended to all of dGL.

```
            choosing xadd and yadd");
      dC("((d*x-g)^2+(b*y-a)^2)=0", 1);
      <(
        dW(1); QE("Z3"),
        dbx("2*(d*x-g)*(b*y-a)*(b*y-d*x)/
             ((d*x-g)^2+(b*y-a)^2)", 1)
      )
    )
  ),
  QE("Z3")
)
```

Here, the print statements help the LLM to track progress and get information about the proof state in case of failure. The use of `QE("Z3")` indicates that the Z3 SMT solver is used to discharge the corresponding proof obligation. In the prompt, the LLM is encouraged to break down the proof rather than using auto so that the proof remains stable despite changes to the automation and quicker to check.

APPENDIX B
DIFFERENTIAL GAME LOGIC

Eq. (3) summarizes the syntax of dGL.

$$\begin{aligned}\alpha \equiv\ & x := e \mid x := * \mid \alpha;\beta \mid ?Q \mid \alpha^* \mid \alpha \cup \beta \\ & \mid \{x' = f(x) \& Q\} \mid x := \otimes \mid !Q \mid \alpha^\times \mid \alpha \cap \beta \\ & \mid \{x' = f(x) \& Q\}^d\end{aligned} \tag{3}$$

Assignment $x := e$ sets variable x to the value of expression e. Nondeterministic assignment $x := *$ allows Angel to set variable x to a real number. Sequential composition $\alpha;\beta$ plays game α followed by game β. Test $?Q$ results in Angel losing if formula Q is false. Repetition α^* lets Angel choose to repeat game α any number of times in a loop. Choice $\alpha \cup \beta$ lets Angel choose to play either game α or game β. Differential game $\{x' = f(x) \& Q\}$ lets Angel choose how long to follow the differential equation $x' = f(x)$ while formula Q remains true, updating variable x per the evolution of the system. In the dual games $x := \otimes$ (nondeterministic assignment), $!Q$ (test), $\alpha^\times$, and $\alpha \cap \beta$ Demon makes the corresponding choices instead of Angel.

The formula $\langle\alpha\rangle Q$ is true when Angel has a strategy to play game α and reach a state where formula Q is true, regardless of how Demon plays. Dually, the formula $[\alpha]Q$ is true when Demon has a winning strategy. Formulas are generated as follows,

$$\begin{aligned}\phi :=\ & \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \\ & \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi\end{aligned}$$

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. dGL *inference rules* provide a way to prove that a formula is valid, i.e., true in all states. An inference rule has the form

$$\frac{\text{premise 1} \quad \cdots \quad \text{premise n}}{\text{conclusion}},$$

and can be read as saying that if all premises are valid, then the conclusion is valid. Each premise and conclusion is a *sequent*, which can be thought of as dGL formulas of the form $\phi_1 \wedge \cdots \wedge \phi_n \rightarrow (\psi_1 \vee \cdots \vee \psi_m)$, where the ϕ_i are the assumptions and ψ_j are the conclusions.

A full explanation, including the semantics of dGL, can be found in the literature [35].

REFERENCES

[1] L. Bu, Y. Li, L. Wang, and X. Li, "Bach: Bounded reachability checker for linear hybrid automata," in *Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '08. IEEE Press, 2008.

[2] A. Kundu, S. Das, and R. Ray, "Sat-reach: A bounded model checker for affine hybrid systems," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 2, Jan. 2023.

[3] R. Ray, A. Gurung, B. Das, E. Bartocci, S. Bogomolov, and R. Grosu, "Xspeed: Accelerating reachability analysis on multi-core processors," in *Hardware and Software: Verification and Testing - 11th International Haifa Verification Conference, HVC 2015, Haifa, Israel, November 17-19, 2015, Proceedings*, ser. Lecture Notes in Computer Science, N. Piterman, Ed., vol. 9434. Springer, 2015, pp. 3–18.

[4] M. Althoff, "Guaranteed state estimation in CORA 2021," in *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21), Brussels, Belgium, July 9, 2021*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 80. EasyChair, 2021, pp. 161–175.

[5] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling, "Juliareach: a toolbox for set-based reachability," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, N. Ozay and P. Prabhakar, Eds. ACM, 2019, pp. 39–44.

[6] A. Platzer, "Logics of dynamical systems," in *LICS*. IEEE, 2012, pp. 13–24.

[7] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *CADE*, 2015, pp. 527–538.

[8] C. Zhou, J. Wang, and A. P. Ravn, "A formal description of hybrid systems," in *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, ser. Lecture Notes in Computer Science, R. Alur, T. A. Henzinger, and E. D. Sontag, Eds., vol. 1066. Springer, 1995, pp. 511–530.

[9] S. Wang, N. Zhan, and L. Zou, "An improved HHL prover: An interactive theorem prover for hybrid systems," in *Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Butler, S. Conchon, and F. Zaïdi, Eds., vol. 9407. Springer, 2015, pp. 382–399.

[10] A. Sogokon, S. Mitsch, Y. K. Tan, K. Cordwell, and A. Platzer, "Pegasus: Sound continuous invariant generation," *Form. Methods Syst. Des.*, vol. 58, no. 1, pp. 5–41, 2022, special issue for selected papers from FM'19.

[11] K. Ghorbal, A. Sogokon, and A. Platzer, "Invariance of conjunctions of polynomial equalities for algebraic differential equations," in *SAS*, ser. LNCS, M. Müller-Olm and H. Seidl, Eds., vol. 8723. Springer, 2014, pp. 151–167.

[12] J. H. Davenport and J. Heintz, "Real quantifier elimination is doubly exponential." *J. Symb. Comput.*, vol. 5, no. 1/2, pp. 29–35, 1988.

[13] A. Platzer and Y. K. Tan, "Differential equation invariance axiomatization," *J. ACM*, vol. 67, no. 1, pp. 6:1–6:66, 2020.

[14] A. Platzer and L. Qian, "Axiomatization of compact initial value problems: Open properties," *J. ACM*, vol. 72, no. 6, pp. 1–51, 2025.

[15] S. Polu and I. Sutskever, "Generative language modeling for automated theorem proving," *ArXiv*, vol. abs/2009.03393, 2020.

[16] J. Qin, A. Du, D. Zhang, M. Lentz, and D. Zhuo, "Can large language models verify system software? a case study using fscq as a benchmark," in *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems*, ser. HotOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 34–41.

[17] S. Varambally, T. Voice, Y. Sun, Z. Chen, R. Yu, and K. Ye, "Hilbert: Recursively building formal proofs with informal reasoning," in *NeurIPS Workshop*, 2025. [Online]. Available: https://arxiv.org/abs/2509.22819

[18] A. Kabra, J. Laurent, S. Mitsch, and A. Platzer, "Hybrid game control envelope synthesis," *Proc. ACM Program. Lang.*, vol. 10, no. OOPSLA1, pp. 850–876, 2026.

[19] ——, "CESAR: Control envelope synthesis via angelic refinements," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, B. Finkbeiner and L. Kovács, Eds., vol. 14570. Springer, 2024, pp. 144–164.

[20] A. Kabra, J. Laurent, R. C. G. Martins, S. Mitsch, and A. Platzer, "Artifact for LLM-Powered Automatic Theorem Proving and Synthesis for Hybrid Systems and Games," 5 2026.

[21] K. Yang, A. M. Swope, A. Gu, R. Chalamala, P. Song, S. Yu, S. Godil, R. J. Prenger, and A. Anandkumar, "Leandojo: Theorem proving with retrieval-augmented language models," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: http://papers.nips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html

[22] E. First, M. N. Rabe, T. Ringer, and Y. Brun, "Baldur: Whole-proof generation and repair with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, S. Chandra, K. Blincoe, and P. Tonella, Eds. ACM, 2023, pp. 1229–1241. [Online]. Available: https://doi.org/10.1145/3611643.3616243

[23] A. Thakur, G. Tsoukalas, Y. Wen, J. Xin, and S. Chaudhuri, "An in-context learning agent for formal theorem-proving," in *First Conference on Language Modeling*, 2024. [Online]. Available: https://openreview.net/forum?id=V7HRrxXUhN

[24] A. Lattuada, T. Hance, C. Cho, M. Brun, I. Subasinghe, Y. Zhou, J. Howell, B. Parno, and C. Hawblitzel, "Verus: Verifying rust programs using linear ghost types," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, pp. 286–315, 2023. [Online]. Available: https://doi.org/10.1145/3586037

[25] K. R. M. Leino, "Dafny: An automatic program verifier for functional correctness," in *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, ser. Lecture Notes in Computer Science, E. M. Clarke and A. Voronkov, Eds., vol. 6355. Springer, 2010, pp. 348–370. [Online]. Available: https://doi.org/10.1007/978-3-642-17511-4_20

[26] A. Platzer, "Differential game logic," *ACM Trans. Comput. Log.*, vol. 17, no. 1, pp. 1:1–1:51, 2015.

[27] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "NL2TL: Transforming natural languages to temporal logics using large language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 15 880–15 903. [Online]. Available: https://aclanthology.org/2023.emnlp-main.985/

[28] A. Kabra, J. Laurent, S. Bharadwaj, R. Martins, S. Mitsch, and A. Platzer, "Can large language models autoformalize kinematics?" in *FMCAD*, A. Irfan and D. Kaufmann, Eds. TU Wien Academic Press, 2025, pp. 78–83.

[29] S. Mitsch, I. Patel, H. H. S. Kannan, X. Jin, B. Zhan, and S. Wang, "ARCH-COMP25 category report: Hybrid systems theorem proving," in *Proceedings of 12th Int. Workshop on Applied Verification for Continuous and Hybrid Systems*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 108. EasyChair, 2025, pp. 152–168. [Online]. Available: /publications/paper/ZlZZ

[30] H. Sheng, A. Bentkamp, and B. Zhan, "HHLPy: Practical verification of hybrid systems using Hoare logic," in *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*, ser. Lecture Notes in Computer Science, M. Chechik, J. Katoen, and M. Leucker, Eds., vol. 14000. Springer, 2023, pp. 160–178. [Online]. Available: https://doi.org/10.1007/978-3-031-27481-7_11

[31] J. J. H. y Munive, S. Foster, M. Gleirscher, G. Struth, C. P. Laursen, and T. Hickman, "IsaVODEs: Interactive verification of cyber-physical systems at scale," *J. Autom. Reason.*, vol. 68, no. 4, p. 21, 2024. [Online]. Available: https://doi.org/10.1007/s10817-024-09709-2

[32] L. C. Paulson and J. C. Blanchette, "Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers," in *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, ser. EPiC Series in Computing, G. Sutcliffe, S. Schulz, and E. Ternovska, Eds., vol. 2. EasyChair, 2010, pp. 1–11.

[33] J. T. Slagel, M. M. Moscato, L. M. White, C. A. Muñoz, S. Balachandran, and A. Dutle, "Embedding differential dynamic logic in PVS," in *Proceedings 18th International Workshop on Logical and Semantic Frameworks, with Applications and 10th Workshop on Horn Clauses for Verification and Synthesis, LSFA/HCVS 2023, and 10th Workshop on Horn Clauses for Verification and SynthesisRome, Italy & Paris, France, 1-2 July, 2023 & 23rd April 2023*, ser. EPTCS, T. Kutsia, D. Ventura, D. Monniaux, and J. F. Morales, Eds., vol. 402, 2023, pp. 43–62. [Online]. Available: https://doi.org/10.4204/EPTCS.402.7

[34] S. Mitsch, H. Sheng, B. Zhan, S. Wang, S. Foster, and J. J. H. Y. Munive, "ARCH-COMP23 category report: Hybrid systems theorem proving," in *Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 96. EasyChair, 2023, pp. 170–188. [Online]. Available: /publications/paper/TtcQ

[35] A. Platzer, *Logical Foundations of Cyber-Physical Systems*. Cham: Springer, 2018.

[36] N. Fulton, S. Mitsch, R. Bohrer, and A. Platzer, "Bellerophon: Tactical theorem proving for hybrid systems," in *ITP*, ser. LNCS, M. Ayala-Rincón and C. A. Muñoz, Eds., vol. 10499. Springer, 2017, pp. 207–224.

[37] J. Laurent and A. Platzer, "Oracular programming: A modular foundation for building llm-enabled software," *arXiv preprint arXiv:2502.05310*, 2025.

[38] L. Geretti, J. A. D. Sandretto, M. Althoff, L. Benet, P. Collins, M. Forets, S. Mitsch, I. Patel, M. Perschl, C. Schilling, and J. Tillet, "ARCH-COMP25 category report: Continuous and hybrid systems with nonlinear dynamics," in *Proceedings of 12th Int. Workshop on Applied Verification for Continuous and Hybrid Systems*, ser. EPiC Series in Computing, G. Frehse and M. Althoff, Eds., vol. 108. EasyChair, 2025, pp. 39–70.

[39] K. Ghorbal and A. Platzer, "Characterizing algebraic invariants by differential radical invariants," in *TACAS*, ser. LNCS, E. Ábrahám and K. Havelund, Eds., vol. 8413. Springer, 2014, pp. 279–294.

[40] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds. Berlin, Heidelberg: Springer, 2008, pp. 337–340.

[41] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Commun. ACM*, vol. 18, no. 8, p. 453–457, Aug. 1975.

[42] S. Mitsch and A. Platzer, "ModelPlex: verified runtime validation of verified cyber-physical system models," *Formal Methods Syst. Des.*, vol. 49, no. 1-2, pp. 33–74, 2016.