Can Large Language Models Autoformalize Kinematics?

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

***Abstract*—Autonomous cyber-physical systems like robots and self-driving cars could greatly benefit from using formal methods to reason reliably about their control decisions. However, before a problem can be solved it needs to be stated. This requires writing a formal physics model of the cyber-physical system, which is a complex task that traditionally requires human expertise and becomes a bottleneck.**

This paper experimentally studies whether Large Language Models (LLMs) can automate the formalization process. A 20 problem benchmark suite is designed drawing from undergraduate level physics kinematics problems. In each problem, the LLM is provided with a natural language description of the objects' motion and must produce a model in differential game logic (dGL). The model is (1) syntax checked and iteratively refined based on parser feedback, and (2) semantically evaluated by checking whether symbolically executing the dGL formula recovers the solution to the original physics problem. A success rate of 70% (best over 5 samples) is achieved. We analyze failing cases, identifying directions for future improvement. This provides a first quantitative baseline for LLM-based autoformalization from natural language to a hybrid games logic with continuous dynamics.

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

I. Introduction

Formal methods seem especially useful in the safety-critical but mathematically complex domain of cyber-physical systems (CPSs), i.e., systems like robots and planes where discrete software interacts with the continuous dynamics of the real world. However, a bottleneck preventing broader industry adoption is the challenge of writing a formalization for these systems [20]. Unlike program verification, where a mathematically meaningful object to verify already exists in the form of a program, for CPSs, we must first create a formal model of the physical system and environment. This process is notoriously difficult, time-consuming, and error-prone.

Can large language models (LLMs) help autoformalize physical systems? They have shown autoformalization abilities for mathematics [12], [24] and CPS contracts [1], [17]. This paper presents a first, experiment-focused exploration of the ability of LLMs to autoformalize the underlying physics problems, faithfully preserving their exact continuous dynamics and discrete transitions in differential game logic (dGL) [18]. We propose 20 benchmarks derived from hard undergraduate kinematics problems [8], [13], [14]. We find that, supported by few-shot prompting and parser feedback, OpenAI o3 has a top-5 accuracy of 70% (success rate on selecting the best out of 5 attempts per benchmark). The main cause of failure is checker limitations for the most complex problems.

Our benchmarks and pipeline are available as an online artifact at https://doi.org/10.1184/R1/28934195. Evaluating autoformalization of a physical system is challenging because every problem has many valid formal representations, and conversely, models can be incorrect for many subtle reasons. To automatically assess the semantic proximity of generated formal models to the original physics problem, we provide a checker that:

1) performs *symbolic execution* over the generated formal model, examining whether the expected symbolic solution to the original physics problem is recovered.
2) ensures the model is not in stasis by requiring that a minimum number of variables are mutated, indicating they are subject to physical effects.

Recovering the expected solution to the original natural language problem is a strong indication that the formalization models the problem correctly, though the checker may sometimes reject answers that are in principle correct but missed some implicit assumptions that the original solution made.

Autoformalization of physical models can have tremendous impact on reliable autonomy. Unlike in mathematics, validating a hypothesized CPS formalization is possible with comparisons between real measurements and predicted measurements during test runs [15]. Easy formalization would allow every modular component to be modeled formally, with all assumptions explicit. Verification could lead to safer systems and synthesis could make control system design easier while ensuring no edge cases are missed relative to the formal model.

II. Overview

This section explains by example what the process of autoformalizing a physics problem involves. Consider the following physics question [8, problem 1.2], which will serve as a running example:

> "A point traversed half the distance with a velocity v_0. The remaining part of the distance was covered

https://doi.org/

with velocity v_1 for half the time, and with velocity v_2 for the other half of the time. Find the mean velocity of the point averaged over the whole time of motion."

Solution: "$v_{\mathsf{avg}} = \frac{2v_0(v_1+v_2)}{2v_0+v_1+v_2}$"

The physical motion described here is hybrid, with continuous dynamics (motion at velocity v_0, v_1, and finally v_2) as well as discrete transitions (changing velocity under specified conditions). Additionally, there are non-trivial conditions describing *when* the discrete transitions happen. We seek to symbolically model this motion formally.

A canonical way to express control problems is via hybrid games [6], [23]. Hybrid games are commonly formalized in two ways: via hybrid automata [6] and via logic [18]. In this paper, we target the logic formalization as it is closer to code, which LLMs have been more exposed to and trained for. *Differential game logic* (dGL) is a logic expressing two-player hybrid games with a relatively complete axiomatization [18]. A detailed and gradual introduction can be found in the literature [19]. Model 1 shows an example of a dGL hybrid game which formalizes the running example.

Model 1 Running Example: Finding Mean Velocity

setup	1	$\langle\, x := 0\,;\, d_{\mathsf{vavg}} := 0\,;$
phase 0	2	$\{x' = v_0, d'_{\mathsf{vavg}} = v_{\mathsf{avg}}\}\;;\;\; d_h := x\,;$
phase 1	3	$t := 0\,;\, \{x' = v_1, d'_{\mathsf{vavg}} = v_{\mathsf{avg}}, t' = 1\}\;;$
phase 2	4	$t_h := t\,;\, \{x' = v_2, d'_{\mathsf{vavg}} = v_{\mathsf{avg}}, t' = 1\}\;;$
transitions	5	$?t = 2t_h\,;\, ?x = 2d_h$
win when	6	$\,\rangle\, d_{\mathsf{vavg}} = x$

In this game, a player (canonically called Angel) runs physical dynamics per the problem specification, moving the point's position x with velocity v_0 in phase 0 on Line 2, with velocity v_1 in phase 1 on Line 3, and with velocity v_2 in phase 2 on Line 4). Angel is forced to transition between phases under the exact conditions specified by the problem because of *tests* (assertions) on Line 5. These assertions use *auxiliary variables* t_h to keep track of the time elapsed in phase 1, t to keep track of time elapsed in phase 1 and phase 2 combined, and d_h to keep track of half the distance covered. On Line 5, test $?x = 2d_h$ ensures that the first phase transition occurred when half the distance was covered. The test $?t = 2t_h$ ensures that the second transition happened at half the total run time of phases 1 and 2. If Angel fails either test condition, she immediately loses the game.

Notice that there is a variable v_{avg} representing average velocity, and Line 6 says that Angel will win the game only if v_{avg} was correct, as checked using auxiliary variable d_{vavg}, which tracks the displacement covered by a particle traveling at v_{avg} through all phases. The value that v_{avg} must hold for Angel to win matches the solution to the original problem, $v_{\mathsf{avg}} = \frac{2v_0(v_1+v_2)}{2v_0+v_1+v_2}$. In this example, there happens to be no adversarial dynamics (Angel is the only player making decisions). However, in problems with adversarial nondeterminism such as unpredictable environmental factors or controller latency, a second, symmetric, adversarial player (canonically called Demon) models the situation by resolving nondeterminism with the goal to make Angel lose. Adversarial dynamics are also useful for optimization problems, e.g., "choose the launch angle that minimizes drift", which can be modeled as Angel and Demon competing to find this angle.

Having formalized the problem as a dGL hybrid game, we next describe how symbolic execution enables a check for whether the model aligns with the natural language question. For the fragment of dGL where most of the chosen benchmark problems to lie (loop-free, polynomial solutions), symbolic execution is decidable and existing work [10] describes how to perform it. Backwards symbolic execution over a game lets us compute the conditions under which Angel (or dually, Demon) can win the game. For example, symbolic execution of the subgame $\langle Line\,5\rangle d_{\mathsf{vavg}} = x$ at the end of Model 1 evaluates to $x = d_{\mathsf{vavg}} \wedge t = 2t_h \wedge x = 2d_h$ which is precisely the weakest precondition starting from which Angel passes the tests of Line 5 and ends the game in a state where she wins. Consider the precondition ϕ of Line 1 computed in this way. Angel should be able to win the game exactly when v_{avg} is set to the correct solution, i.e., $v_{\mathsf{avg}} = \frac{2v_0(v_1+v_2)}{2v_0+v_1+v_2} \leftrightarrow \phi$. Such a check is implemented using Z3 and Mathematica.

Observe that even a minor modeling mistake can result in this check failing. In fact, symbolically executing Model 1 reveals that it is actually equivalent to $\top$. It suffers from a subtle exploit where Angel can always run each phase for 0 time, in which case $x_{\mathsf{avg}} = x = 0$ regardless of what v_{avg} is set to. The problem is that the model lets Angel choose the distance that the particle will travel, as her choice at Line 2 determines the value of half-distance d_h, which in turn determines the distance that phases 1 and 2 must cover. The fix is simple: the assignment $d_h := x$ on Line 2 should be turned to a test $?d_h = x$. The automated check was able to catch this difference and is a strong signal of correctness, especially in combination with manual review. The automated checker has a second test to ensure the system is not in stasis, ruling out empty games that exploit the symbolic execution test by setting the win condition directly to the desired solution. For this example, the second test checks via static analysis that at least two variables are written to, since *any* reasonable model will at least modify the variables representing the particle's position and time. The checker is provided with the minimum number of variables that must be written to and the expected solution for every benchmark problem.

Finally, we see the benefit of formal modeling when the model with Line 2 corrected is still not equivalent to the expected solution. The problem is that the textbook solution is correct *only under some unstated assumptions*: distance d is non-zero, velocity v_0 is positive, and $v_1 + v_2$ is also positive. Finally, Model 2, makes these assumptions. Under the assumptions of Model 2, Line 1, the modal formula from Model 2, Line 2 to Line 7 is equivalent to $\frac{2v_0(v_1+v_2)}{2v_0+v_1+v_2}$. In unverified or informal physics, subtle modeling glitches are

widespread. As a human, it is easy to neglect a measure zero case like $d_h = 0$ during an intermediate calculation, but formalization forces this case to be explicit.

Model 2 Repaired Model: Finding Mean Velocity

assum2	1	$(d_h > 0 \land v_0 > 0 \land v_1 + v_2 > 0) \rightarrow$
setup	2	$\langle\, x := 0\,;\, d_{\mathsf{vavg}} := 0\,;$
phase 0	3	$\{x' = v_0, d'_{\mathsf{vavg}} = v_{\mathsf{avg}}\}\;;\;\; ?d_h = x\,;$
phase 1	4	$t := 0\,;\, \{x' = v_1, d'_{\mathsf{vavg}} = v_{\mathsf{avg}}, t' = 1\}\;;$
phase 2	5	$t_h := t\,;\, \{x' = v_2, d'_{\mathsf{vavg}} = v_{\mathsf{avg}}, t' = 1\}\;;$
transitions	6	$?t = 2t_h\,;\, ?x = 2d_h$
win when	7	$\rangle\, d_{\mathsf{vavg}} = x$

This example begins to demonstrate why formalizing a hybrid physical problem is *not* a simple translation task. Introducing the right auxiliary variables, absent in the specification, is often necessary for a clean, tractable formalization. All the pieces of the model must fit together precisely, e.g., even making a weak inequality strict can transform a correct model into one admitting no runs [22]. Correctness requires careful attention to the subtleties of the problem specification, e.g., benchmark problem 10 (available in the online artifact [9]) has one variable representing a *displacement* and another, representing a *distance*. The former is a signed number while the latter must be modeled as non-negative. Models must prune mathematically degenerate cases using background knowledge of the physical world, e.g., by eliminating imaginary roots of polynomials computing real quantities. Additionally, there are many ways to model the same problem, and the right choice of coordinate system or frame of motion can be crucial to writing a model for which symbolic reasoning is tractable.

Given the complexity of the task, it is unclear whether LLMs can succeed at all, making a careful evaluation necessary. We create a benchmark suite drawing from textbooks and problem sets [8], [13], [14], choosing tricky problems with varying structures to systematically identify the strengths and weaknesses of LLMs on autoformalization.

III. Methodology

The autoformalization system, shown by Fig. 1, takes as input the natural language specification of the physical problem to model, and produces either a correct dGL formalization of a failure (no formalization found that passed the checker). It uses two types of LLM queries: one is *Propose Formalization*, and the second one is *Revise Formalization*. The artifact [9] shows all the prompt templates used.

The autoformalization system first queries the LLM to propose the formalization using the *Propose Formalization* query, providing up to four solved examples as part of the prompt. It provides an LLM with the natural language description of the problem and asks it to produce a dGL formula, including in the prompt common dGL syntax mistakes and their remedies. An example of a problem description the prompt would use based on the running example is

> An object A starts from rest and travels a distance d. For the first half of the distance, it travels with velocity v0. For the second half of the distance, it spends half the time traveling with velocity v1 and half the time traveling with velocity v2. Another object B tracks A. Its motion has the same duration, starting, and ending points as A, but it travels with uniform velocity vavg. Let vavg remain a free variable.

The prompt also includes upto four solved examples. The returned formal model is passed to the KeYmaera X dGL parser for a syntax check.

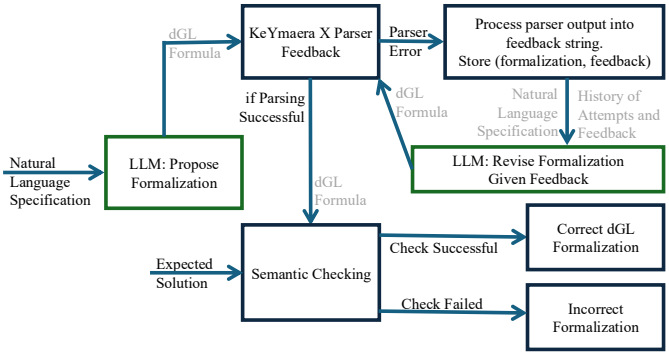

Fig. 1. Autoformalization pipeline

If the syntax check fails, then the parser output is processed into short feedback strings. An example of a parser feedback string is

> The input formula contains an unsupported Unicode character (possibly {`bad_char`}). Use only ASCII characters.

Where `bad_char` is a parameter extracted from parser output. The incorrect formalization along with the feedback is then appended to a list of previous failed formalization attempts.

The LLM is prompted to repair the formalization given past attempts and feedback using the *Revise Formalization* query. The prompt first provides up to two examples of repair. Then it provides the natural language question, and finally past incorrect proposals along with the feedback indicating why these proposals were wrong. The LLM replies with a new, repaired dGL game, which is again passed to the parser for checking. This syntactic repair loop is allowed to run at most three times, after which autoformalization fails. A full implementation along with examples is in the online artifact [9].

If the syntax check succeeds, then the dGL formula is passed onto the semantic evaluation system. This system takes as input the expected symbolic solution to the problem along with the proposed dGL formula. For our running example, the expected solution consists of the formula $v_{\mathsf{avg}} = \frac{2v_0(v_1+v_2)}{2v_0+v_1+v_2}$, to test against for equivalence, and minimum expected number of variables that are written to, 2. If the checker finds that

the dGL formula is equivalent to the expected solution under the initial assumptions of the formula and has enough variable writes, then it declares that the formalization is acceptable. The pipeline is implemented using Delphyne [11] and KeYmaera X [5], which in turn uses Z3 [3] and Mathematica.

IV. Related Work

The use of LLMs for autoformalizing mathematics has received significant research attention [12], [24], including for purely continuous systems with partial differential equation [4], [7]. Recent work also uses LLMs to synthesize contracts for cyber-physical systems [1], [16], [17]. However, we focus on LLM based autoformalization for *physics models* of hybrid systems, with discrete transitions and continuous dynamics faithfully represented, which to the best of our knowledge has not yet been studied. Compared to general mathematics and LTL specifications, we expect this domain to be more data-scarce, with few examples of differential dynamic logic system formalizations and even fewer examples of differential game logic games available on the Internet. Additionally, we expect CPS model formalization to have a more operational flavor, modeling physical realities required to make a system work rather than abstract concepts that can be simplified for theoretical convenience (mathematics) or focused on only output behavior (LTL contracts). This difference makes it worth investigating the hybrid systems modeling domain.

V. Evaluation

We propose 20 benchmarks derived from challenging problems in undergraduate physics textbooks and problem sets [8], [13], [14]. The online artifact [9] lists them. Phrasing is changed to focus on modeling the physical situation described in the problem (rather than on solving for a variable) and the numbers are changed to induce rational answers to avoid failures of the automated check over numerical imprecision. Each benchmark is accompanied by an expected symbolic solution that the model should be equivalent to and the minimum expected number of variables that must be written to. Four additional problems, separate from the benchmarks, serve as examples for few-shot prompting.

We run the LLM autoformalization system using OpenAI GPT 4o, GPT 4.1 (best available OpenAI non-reasoning model) and OpenAI o3 (best available OpenAI reasoning model) on the benchmark set, getting successively improving results. Outcomes are classified into 4 buckets:

1) *Success*: Correct, checker-certified formalization.
2) *Failed*: autoformalization failed either the syntax or semantic check.
3) *Timeout*: The solver timed out after 3 minutes of symbolic execution. Symbolic execution uses real quantifier elimination, which is doubly exponential [2], and can result in timeouts.
4) *Tool Failure*: The solver encountered an expression that it could not automatically symbolically execute (e.g., ODE with a square roots on the right hand side).

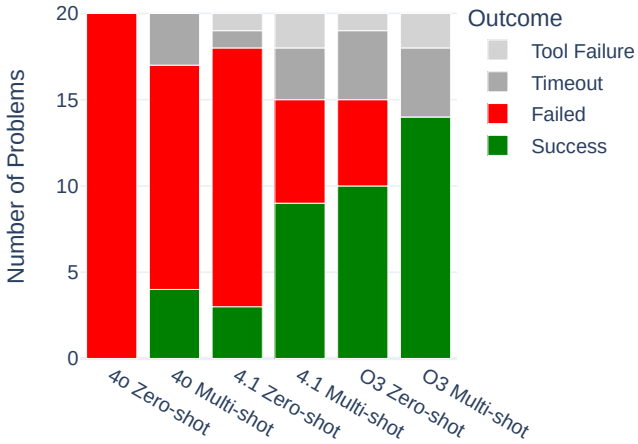

Fig. 2. Outcomes of autoformalization by GPT 4o, GPT-4.1 and o3, with zero-shot and multi-shot prompting.

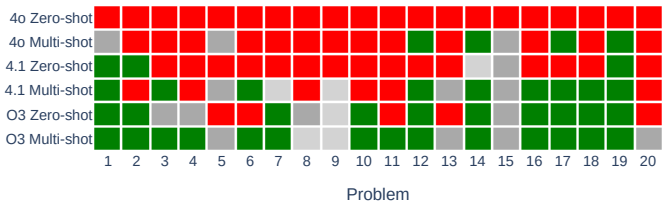

Fig. 3. Problem outcome by model.

Fig. 2 shows the outcome. The autoformalization system is sampled five times independently with temperature 1, and the best outcome is reported (where tool failure and solver timeout are considered better than failure). For each sample, the LLM is given three chances to repair syntax if the KeYmaera X parser reported an error. We evaluate two different prompts, one using few-shot prompting considering four examples (*multi-shot*) and another one without examples (*zero-shot*).

GPT 4o without few-shot prompting fails on every benchmark. Few-shot prompting significantly improves performance for all models. With few-shot prompting o3 has a success rate of 70%, which is higher than GPT 4.1's 45% and GPT 4o's 20%. Our results support that the improvements and the introduction of reasoning LLM models significantly increase their success rate on mathematical tasks, enabling strong performance in the autoformalization of hybrid games.

VI. Discussion

Fig. 3 shows the outcome of each problem by model. We can observe that models found the same problems hard. There are six problems that no model solves. For these, the best performing model produces a formalization that could potentially be correct but is too complex for the checker to verify. The problems can be classified into three groups that explain where their complexity arises from.

1) Optimization Problems: Problem 5 and 20.
2) Non-polynomial continuous dynamics: Problem 8 and 9.
3) Complicated dynamics: Problem 13 and 15.

The first category, for example, includes problem 5, which is about identifying the launch angle of a boat that minimizes drift, modeled by adversarial Angel trying to pick a better angle than Demon, who must set the optimal angle to win. The interplay of the two players leads to combinatorial branching over possible sequence of events, while continuous dynamics occur under constraints preventing control blocking. These constraints make symbolic reasoning about the differential equations particularly computationally expensive.

An example of the second category is problem 8, which is about drones arranged in an equilateral triangle that cyclically pursue each other with constant radial velocity, collapsing inwards in increasingly rapid revolutions, till they crash. Their dynamics, often modeled with a non-polynomial square root expression on the right side of an ODE, cannot be handled automatically by the tool. Problem 13 is an example from the third category. It has three phases of dynamics, including one where acceleration has a continuous dependence on time. Such a differential equation is more expensive to symbolically evaluate, making a timeout the likely outcome.

Sometimes a problem fundamentally requires capabilities that a solver lacks, and the solver must be extended. Other times a difficult but crucial part of formalizing problems is choosing the right abstractions and representations that let proofs succeed. For the six failing problems, the solution likely lies in a combination of these methods. The solver must be extended to better automatically handle, for instance, non-polynomial dynamics by using continuous invariant generation [21]. Complementing this, the LLM should be guided to use tricks like polar coordinates and auxiliary variables to rewrite dynamics in friendlier forms, and to also provide checking hints such as continuous invariants.

Unsolved benchmarks provide a roadmap for improvement in both these directions. Additionally, the rapid improvement in success rate displayed by better LLM models suggests that autoformalization of hybrid games will continue to improve with advancements in LLMs.

VII. Future Work

Beyond the broader adoption of formal methods in offline controller design, CPS model autoformalization enables a new class of applications, where autonomous systems autoformalize unfamiliar situations they face in the open world and derive formally justified control decisions to respond to them. For such applications, the input to autoformalization would be vision and sensor data rather than a natural language description. LLM autoformalization from such multimodal input data provides an interesting future research direction.

Acknowledgment

This work was supported by the National Science Foundation under Award Number 2220311, and by an Alexander von Humboldt Professorship.

Authors' Note

This is a pre-print. The definitive version of record appears in the proceedings of FMCAD 2025 (https://fmcad.org), as: Aditi Kabra, Jonathan Laurent, Sagar Bharadwaj, Ruben Martins, Stefan Mitsch, André Platzer. "Can Large Language Models Autoformalize Kinematics?", FMCAD 2025.

References

[1] Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu 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*, pages 15880–15903, Singapore, December 2023. Association for Computational Linguistics.

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

[3] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[4] Mengge Du, Yuntian Chen, Zhongzheng Wang, Longfeng Nie, and Dongxiao Zhang. Large language models for automatic equation discovery of nonlinear dynamics. *Physics of Fluids*, 36(9), 2024.

[5] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In *CADE*, pages 527–538, 2015.

[6] Thomas A. Henzinger, Benjamin Horowitz, and Rupak Majumdar. Rectangular hybrid games. In *Proceedings of the 10th International Conference on Concurrency Theory*, CONCUR '99, page 320–335, Berlin, Heidelberg, 1999. Springer-Verlag.

[7] Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2678–2710, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

[8] I. E. Irodov. *Problems in General Physics*. Mir Publishers, 1988.

[9] Aditi Kabra, Jonathan Laurent, Sagar Bharadwaj, Ruben Martins, Stefan Mitsch, and Andre Platzer. Artifact for Can Large Language Models Autoformalize Kinematics? 2025.

[10] Aditi Kabra, Jonathan Laurent, Stefan Mitsch, and André Platzer. CESAR: Control envelope synthesis via angelic refinements. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 14570 of *LNCS*, pages 144–164. Springer, 2024.

[11] Jonathan Laurent and André Platzer. Oracular programming: A modular foundation for building llm-enabled software, 2025.

[12] Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Fan Yang, Xian Zhang, and Xiaoxing Ma. Autoformalize mathematical statements by symbolic equivalence and semantic consistency. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 53598–53625. Curran Associates, Inc., 2024.

[13] Samuel J. Ling, Jeff Sanny, and William Moebs. *University Physics: Volume 1*. OpenStax, Houston, TX, September 19 2016.

[14] MIT OpenCourseWare. 8.01SC Classical Mechanics Problem Set 1. https://ocw.mit.edu/courses/8-01sc-classical-mechanics-fall-2016/resources/mit8_01f16_pset1_new/, 2016. Fall 2016, undergraduate level.

[15] Stefan Mitsch and André Platzer. ModelPlex: verified runtime validation of verified cyber-physical system models. *Formal Methods Syst. Des.*, 49(1-2):33–74, 2016.

[16] Daniel Neider and Rajarshi Roy. *What Is Formal Verification Without Specifications? A Survey on Mining LTL Specifications*, pages 109–125. Springer Nature Switzerland, Cham, 2025.

[17] Jiayi Pan, Glen Chou, and Dmitry Berenson. Data-efficient learning of natural language to linear temporal logic translators for robot task specification. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11554–11561, 2023.

[18] André Platzer. Differential game logic. *ACM Trans. Comput. Log.*, 17(1):1:1–1:51, 2015.

[19] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, 2018.

[20] Kristin Yvonne Rozier. Specification: The biggest bottleneck in formal methods and autonomy. In Sandrine Blazy and Marsha Chechik, editors, *Verified Software. Theories, Tools, and Experiments*, pages 8–26, Cham, 2016. Springer International Publishing.

[21] Andrew Sogokon, Stefan Mitsch, Yong Kiam Tan, Katherine Cordwell, and André Platzer. Pegasus: Sound continuous invariant generation. *Form. Methods Syst. Des.*, 58(1):5–41, 2022. Special issue for selected papers from FM'19.

[22] Yong Kiam Tan and André Platzer. Switched systems as hybrid programs∗∗this research was sponsored by the afosr under grant number fa9550-16-1-0288. the first author was also supported by a ∗star, singapore. *IFAC-PapersOnLine*, 54(5):247–252, 2021. 7th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2021.

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

[24] Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.