Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR meets Machine Learning and Explanation

Extracting Verified Action Theories from Informal Specifications via Explanation-Guided Refinement

Stylianos Loukas Vasileiou[1] , **Minh Nguyen**[1] , **Tran Cao Son**[1] , **Huiping Cao**[1] , **Enrico Pontelli**[1]
[1]New Mexico State University
{stelios, minh, stran, hcao, epontell}@nmsu.edu

Abstract

Acquiring correct action theories from informal specifications remains a central challenge in KR. Large Language Models can generate plausible domain models from natural language, but the resulting theories frequently contain missing preconditions, incorrect effects, or superfluous actions. Existing refinement approaches either require human experts to correct these errors or assume that the input specification is itself correct. We present a framework that iteratively refines LLM-generated action theories using formal explanations grounded in SAT-based verification. Each candidate theory is encoded as a bounded SAT problem and tested against solvable tasks, which must admit a valid plan, and unsolvable tasks, which must be correctly rejected. When a test fails, we extract a formal explanation that pinpoints the specific theory constraints responsible for the failure, and feed this explanation back to the LLM to guide its next revision. Our initial evaluation across six planning domains shows that our framework can converge to correct theories.

1 Introduction

Acquiring correct action theories from informal specifications is a long-standing challenge in knowledge representation. In automated planning, formal domain models (e.g., specifying the preconditions and effects of each action) are typically authored by hand in languages like PDDL (Ghallab, Nau, and Traverso 2004). This process is both tedious and error-prone as even domain experts might struggle to fully specify frame axioms, effects, and subtle preconditions. Automating the acquisition of action theories from natural-language descriptions would significantly lower the barrier to deploying planning technology, yet the resulting theories must be formally correct if they are to support sound reasoning about actions and change.

Large language models (LLMs) have emerged as promising tools for generating action theories from text (Liu et al. 2023; Guan et al. 2023; Gestrin, Kuhlmann, and Seipp 2024; Oswald et al. 2024), but a growing body of evidence shows that LLM-generated domain models frequently contain missing preconditions, incorrect effects, or spurious actions (Oswald et al. 2024; Zuo et al. 2025; Huang and Zhang 2025). Several iterative refinement approaches have been proposed, but they either require human experts to inspect and correct the generated models, or they verify only syntactic consistency without checking whether the theory actually supports correct planning behavior. Moreover, all existing approaches assume correct domain specifications, yet in practice descriptions are often incomplete or erroneous, as even a domain expert may omit preconditions, misstate effects, or underspecify action interactions. Kambhampati et al. (2024) articulate a broader vision, the LLM-Modulo framework, in which LLMs serve as idea generators while external model-based critics ensure correctness. Our work in this paper provides a concrete instantiation of this vision for action theory acquisition, with SAT-based verification and formal explanation generation serving as the formal critics.

Particularly, we present a framework (Figure 1) that couples LLM-based theory generation with SAT-based verification and explanation-guided refinement. Specifically, given a natural-language domain description, an LLM generates a candidate action theory, which is encoded as a bounded satisfiability problem and tested against tasks with known solvability status. The test suite includes both *solvable* tasks and *unsolvable* tasks: solvable tasks verify that the theory is *complete* enough to produce plans where plans should exist, while unsolvable tasks verify that the theory is *safe*, i.e., that it does not permit behaviors that should be unsafe or impossible. When a task fails, we compute a formal explanation via minimal unsatisfiable subset (MUS) analysis (Marques-Silva 2012; Vasileiou, Previti, and Yeoh 2021) that pinpoints which theory constraints are responsible, and feed it back to the LLM for revision. The loop repeats until all tasks pass or a budget is exhausted.

Our experimental evaluation of our framework across planning domains with natural-language descriptions of varying fidelity show consistent improvements in convergence rate and final theory correctness, demonstrating that explanation-guided refinement has the potential to improve the reliability of automated action theory acquisition. To our knowledge, this is the first approach that combines formal explanation-based verification in a refinement loop, requiring no human-in-the-loop corrections, and the first to demonstrate robustness to incorrect domain specifications.

2 Related Work

Several systems use LLMs to generate planning domain models from natural language (Liu et al. 2023; Gestrin, Kuhlmann, and Seipp 2024; Oswald et al. 2024). Iterative approaches exist but rely on human corrective feedback

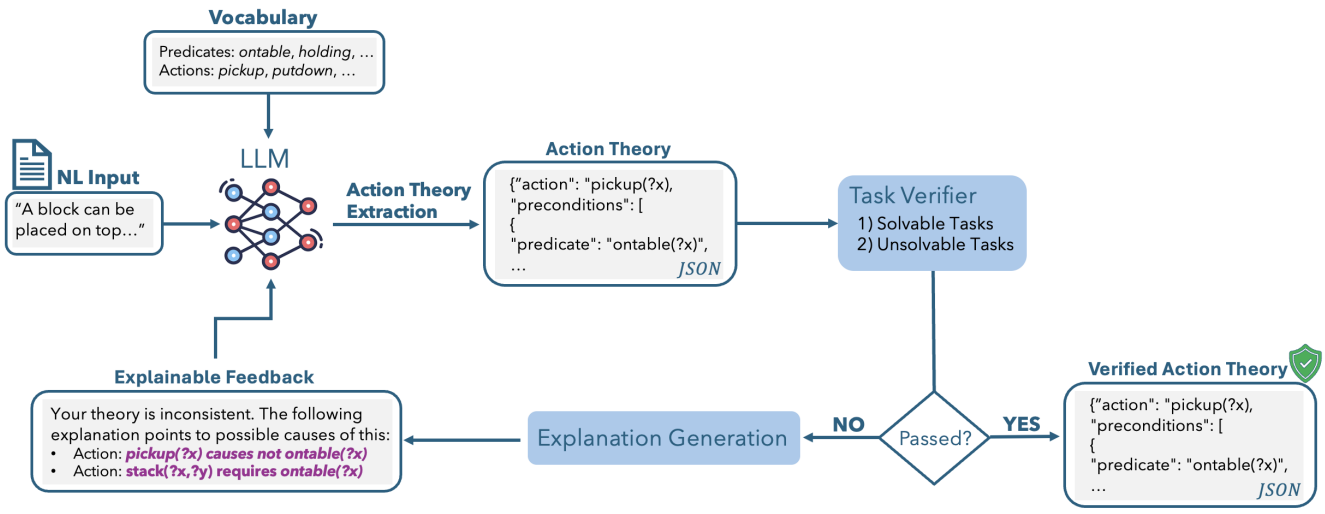

Figure 1: The EgRAT framework for explanation-guided refinement of action theories. An LLM extracts an action theory from an informal description, and the theory is verified against a suite of solvable and unsolvable tasks; if verification fails, a formal explanation is generated and fed back to the LLM to guide its next revision. The loop repeats until all tasks pass or an iteration budget is exhausted.

(Guan et al. 2023) or automated consistency checks that reduce but do not eliminate manual intervention (Smirnov et al. 2024). Moreover, evaluations consistently show that LLM-generated PDDL frequently misaligns with intended domain semantics (Huang and Zhang 2025; Zuo et al. 2025).

Closest to our work is the LLM+AL (Ishay and Lee 2025) framework that uses an LLM to generate BC+ action theories with ASP-based self-revision, but requires a richer input (full typed signatures) and checks only satisfiability and sample queries, not planning tasks with known solvability. The authors report that human corrections are still needed for most benchmarks. Mahdavi et al. (2024) also propose an automated refinement pipeline, but require a domain simulator for feedback and provide only observational signals (state mismatches) rather than formal explanations identifying responsible constraints. They also assume correct domain specifications.

3 Explanation-Guided Refinement

We now describe our proposed framework for *explanation-guided refinement of action theories* (EgRAT). In our setting, an *action theory* is a set of STRIPS-style action schemas, where each schema specifies typed parameters, a set of positive (or negative) precondition predicates, and sets of positive and negative effects. Given an informal description of a domain and a vocabulary with the available predicate and action names to use, the LLM generates an initial candidate action theory in JSON format.[1] Note that the vocabulary constrains which symbols the LLM may use but not how they relate; the LLM must determine all precondition–effect relationships. In practice, such vocabularies can be provided by a user who knows what the domain involves without knowing the precise formalization. This theory is then verified and refined as shown in Figure 1.

3.1 SAT Encoding

Each candidate theory is tested against a suite of tasks with known solvability status. Following the planning-as-satisfiability paradigm (Kautz and Selman 1992), we encode the theory together with a task into a knowledge base KB, which is a set of CNF formulae. The encoding introduces Boolean variables a_t^i for each ground action a^i at each timestep $t \in \{0, \ldots, H-1\}$, and f_t^j for each ground predicate f^j at each timestep $t \in \{0, \ldots, H\}$, where H is the plan horizon. The KB encodes initial and goal states, action constraints, explanatory frame axioms, and action mutexes.

Importantly, initial and goal states are determined by the task and are always correct, while everything else may contain errors that the verification step must detect. During encoding, each clause is tagged with a human-readable *template* describing its origin (e.g., "Action `stack(?x,?y)` requires `holding(?x)`"), which is later used to translate logic-based clauses into natural-language explanations.

3.2 Verification

The test suite contains two types of tasks. *Solvable* tasks specify a set of objects, an initial state, a goal state, and an expected plan length H; a correct theory must find a plan of length at most H. *Unsolvable* tasks also specify objects, an initial state, and a goal state, but no valid plan exists; a correct theory must not admit one.

Solvable tasks. We encode the candidate theory with the task (at horizon H) and check whether the KB is satisfiable. If satisfiable, a plan exists and the task passes. If unsatisfiable, the theory is over-constrained, i.e., it blocks plans that should be valid, indicating wrong or missing preconditions and/or effects. Note that, optionally, an expected (valid) plan can be added to the task, where we additionally force its ac-

[1] We use JSON rather than PDDL because it provides a schema-constrained format that LLMs can generate and parse reliably, avoiding the syntactic brittleness of PDDL serialization.

tion sequence as unit formulae and verify that the KB remains satisfiable under those assumptions, thereby catching theories that admit *some* plan but block the expected ones.

Unsolvable tasks. We encode similarly as before, but we now check whether the KB is unsatisfiable. If it is, then we confirm that the theory correctly rejects the task. If a satisfying assignment is found, the theory is too permissive, i.e., it admits a plan where none should exist, indicating missing preconditions or incorrect effects.

3.3 Explanation Generation via MUS

When a task fails verification, we compute a *formal explanation* by computing minimal unsatisfiable subsets (MUS) (Marques-Silva 2012; Vasileiou, Previti, and Yeoh 2021) from the KB. An MUS is a minimal subset of formulae that is unsatisfiable; removing any single formula restores satisfiability. Each formula in the MUS therefore contributes necessarily to the conflict.

To compute an MUS, we partition the KB into *hard* formulae (initial and goal states, and mutex actions) that are not subject to diagnosis, and *soft* formulae (all other constraints) that are potentially erroneous. The MUS is then computed over the soft formulae only, so it identifies exactly which theory-derived constraints are responsible for the task failure.

Solvable task failures. The MUS identifies the minimal set of theory constraints that, together with the task, make the KB unsatisfiable. For example, an MUS might contain: "Action `stack(?x,?y)` requires `ontable(?x)`" together with a frame axiom, revealing that a spurious precondition prevents a needed action from executing.

Example: Solvable Task Failure in BLOCKSWORLD

Consider a solvable task with initial state {`ontable(a)`, `clear(a)`, `ontable(b)`, `clear(b)`, `handempty`} and goal {`on(a,b)`}, expecting the plan ⟨`pickup(a)`, `stack(a,b)`⟩. Suppose the candidate theory includes a spurious precondition: `stack(?x,?y)` requires `ontable(?x)`. After `pickup(a)` deletes `ontable(a)`, `stack(a,b)` cannot execute, and the encoding is unsatisfiable. The MUS over the theory-derived (soft) clauses returns:

1. `pickup(?x)` causes ¬`ontable(?x)` [effect]
2. `stack(?x,?y)` requires `ontable(?x)` [precondition]

The MUS is mapped to its templates and combined with the failing task description to produce the feedback: *"Task: achieve `on(a,b)` from initial states `{ontable(a), ...}`. The following theory constraints conflict with this task: (1) `pickup(?x)` causes ¬`ontable(?x)`; (2) `stack(?x,?y)` requires `ontable(?x)`. At least one constraint is incorrect. Revise the theory."* Given this feedback, the LLM can identify and remove the spurious precondition in the next iteration.

Unsolvable task failures. When an unsolvable task is incorrectly solved, we extract the (spurious) plan π found by the solver. To explain *why* π achieves the goal, we follow the methodology described in (Vasileiou, Previti, and Yeoh 2021; Vasileiou et al. 2022) for explaining plan validity. In essence, we ask "why does this plan achieve the (task) goal?" The MUS from the KB then identifies the minimal set of formulae that connects the plan's actions to goal achievement; at least one constraint in this chain must be erroneous.

Example: Unsolvable Task Failure in BLOCKSWORLD

Consider an unsolvable task with two blocks, initial state {`handempty`, `ontable(b)`, `on(a,b)`, ¬`clear(a)`}, and goal {`holding(a)`}: since `a` is not clear, no plan should achieve the goal. Suppose, however, that the candidate theory has the wrong-polarity precondition: `unstack(?x,?y)` requires ¬`clear(?x)` instead of `clear(?x)`. The knowledge base is satisfiable, and the solver then returns the plan $\pi = \langle$`unstack(a,b)`$\rangle$. The explanation would then be:

1. `unstack(?x,?y)` requires ¬`clear(?x)` [precondition]
2. `unstack(?x,?y)` causes `holding(?x)` [effect]

This produces the feedback: *"Task: the goal {`holding(a)`} should be unreachable from initial states {`handempty`, `ontable(b)`, `on(a,b)`, ¬`clear(a)`}, but your theory admits the plan ⟨`unstack(a,b)`⟩. The following constraints enable this plan: (1) `unstack(?x,?y)` requires ¬`clear(?x)`; (2) `unstack(?x,?y)` causes `holding(?x)`. At least one constraint is incorrect. Revise the theory."* Given this feedback, the LLM can identify the wrong-polarity precondition and replace it with `clear(?x)`.

Each formal explanation computed is mapped back to its human-readable template, producing a targeted explanation that points the LLM to the possible erroneous theory actions to revise. Multiple explanations can also be computed and provided, which may help to surface distinct failure causes.

3.4 Refinement

The explanation, together with the failing task description, is fed back to the LLM for revision. The LLM is then prompted to produce an updated theory, which is subsequently re-encoded and re-verified. This loop repeats until all tasks pass or the iteration budget is exhausted, with the best-scoring theory retained.

Task selection. At each iteration, all tasks in the suite are evaluated and a single failing task is selected to drive the next refinement. We prioritize *solvable* failures over unsolvable ones, as MUSes from over-constrained encodings tend to localize the missing or spurious constraint more directly than the spurious-plan analysis required for unsolvable failures; if no solvable task fails, the first failing unsolvable task is selected instead. To prevent the loop from stalling on persistently hard tasks (e.g., those requiring structural changes the LLM cannot produce in a single step), a task that fails to be repaired after a small number of consecutive attempts is temporarily skipped so that refinement can make progress elsewhere, and is reinstated as soon as the theory passes it again. Across iterations, we retain the best-scoring theory seen so far and revert to it when the score regresses, ensuring that the loop never moves backward overall.

4 Experimental Evaluation

We evaluated our framework on four classical planning domains (BLOCKSWORLD, MICONIC, LOGISTICS, MYSTERY)

			EgRAT					Ablation				
Domain	p	$\mathcal{A}_0$	$\mathcal{A}^*$	solv.	unsolv.	$\mathcal{I}$	t	$\mathcal{A}^*$	solv.	unsolv.	$\mathcal{I}$	t
BLOCKSWORLD	0.4	10	**20**	10	10	6	1	19	9	10	12	3
	0.6	9	**20**	10	10	5	1	10	0	10	10	2
	0.8	8	**20**	10	10	4	1	10	0	10	10	2
LOGISTICS	0.4	10	**20**	10	10	10	7	10	0	10	10	8
	0.6	10	**20**	10	10	10	13	10	0	10	10	22
	0.8	10	**20**	10	10	10	27	10	0	10	10	23
MICONIC	0.4	10	**20**	10	10	2	1	10	0	10	10	3
	0.6	10	**20**	10	10	6	2	12	2	10	10	3
	0.8	10	**20**	10	10	10	3	10	0	10	10	3
MYSTERY	0.4	9	16	10	6	10	2	10	0	10	10	3
	0.6	0	13	10	3	10	2	10	0	10	10	2
	0.8	2	12	3	9	10	3	9	0	9	10	3
ANT	0.4	12	**20**	10	10	4	2	**20**	10	10	8	3
	0.6	8	**20**	10	10	6	3	10	0	10	10	4
	0.8	10	**20**	10	10	5	2	11	2	9	10	4
WINDOW	0.4	10	**20**	10	10	7	1	**20**	10	10	8	1
	0.6	10	**20**	10	10	6	1	**20**	10	10	9	2
	0.8	10	**20**	10	10	5	1	10	0	10	10	2
Convergence				83% (15/18)					16.6% (3/18)			

Table 1: Results across domains and corruption levels p. $\mathcal{A}_0$: initial theory score (before refinement); $\mathcal{A}^*$: best total score achieved during refinement, with number of solvable (solv.) and unsolvable (unsolv.) tasks passed; $\mathcal{I}$: refinement iterations used; t: runtime (m). **Bold** indicates convergence to a fully correct theory ($\mathcal{A}^* = 20$).

from the IPC benchmarks[2]. We also created two simple, but novel, domains (ANT, involving navigating an ant through a grid; WINDOW involving opening/closing and locking/unlocking windows) to evaluate the framework on domains that current LLMs might not have been trained on. For each domain, we construct a benchmark of 20 evaluation tasks: 10 solvable instances, and 10 unsolvable instances. Solvable tasks test *completeness* (i.e., the theory must admit a plan where one exists), while unsolvable tasks test *safety* (i.e., the theory must not permit plans in situations where no valid plan exists).

To assess robustness to varying initial theory quality, we corrupted the natural language domain description at three levels $p \in \{0.4, 0.6, 0.8\}$, where p controls the fraction of action model components (preconditions and effects) that are randomly removed, added, or have their polarity flipped in the description before being provided to the LLM. This simulates real-world scenarios where domain descriptions might be incorrectly or partially specified.

We compared our framework (EgRAT) with one formal explanation per feedback against an ablation that provides only simple failure indicators (e.g., a given task (with its details) cannot be solved) without formal explanations. Both methods use GPT-5.2 with temperature 0.4 and a maximum token budget of 4096 as the LLM,[3] and an iteration budget of 10. All experiments were run on a Mac Studio (Apple M3 Max, 256 GB RAM). The MUS computation was implemented using the PySAT library (Ignatiev, Morgado, and Marques-Silva 2018). Each experiment was run 5 times and the reported scores are averaged (and rounded) across runs.[4]

Results Table 1 presents the main results across all domains. Most experiments completed in 1–4 minutes, with the exception of LOGISTICS that required 7–27 minutes due to its larger search space (grounding 6 actions, 9 predicates, over multiple objects) producing more complex SAT encodings. EgRAT converges to a fully correct theory in 15 out of 18 experiments (83%), requiring an average of 6.4 iterations when it converges. Specifically, it converged to a correct theory for BLOCKSWORLD, ANT, and WINDOW across all corruption levels within 4–7 iterations. LOGISTICS converged at all three corruption levels but consistently required the full 10 iterations. MICONIC converged in just 2 iterations at p=0.4 but requiring 10 at p=0.8. Interestingly, the MYSTERY domain proved most challenging. While EgRAT reached only $A^* \in \{12, 13, 16\}$ across corruption levels, at lower corruption levels ($p \in \{0.4, 0.6\}$), it solved all 10 solvable tasks but only 3–6 unsolvable tasks, indicating overly permissive theories that fail to block invalid plans. We hypothesize that this is partly due to MYSTERY's obfuscated vocabulary: the domain is structurally isomorphic to BLOCKSWORLD but uses unintuitive predicate and action names (e.g., harmony, feast), preventing the LLM from leveraging its prior knowledge of block-manipulation

[2]https://www.icaps-conference.org/competitions/

[3]We chose GPT-5.2 as it is currently one of the best performing models. We also chose a temperature of 0.4 as it yielded the best results in preliminary experiments.

[4]Code repository: https://github.com/vstylianos/KR26-EgRAT/tree/main

semantics when interpreting the formal feedback. Nonetheless, A^* improves steadily across iterations, suggesting that convergence may be achievable with a higher iteration budget. We plan to investigate this, along with richer feedback strategies such as aggregating explanations across multiple failing tasks, in future work.

In contrast, the ablation converges in only 3 out of 18 completed runs (16.6%), all in relatively simpler configurations (ANT and WINDOW at p=0.4; WINDOW at p=0.6). In 12 of 18 runs, the ablation passes all unsolvable tasks but zero solvable tasks. Without formal explanations identifying which conditions might cause the failure, the LLM defaults to overly restrictive theories that correctly reject invalid plans but cannot admit valid ones. This shows that formal explanations might be necessary for the LLM to meaningfully repair action theories.

5 Conclusions

We presented EgRAT, the first framework for action theory acquisition that combines formal verification with formal explanations in a refinement process without human-in-the-loop corrections and with demonstrated robustness to incorrect domain specifications.

Our work has several limitations that suggest directions for future research. First, the framework currently handles only STRIPS-style action theories; extending to richer formalisms such as conditional effects or temporal planning, would broaden its applicability. Second, the test suite of solvable and unsolvable tasks must be constructed in advance, which assumes some ground-truth knowledge of the domain. Relaxing this requirement, for instance, by bootstrapping tasks from observed execution traces, would make the framework more self-contained. Third, while the formal explanations are locally precise, some domains (such as MYSTERY) remain challenging, suggesting that richer feedback strategies (e.g., aggregating explanations across multiple failing tasks) may be needed for such domains. Finally, evaluating across a broader range of LLMs and experimental settings would clarify how model capability interacts with the quality of formal feedback.

Acknowledgements

The authors acknowledge the partial support of the NSF grants 2139028, 2151254, 1914635, and the internal IAAM grant 139198. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring agencies, or the U.S. government.

AI Declaration

Generative AI tools were used solely to polish the writing and catch grammatical errors; all technical content and ideas, and conclusions are the authors' own.

References

Gestrin, E.; Kuhlmann, M.; and Seipp, J. 2024. NL2Plan: Robust LLM-driven planning from minimal text descriptions. In *ICAPS 2024 Workshop on Human-Aware and Explainable Planning (HAXP)*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory and practice*. Morgan Kaufmann Publishers.

Guan, L.; Valmeekam, K.; Sreedharan, S.; and Kambhampati, S. 2023. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36.

Huang, C., and Zhang, L. 2025. On the limit of language models as planning formalizers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 4880–4904.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*.

Ishay, A., and Lee, J. 2025. LLM+AL: Bridging large language models and action languages for complex reasoning about actions. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 24212–24220.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235, 22895–22907. PMLR.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, 359–363. John Wiley and Sons.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Mahdavi, S.; Aoki, R.; Tang, K.; and Cao, Y. 2024. Leveraging environment interaction for automated PDDL translation and planning with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Marques-Silva, J. 2012. Computing minimally unsatisfiable subformulas: State of the art and future directions. *Journal of Multiple-Valued Logic & Soft Computing* 19.

Oswald, J.; Srinivas, K.; Kokel, H.; Lee, J.; Katz, M.; and Sohrabi, S. 2024. Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 34, 423–431.

Smirnov, P.; Joublin, F.; Ceravola, A.; and Gienger, M. 2024. Generating consistent pddl domains with large language models. *arXiv preprint arXiv:2404.07751*.

Vasileiou, S. L.; Yeoh, W.; Son, T. C.; Kumar, A.; Cashmore, M.; and Magazzeni, D. 2022. A logic-based explanation generation framework for classical and hybrid planning problems. *J. Artif. Intell. Res.* 73:1473–1534.

Vasileiou, S. L.; Previti, A.; and Yeoh, W. 2021. On exploiting hitting sets for model reconciliation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, 6514–6521.

Zuo, M.; Velez, F. P.; Li, X.; Littman, M.; and Bach, S. 2025. Planetarium: A rigorous benchmark for translating text to structured planning languages. In *Proceedings of the Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, 11223–11240.