Language Model as Planner and Formalizer under Constraints

Anonymous ACL submission

Abstract

In recent work on AI planning, Large Language Models (LLMs) are either used as planners to generate executable plans, or as formalizers to represent the planning domain and problem in formal language that can derive plans deterministically. However, both lines of work rely on standard benchmarks that only include generic and simplistic environmental specifications, leaving the robustness of LLMs' planning ability understudied. We bridge this gap by augmenting widely used planning domains with manually annotated, fine-grained, and rich natural language constraints spanning five distinct categories. Our experiments show that introducing constraints significantly decreases performance across all methods, and that the two methodologies each excel on different types of constraints. [1]

1 Introduction

Large Language Models (LLMs) have garnered attention in recent years for their capabilities in planning domains. An intuitive line of work is having them output a plan directly (LLM-as-Planner). With the emergence of reasoning models, formal planning tasks that models originally have struggled on (Valmeekam et al., 2024a; Kambhampati et al., 2024) now have improved performance in accuracy (Valmeekam et al., 2024b; Huang and Zhang, 2025). Another line of work instead has LLMs formalize the problem in some formal language (LLM-as-Formalizer), such as Planning Definition Language (PDDL) (Xie et al., 2023; Liu et al., 2023a; Zhang et al., 2024a,b; Zhu et al., 2024) or a Satisfiability Modulo Theories (SMT) solver, such as Z3 (Hao et al., 2025). This formal representation can then be passed into a solver, which then outputs a plan deterministically. This methodology is more interpretable and trustworthy while showing promising performance.

[1] Our code and data are attached with the submission.

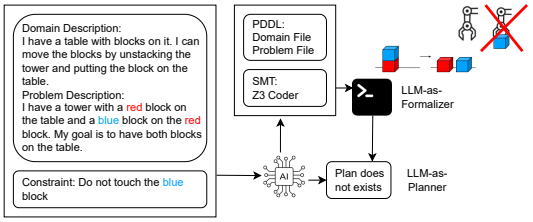

Figure 1: We explore the performance of both LLM-as-Planner and LLM-as-Formalizer when a new constraint is introduced. We also allow problems where the plan does not exists after introducing the constraint.

However, most previous work relies on standard planning benchmarks where environments are described in a generic and simplistic manner (Figure 1 exemplifies the classic BlocksWorld domain (IPC, 1998)). This creates two concerns. First, LLMs may have memorized the specification of these well-established domains, leading to overestimation of their planning or formalizing ability and robustness. Second, real-life planning is filled with various constraints, disregarding which will lead to safety concerns. However, only a few previous works recognized these risks. Yang et al. (2023) translates natural language constraints into Linear Temporal Logic to monitor LLM-as-Planner but does not support general planning per se. Guo et al. (2024) encodes constraints with SMT and combines with PDDL for Task and Motion Planning though without any public data or code. Moreover, they simplify the task by assuming part of the PDDL which is often not realistic, and their categorization of constraints is too coarse-grained. Both works focus on leveraging formal representations for specific robotic tasks, but lack benchmark evaluations and insights from a language perspective.

To bridge this gap, we systematically evaluate both LLM-as-planner and LLM-as-formalizer on planning with constraints. We introduce five fine-grained categories, and manually annotate

rich natural language constraints on the widely used BlocksWorld domain across three distinctive datasets. Using this new benchmark, we find that planning with constraints remains a challenging task, and while using reasoning models in LLM-as-Planner has recently surpassed the performance of LLM-as-Formalizer, LLM-as-Formalizer remains a competitive method and can outperform LLM-as-Planner depending on the constraint type.

2 Task

We consider formal planning tasks where the model is given a textual description description of the domain ($\mathbb{DD}$) and the problem ($\mathbb{PD}$) to output a plan consisting of an sequence of symbolic actions. Each domain and problem can be described via formal languages. While there are many formal languages for this task, we will be focusing on PDDL in this work. Experiments involving Z3 are discussed in Appendix, Section F, due to incomplete and unpromising results. In PDDL, which is native and designed for this purpose, a domain file ($\mathbb{DF}$) describes all the actions and properties that hold true across problems, while a problem file ($\mathbb{PF}$) describes specific configurations of each problem instance.

In addition to the $\mathbb{DD}$ and $\mathbb{PD}$, we also supply the model with a natural language description of a constraint ($\mathbb{C}$). For LLM-as-Planner, the model is asked to output the plan. For LLM-as-Formalizer, the model is asked to formalize the domain, problem and constraint in PDDL which is then passed into a solver to output a plan. To evaluate both approaches above, we work with fully-observable textual environments, where the provided $\mathbb{DD}$ and $\mathbb{PD}$ contain all necessary information for the model to make a complete plan.

3 Data

We consider two widely used planning domains:
BlocksWorld (IPC, 1998) is a domain to rearrange stacks of blocks on a table using a robotic arm.
Mystery BlocksWorld (Valmeekam et al., 2024a) obfuscates the original BlocksWorld domain by replacing all the names of the types, predicates, actions, and objects with nonsensical words, akin to a *wug test* (Berko, 1958). As a control group, it evaluates whether models create plans via lexical pattern-matching and memorization. For both domains, we use the `BlocksWorld-100` and `MysteryBlocksWorld-100` benchmarks Huang and Zhang (2025). Each has 100 instances that consist of $\mathbb{DD}$ and $\mathbb{PD}$, which are input to models, and ground-truth $\mathbb{DF}$ and $\mathbb{PF}$, which are used to evaluate a predicted plan. Since these problems only involve less than 15 blocks, we create another dataset of the same size, `BlocksWorld-XL-100`, where each problem contains 50 blocks to evaluate robustness over complexity.

Next, we manually annotate 100 natural language constraints $\mathbb{C}$ across five categories, which expands the categories from Guo et al. (2024).
Numerical constraints involve numbers and numerical relations. E.g., "The higher number the block, the heavier it is. Once you start moving blocks, do not stack lighter blocks on heavier blocks."
Sequential constraints enforce a specific temporal pattern of actions. E.g., "If you move block1, you must move block2 after."
State-Based constraints involve the states and attributes of entities. E.g., "Block 5 is fragile and no other block can be placed on top of it.".
Initial constraints must hold at the beginning. These constraints will override the original initial states in the problem descriptions.
Goal constraints must hold at the end. These constraints will override the original goal found in the problem descriptions.

Each category includes 20 constraints that are paired with a BlocksWorld problem. Then, we modify the ground-truth PDDL to satisfy the constraint. In total, for each dataset, we have 100 instances with unique $\mathbb{DD}$, $\mathbb{PD}$, $\mathbb{C}$, $\mathbb{DF}$, and $\mathbb{PF}$ tuples. For the Mystery BlocksWorld domain, we obfuscate the keywords in both the constraint $\mathbb{C}$ and descriptions $\mathbb{DD}$ and $\mathbb{PD}$ with meaningless placeholders to keep the constraints consistent across both datasets. Examples are shown in Appendix A.

4 Experimental Setup

Given the $\mathbb{DD}$, $\mathbb{PD}$ and $\mathbb{C}$, we prompt the LLM to produce two different outputs: a **Plan** directly, the **PDDL** $\mathbb{DF}$ and $\mathbb{PF}$, we ask the model to either **generate** the entire PDDL in a zero-shot setting, or to output PDDL without the constraint before suggesting an **edit** to that PDDL to satisfy the given constraint. Both methods also included a revision step where if the generated PDDL leads to an error when solved, the model is given three chances to fix the error by re-generating the PDDL or the edit.

Following past work (Guan et al., 2023; Zhu et al., 2024), the plan produced from LLM-as-

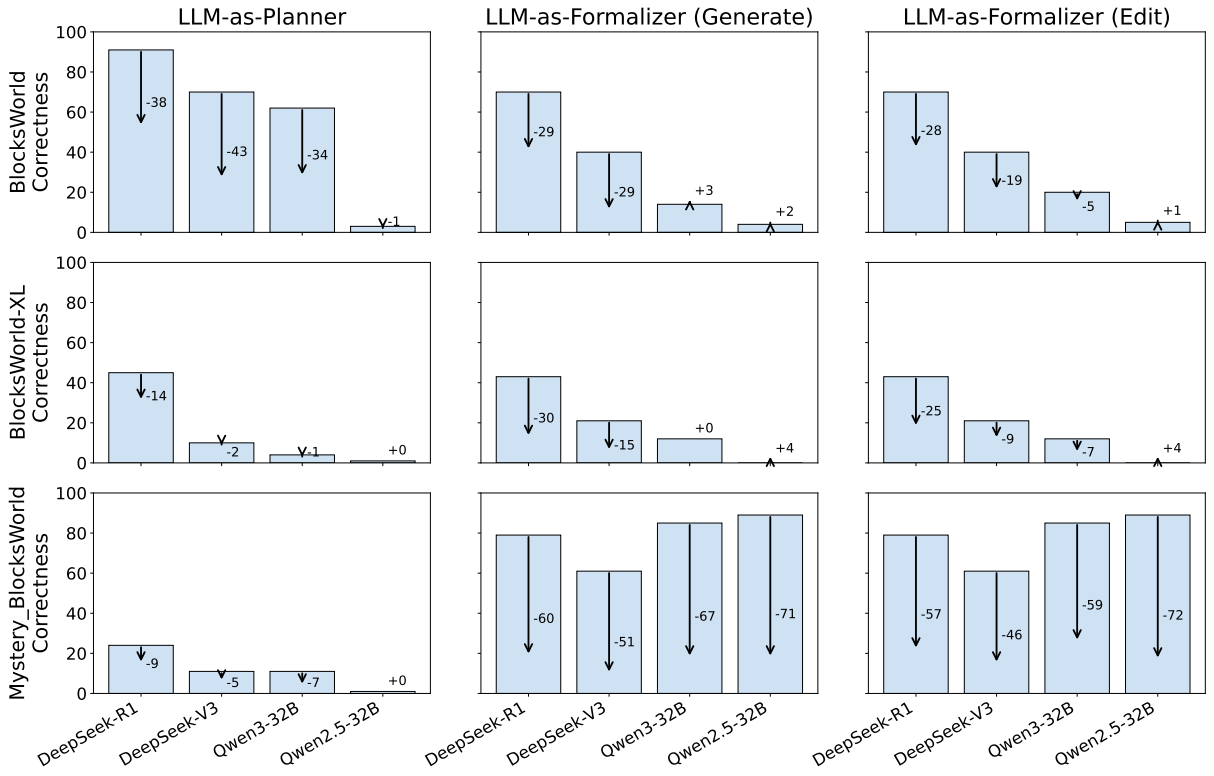

Figure 2: Performance of LLM-as-Planner, LLM-as-Formalizer (PDDL) on `BlocksWorld-100`, `BlocksWorld-XL-100` and `MysteryBlocksWorld-100` data. When constraints are introduced to the task, performance degrades dramatically. Revision is not included in this result to demonstrate that without extra help, this task is difficult. For all results, see Appendix, Section D.

Planner is validated against the ground-truth $\mathbb{DF}$s and $\mathbb{PF}$s provided above, instead of being compared against "ground-truth" plans (Lyu et al., 2023; Liu et al., 2023b; Pan et al., 2023) since there could be multiple correct plans. Similarly, the predicted $\mathbb{DF}$ and $\mathbb{PF}$ for the LLM-as-formalizer approach are not compared against the ground-truth, as only the eventual plan is validated because there might be more than one way to formalize. We evaluate the predicted plans using *correctness* which indicates the percentage of generated PDDL that passes the solver and leads to correctly validated plans. We use dual-bfws-ffparser planner implemented by Muise (2016) as the solver and VAL (Howey et al., 2004) as the validator. See more details in Appendix C.

For both of the LLM-as-planner and LLM-as-formalizer approaches, we consider Deepseek-R1, Deepseek-V3 (Guo et al., 2025), Qwen3-32B (Team, 2025), and Qwen2.5-Coder-32B-Instruct (Hui et al., 2024). We query these models using KANI (Zhu et al., 2023) with default hyperparameters on 4 H100 GPUs.

5 Results

Figure 2 displays the performance of LLM-as-Planner and LLM-as-Formalizer with the generate and edit setting (without revision) on the constrained and non-constrained data. **Almost all models paired with both methods struggle when constraints are added**. On `BlocksWorld-100`, DeepSeek-R1, -V3, and Qwen3-32B excel as planners, but their performance falls by almost half when given constraints. On `BlocksWorld-XL-100`, which is much more challenging for all settings as expected, a similar trend can be observed. As formalizers, models see a similar degradation, notably on `MysteryBlocksWorld-100` where all performance decreases by about 80%.

Figure 3 displays the performance by constraint categories of LLM-as-Planner and all settings of LLM-as-Formalizer. **The two methodologies prefer and struggle on different constraints.** LLM-as-Planner performs well on sequential, state-based and goal constraints, while LLM-as-Formalizer excels on state-based constraints. A possible explanation could be as the state-based constraints impact the predicates and states in PDDL, which makes the

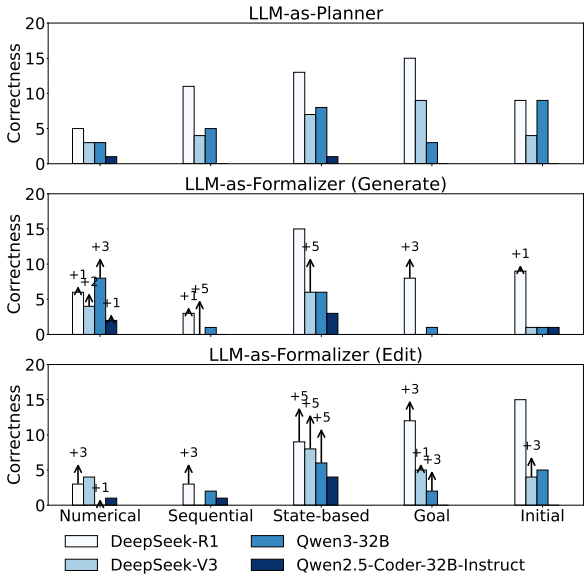

Figure 3: Performance of LLM-as-Planner and LLM-as-Formalizer (PDDL) by constraint categories on `BlocksWorld-100`. Each category contains 20 instances. The arrows represent the performance increase in the correctness once revision was included.

modification easy to generate. Meanwhile, numerical and sequential constraints are harder to encode in PDDL, leading to the lower correctness.

Figure 3 also displays that generating the entire PDDL excels on different constraints that editing the non-constrained PDDL. The performance on adding State-based constraints benefits from the generation task, while initial and goal constraints benefit from the editing task. This makes sense as the initial and goal constraints override the original problem setup which is easier to edit than to generate the entire PDDL based on two different problem setups.

We explore qualitative examples of where both methodologies succeed or fail. For full examples see Appendix, Section E. LLM-as-Planner tends to fail when the model outputs a plan that does not satisfy the newly added constraints. For example, for the constraint "Once you unstack a block, you cannot put it down on the table.", Deepseek-R1 returned a plan that did not satisfy the newly-added constraint (it unstacks block6 before putting it on the table) but it also does not follow PDDL rules (it attempts to pick up block5 even though block5 has block3 on top of it).

Whereas for LLM-as-Formalizer (Generate) on the same problem, Deepseek-R1 correctly introduces a new predicate that satisfies the constraint, which is added to the unstack and putdown action, and when passed through a solver, it correctly determines that with the added constraint, a plan does not exist.

However, LLM-as-Formalizer tends to struggle with semantic errors, such as missing needed predicates to satisfy constraints. For example, on the constraint "If you move block1, you must move block2 after.", LLM-as-Formalizer (Generate) using Deepseek-R1 returned a $\mathbb{DF}$ Which not only forgets predicates found in the non-constrained version of the problem, but also does not include any predicates that would satisfy the constraint. When passed into the solver, it output a plan that did not satisfy the constraint. In contrast, for LLM-as-Planner, Deepseek-R1 returned a plan which satisfies the added constraint by moving block2 once block1 has been moved.

6 Conclusion

We study the robustness of two types of LLM planning frameworks—LLM-as-Planner and LLM-as-Formalizer—by introducing rich, fine-grained constraints into planning problems. To enable a thorough comparison, we develop a benchmark by integrating and extending existing datasets and manually annotating constraints with natural language descriptions. Additionally, we implement two LLM-as-Formalizer strategies, **generate** and **edit**, and evaluate them alongside LLM-as-Planner across four models on our benchmark. The experimental results show that (1) planning under constraints remains a significant challenge for both frameworks, and (2) neither framework is universally better—performance depends on the type of constraint, with LLM-as-Formalizer outperforming on some categories. We further analyze and characterize common failure modes of both frameworks (Section 5). We hope our implementation of LLM-as-Formalizer and the proposed benchmark provide a foundation for future research in constraint-aware planning with language models.

7 Limitation

Our categorization of constraints is not the only way to categorize constraints, and many constraints can fall into more than one category. For example "Do not stack block 1 on block 2" can be considered a state-based constraint, as well as a sequential constraint, as it impacts the sequence of actions needed to perform the task. Nevertheless, we believe this categorization of constraints is rich and is a good

way to see what types of constraints different languages perform well on.

Our evaluation metric is counting the number of correct plans both from using the LLM as a planner, as well as using a solver after generating PDDL or Z3. However, there is a chance that false positives will occur as the plan may be correct, but the generated code does not actually satisfy the constraint. While there has been previous work, like Zuo et al. (2024) that evaluated generated PDDL using graph isomorphism to make sure that the generated problem file is equivalent to a ground-truth problem file, such strategy does not exist for evaluating domain files. Therefore, we feel this is the best method to evaluate generating entire PDDL despite the chance of false positives.

Since this work uses only the BlocksWorld and Mystery BlocksWorld, it is a small toy example to the usage of LLMs as formalizers and are not representative to problems in the real world, which would be much more challenging. This may pose a risk to users using this code on real world problems.

The datasets we use and we propose are all under the MIT License.

References

Jean Berko. 1958. The child's learning of english morphology. *Word*, 14(2-3):150–177.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Weihang Guo, Zachary Kingston, and Lydia E. Kavraki. 2024. Castl: Constraints as specifications through llm translation for long-horizon task and motion planning. *Preprint*, arXiv:2410.22225.

Yilun Hao, Yang Zhang, and Chuchu Fan. 2025. Planning anything with rigor: General-purpose zero-shot planning with LLM-based formalized programming. In *The Thirteenth International Conference on Learning Representations*.

R. Howey, D. Long, and M. Fox. 2004. Val: automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–301.

Cassie Huang and Li Zhang. 2025. On the limit of language models as planning formalizers. *Preprint*, arXiv:2412.09879.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

IPC. 1998. International planning competition. https://www.icaps-conference.org/competitions.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023a. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Hanmeng Liu, Ruoxi Ning, Zhiyang Teng, Jian Liu, Qiji Zhou, and Yue Zhang. 2023b. Evaluating the logical reasoning ability of chatgpt and gpt-4. *arXiv preprint arXiv:2304.03439*.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 305–329, Nusa Dua, Bali. Association for Computational Linguistics.

Christian Muise. 2016. Planning.Domains. In *The 26th International Conference on Automated Planning and Scheduling - Demonstrations*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore. Association for Computational Linguistics.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2024a. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. 2024b. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. *Preprint*, arXiv:2409.13373.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*.

Ziyi Yang, Shreyas S. Raman, Ankit Shah, and Stefanie Tellex. 2023. Plug in the safety chip: Enforcing constraints for llm-driven robot agents. *Preprint*, arXiv:2309.09919.

Li Zhang, Peter Jansen, Tianyi Zhang, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024a. PDDLEGO: Iterative planning in textual environments. In *Proceedings of the 13th Joint Conference on Lexical and Computational Semantics (*SEM 2024)*, pages 212–221, Mexico City, Mexico. Association for Computational Linguistics.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024b. PROC2PDDL: Open-domain planning representations from texts. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ACL 2024)*, pages 13–24, Bangkok, Thailand. Association for Computational Linguistics.

Andrew Zhu, Liam Dugan, Alyssa Hwang, and Chris Callison-Burch. 2023. Kani: A lightweight and highly hackable framework for building language model applications. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 65–77, Singapore. Association for Computational Linguistics.

Wang Zhu, Ishika Singh, Robin Jia, and Jesse Thomason. 2024. Language models can infer action semantics for classical planners from environment feedback. *arXiv preprint arXiv:2406.02791*.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael L Littman, and Stephen H Bach. 2024. Planetarium: A rigorous benchmark for translating text to structured planning languages. *arXiv preprint arXiv:2407.03321*.

A Data Examples

We augment constraint descriptions to a ground-truth domain and problem file and create new ground-truth PDDL files that now encode the ground-truth. Here we include an example constraint description and the ground-truth PDDL files for the constraint + problem pair. Listings 1, 2 and 3 display an example constraint description and the annotated ground-truth $\mathbb{DF}$ and $\mathbb{PF}$ for a paired BlocksWorld problem. Listing 4 shows an example ground-truth $\mathbb{PF}$ for the XL dataset, the $\mathbb{DF}$ remains the same. Listings 5, 6 and 7 display the corresponding constraint description, $\mathbb{DF}$ and $\mathbb{PF}$ that have now been obfuscated.

B Prompts

Listings 8, 9, 10, 11 and 12 displays the prompts for all experiment settings given to all models. Whenever possible, we asked the model to return the output in a JSON object for easier parsing.

C Experimental Setup Details

C.1 Planner

We use a `dual-bfws-ffparser` planner to find plans from the generated PDDL. This is a best first width search planner used with a fast forward planner.

C.2 VAL

The VAL library takes in a ground-truth PDDL $\mathbb{DF}$, $\mathbb{PF}$ and a plan and tries to execute the plan in the environment by checking whether each action in the found plan can be executed based on the preconditions written in the ground-truth files. If the plan is executable, VAL then checks whether the final state after executing the plan matches the goal state found in the ground-truth $\mathbb{PF}$. If either the plan is not executable or the final state does not match the goal state, VAL will return an error, therefore the plan found by the planner is not correct.

D Full PDDL Results

Beyond the visualizations above, we show the detailed results of all models on all datasets.

Figures 6-32 display all results.

E Example Model Outputs

Listings 13, 14, 15 and 16 display example full model outputs for LLM-as-Planner and LLM-as-Formalizer (Generate) using Deepseek-R1 once constraints are introduced.

F Z3 Results

For SMT, which is designed for constraint satisfaction, the planning domain and problem are encoded directly as logical constraints, while the plan is an array of assignable variables that satisfies all defined constraints over a fixed number of steps.

We evaluate the model's ability to output SMT code implemented through the Z3 Python Library. We ask the model to generate the entire Z3 code in a zero-shot setting. We also include a revision step where if the generated Z3 code leads to an error when solved, the model is given 2 chances to fix the error.

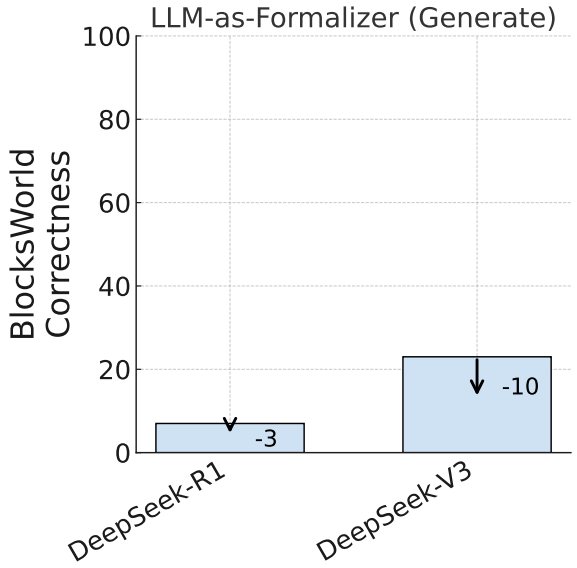

Figure 4: Performance of LLM-as-Formalizer (SMT) on BlocksWorld-100 data. When constraints are introduced to the task, performance degrades. Revision is not included in this result to demonstrate that without extra help, this task is difficult.

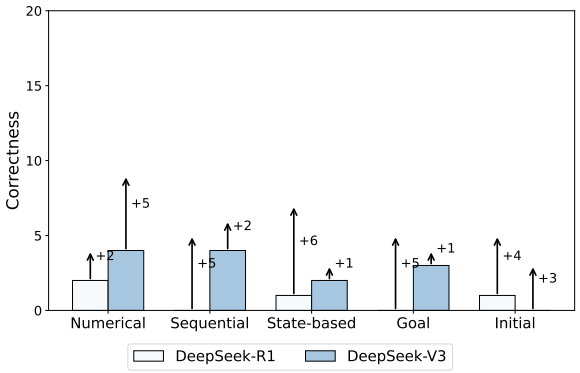

Figure 5: Initial performance vs. final performance of LLM-as-Formalizer (SMT) by constraint categories on BlocksWorld-100 after two revisions. Each category contains 20 instances. Results of Qwen2.5-Coder-32B-Instruct and Qwen3-32B are omitted due to having a 0 percent success rate.

Figure 4 displays that performance drops for Deepseek-R1 and Deepseek-V3 once constraints are added, before any revision steps are taken. Qwen3-32B and Qwen2.5-Coder-32B-Instruct both achieve 0% for both non-constrained and constrained data. This pattern aligns with what we observed across our Z3 experiments, where plan generation through Z3 remained largely unreliable. This appears to stem from Z3's logic-heavy format, which demands a level of syntactic and semantic precision that LLMs do not easily handle in situations with constraints.

From Figure 5, revision improves the performance for Z3 Generation, however results are still much worse than PDDL generation, signifying the difficulty of this task for Z3. This figure shows how plan correctness improves after revision across constraint types. DeepSeek-V3 outperforms DeepSeek-R1 in the numerical and sequential categories. Both models show gains in each category, suggesting that some constraints, such as those with clear step-by-step logic, are easier for Z3 to work with. Others, like goal and initial, appear more difficult, likely because they require more implicit reasoning that is harder for the LLMs to express in Z3's formal logic.

Listing 1: Example constraint description for a state-based constraint

```
Do not stack block1 on top of block2
```

Listing 2: Annotated ground-truth Domain File for BlocksWorld-100 with state-based constraint description

```
(define (domain blocksworld)
;; CONSTRAINT Do not stack block1 on top of block2.
  (:requirements :strips)
(:predicates (clear ?x)
            (on-table ?x)
            (arm-empty)
            (holding ?x)
            (on ?x ?y)
;; BEGIN ADD
             (do-not-stack ?x ?y)
;; END ADD
            )

(:action pickup
  :parameters (?ob)
  :precondition (and (clear ?ob) (on-table ?ob) (arm-empty))
  :effect (and (holding ?ob) (not (clear ?ob)) (not (on-table ?ob))
              (not (arm-empty))))

(:action putdown
  :parameters (?ob)
  :precondition (holding ?ob)
  :effect (and (clear ?ob) (arm-empty) (on-table ?ob)
              (not (holding ?ob))))

(:action stack
  :parameters (?ob ?underob)
  :precondition (and (clear ?underob) (holding ?ob)
;; BEGIN ADD
                     (not (do-not-stack ?ob ?underob))
;; END ADD
               )
  :effect (and (arm-empty) (clear ?ob) (on ?ob ?underob)
              (not (clear ?underob)) (not (holding ?ob))))

(:action unstack
  :parameters (?ob ?underob)
  :precondition (and (on ?ob ?underob) (clear ?ob) (arm-empty))
  :effect (and (holding ?ob) (clear ?underob)
              (not (on ?ob ?underob)) (not (clear ?ob)) (not (arm-empty)))))
```

Listing 3: Annotated ground-truth Problem File for BlocksWorld-100 with state-based constraint description

```
(define (problem blocksworld-p50)
;; CONSTRAINT Do not stack block1 on top of block2.
  (:domain blocksworld)
  (:objects block1 block2 block3 block4 block5 block6 block7 )
  (:init
    (on-table block3)
    (clear block3)
    (on-table block5)
    (clear block5)
    (on-table block7)
    (on block1 block7)
    (clear block1)
    (on-table block2)
    (clear block2)
    (on-table block6)
    (clear block6)
    (on-table block4)
    (clear block4)
    (arm-empty)
```

```
;; BEGIN ADD
    (do-not-stack block1 block2)
;; END ADD
  )
  (:goal (and
    (on-table block5)
    (on block2 block5)
    (on-table block4)
    (on block7 block4)
    (on block6 block7)
    (on block1 block6)
    (on-table block3)
  ))
)
```

Listing 4: Annotated ground-truth Problem File for the XL BlocksWorld-100 problems with state-based constraint description

```
(define (problem blocksworld-p50)
;; CONSTRAINT Do not stack block1 on top of block2.
  (:domain blocksworld)
  (:objects block1 block2 block3 block4 block5 block6 block7 block8 block9 block10 block11 block12
      block13 block14 block15 block16 block17 block18 block19 block20 block21 block22 block23
      block24 block25 block26 block27 block28 block29 block30 block31 block32 block33 block34
      block35 block36 block37 block38 block39 block40 block41 block42 block43 block44 block45
      block46 block47 block48 block49 block50 )
  (:init
    (on-table block10)
    (on block41 block10)
    (on block50 block41)
    (on block29 block50)
    (on block46 block29)
    (on block43 block46)
    (on block5 block43)
    (on block38 block5)
    (on block48 block38)
    (on block8 block48)
    (on block11 block8)
    (on block22 block11)
    (on block13 block22)
    (on block35 block13)
    (on block49 block35)
    (on block20 block49)
    (on block31 block20)
    (on block34 block31)
    (on block17 block34)
    (on block28 block17)
    (on block14 block28)
    (on block47 block14)
    (on block26 block47)
    (on block6 block26)
    (on block4 block6)
    (on block25 block4)
    (on block9 block25)
    (on block23 block9)
    (on block15 block23)
    (on block21 block15)
    (on block18 block21)
    (on block39 block18)
    (on block33 block39)
    (on block1 block33)
    (on block3 block1)
    (on block2 block3)
    (on block44 block2)
    (on block16 block44)
    (on block45 block16)
    (on block7 block45)
    (on block40 block7)
    (on block24 block40)
```

```
    (on block27 block24)
    (on block36 block27)
    (on block19 block36)
    (clear block19)
    (on-table block32)
    (clear block32)
    (on-table block42)
    (on block37 block42)
    (on block30 block37)
    (on block12 block30)
    (clear block12)
    (arm-empty)
;; BEGIN ADD
    (do-not-stack block1 block2)
;; END ADD
  )
  (:goal (and
    (on-table block46)
    (on block45 block46)
    (on block37 block45)
    (on-table block30)
    (on block39 block30)
    (on-table block34)
    (on-table block49)
    (on-table block19)
    (on-table block38)
    (on-table block16)
    (on-table block25)
    (on-table block18)
    (on block5 block18)
    (on block17 block5)
    (on-table block42)
    (on-table block3)
    (on block32 block3)
    (on block35 block32)
    (on block26 block35)
    (on-table block12)
    (on-table block33)
    (on-table block14)
    (on-table block2)
    (on-table block23)
    (on block1 block23)
    (on block6 block1)
    (on-table block4)
    (on block20 block4)
    (on-table block48)
    (on-table block27)
    (on-table block31)
    (on-table block44)
    (on block47 block44)
    (on-table block43)
    (on-table block36)
    (on-table block7)
    (on-table block22)
    (on-table block28)
    (on-table block9)
    (on-table block11)
    (on-table block50)
    (on-table block24)
    (on-table block41)
    (on-table block8)
    (on-table block21)
    (on-table block40)
    (on-table block13)
    (on-table block15)
    (on-table block29)
    (on-table block10)
  ))
)
```

Listing 5: Example constraint description for a state-based constraint for Mystery BlocksWorld

```
Do not perform action3 with object1 and object2.
```

Listing 6: Annotated ground-truth Domain File for Mystery BlocksWorld-100 with state-based constraint description

```
(define (domain mystery_blocksworld)
;; CONSTRAINT Do not perform action3 with object1 and object2.
  (:requirements :strips)
(:predicates (predicate1 ?x)
            (predicate2 ?x)
            (predicate3)
            (predicate4 ?x)
            (predicate5 ?x ?y)
;; BEGIN ADD
             (not-allowed ?x ?y)
;; END ADD
            )

(:action action1
  :parameters (?object1)
  :precondition (and (predicate1 ?object1) (predicate2 ?object1) (predicate3))
  :effect (and (predicate4 ?object1) (not (predicate1 ?object1)) (not (predicate2 ?object1))
              (not (predicate3))))

(:action action2
  :parameters (?object1)
  :precondition (predicate4 ?object1)
  :effect (and (predicate1 ?object1) (predicate3) (predicate2 ?object1)
              (not (predicate4 ?object1))))

(:action action3
  :parameters (?object1 ?object2)
  :precondition (and (predicate1 ?object2) (predicate4 ?object1)
;; BEGIN ADD
                    (not (not-allowed ?object1 ?object2))
;; END ADD
               )
  :effect (and (predicate3) (predicate1 ?object1) (predicate5 ?object1 ?object2)
              (not (predicate1 ?object2)) (not (predicate4 ?object1))))

(:action action4
  :parameters (?object1 ?object2)
  :precondition (and (predicate5 ?object1 ?object2) (predicate1 ?object1) (predicate3))
  :effect (and (predicate4 ?object1) (predicate1 ?object2)
              (not (predicate5 ?object1 ?object2)) (not (predicate1 ?object1)) (not (predicate3)))))
```

Listing 7: Annotated ground-truth Problem File for Mystery BlocksWorld-100 with state-based constraint description

```
(define (problem mystery_blocksworld-p50)
;; CONSTRAINT Do not perform action3 with object1 and object2.
  (:domain mystery_blocksworld)
  (:objects object1 object2 object3 object4 object5 object6 object7 )
  (:init
    (predicate2 object3)
    (predicate1 object3)
    (predicate2 object5)
    (predicate1 object5)
    (predicate2 object7)
    (predicate5 object1 object7)
    (predicate1 object1)
    (predicate2 object2)
    (predicate1 object2)
    (predicate2 object6)
    (predicate1 object6)
    (predicate2 object4)
    (predicate1 object4)
    (predicate3)
```

```
;; BEGIN ADD
    (not-allowed object1 object2)
;; END ADD
  )
  (:goal (and
    (predicate2 object5)
    (predicate5 object2 object5)
    (predicate2 object4)
    (predicate5 object7 object4)
    (predicate5 object6 object7)
    (predicate5 object1 object6)
    (predicate2 object3)
  ))
)
```

Listing 8: Prompt for LLM-as-Planner

```
You are a PDDL expert. Here is a game we are playing.
{domain_description}
{problem_description}
{constraint_description}

Write the plan that would solve this problem.

These are the available actions:
{available_actions}

Here is what the output should look like:
{example_answer}
```

Listing 9: Prompt for LLM-as-Formalizer, while generating the entire code

```
You are a PDDL expert. Here is a game we are playing.
{domain_description}
{problem_description}
{constraint_description}

Write the domain and problem files in minimal PDDL.

These are the available actions:
{available_actions}
```

Listing 10: Prompt for LLM-as-Formalizer, while first generating the non-constrained PDDL for the Edit setting

```
You are a PDDL expert. Here is a game we are playing.
{domain_description}
{problem_description}
Write the domain and problem files in minimal PDDL.

These are the available actions:
{available_actions}
```

Listing 11: Prompt for LLM-as-Formalizer, while modifying the non-constrained PDDL for the Edit setting

```
You are a PDDL expert. Here is a PDDL domain and problem file.
{original_domain_file}
{original_problem_file}

Modify the PDDL files so that it satisfies the following constraint: {constraint_description}
These are the available actions:
{available_actions}
```

Listing 12: Prompt for LLM-as-Formalizer, including revision

```
You are a PDDL expert. The following domain and problem files have the error: {error}

{domain_file}
{problem_file}

Revise the PDDL to remove the error.
```

Listing 13: Example Deepseek-R1 LLM-as-Planner output for State-based constraint: "Once you unstack a block, you cannot put it down on the table." Text in red represents actions that do not satisfy the newly added constraint and actions that do not satisfy BlocksWorld PDDL rules.

```
(unstack block1 block3)
(stack block1 block4)
(unstack block3 block6)
(stack block3 block5)
(unstack block6 block9)
(putdown block6)
(pickup block9)
(stack block9 block6)
(unstack block1 block4)
(stack block1 block9)
(unstack block2 block8)
(stack block2 block1)
(pickup block5)
(stack block5 block2)
(pickup block8)
(stack block8 block5)
(unstack block3 block5)
(stack block3 block8)
(pickup block4)
(stack block4 block3)
(pickup block7)
(stack block7 block4)
```

Listing 14: Example Deepseek-R1 LLM-as-Formalizer $\mathbb{DF}$ output for State-based constraint: "Once you unstack a block, you cannot put it down on the table." Text in green represents PDDL syntax that correctly encodes the new constraint.

```
(define (domain blocksworld)
  (:requirements :strips)
  (:predicates
    (ontable ?x)
    (on ?x ?y)
    (clear ?x)
    (handempty)
    (holding ?x)
    (unstacked ?x)
  )
  (:action pickup
    :parameters (?b)
    :precondition (and (ontable ?b) (clear ?b) (handempty))
    :effect (and (not (ontable ?b)) (not (clear ?b)) (not (handempty)) (holding ?b))
  )
  (:action putdown
    :parameters (?b)
    :precondition (and (holding ?b) (not (unstacked ?b)))
    :effect (and (ontable ?b) (clear ?b) (handempty) (not (holding ?b))))
  (:action stack
    :parameters (?b1 ?b2)
    :precondition (and (holding ?b1) (clear ?b2))
    :effect (and (on ?b1 ?b2) (not (clear ?b2)) (clear ?b1) (handempty) (not (holding ?b1)))
  )
  (:action unstack
    :parameters (?b1 ?b2)
    :precondition (and (on ?b1 ?b2) (clear ?b1) (handempty))
    :effect (and (holding ?b1) (clear ?b2) (not (on ?b1 ?b2)) (not (clear ?b1)) (not (handempty))
    (unstacked ?b1))))
```

Listing 15: Example Deepseek-R1 LLM-as-Formalizer $\mathbb{DF}$ output for Sequential constraint: "If you move block1, you must move block2 after." Text in red represents PDDL syntax that incorrectly encodes the new constraint as well as BlocksWorld PDDL rules.

```
(define (domain blocksworld)
  (:requirements :strips :equality)
  (:predicates
    (on ?x ?y) (clear ?x) (holding ?x) (armempty))
  (:action pickup
    :parameters (?b)
    :precondition (and (on ?b table) (clear ?b) (armempty))
    :effect (and (holding ?b) (not (on ?b table)) (not (clear ?b)) (not (armempty)))
  )
  (:action putdown
    :parameters (?b)
    :precondition (holding ?b)
    :effect (and (on ?b table) (clear ?b) (armempty) (not (holding ?b)))
  )
  (:action stack
    :parameters (?b ?under)
    :precondition (and (holding ?b) (clear ?under))
    :effect (and (on ?b ?under) (clear ?b) (armempty) (not (holding ?b)) (not (clear ?under)))
  )
  (:action unstack
    :parameters (?b ?under)
    :precondition (and (on ?b ?under) (clear ?b) (armempty) (not (= ?under table)))
    :effect (and (holding ?b) (clear ?under) (not (on ?b ?under)) (not (clear ?b)) (not (armempty)))
  )
)
```

Listing 16: Example Deepseek-R1 LLM-as-Planner output for Sequential constraint: "If you move block1, you must move block2 after." Text in green represents actions that satisfy the newly added constraints. The entire is satisfies BlocksWorld PDDL rules.

```
(pickup block3)
(stack block3 block6)
(pickup block9)
(stack block9 block3)
(pickup block12)
(stack block12 block10)
(pickup block4)
(stack block4 block12)
(pickup block8)
(stack block8 block15)
(pickup block7)
(stack block7 block11)
(pickup block13)
(stack block13 block7)
(pickup block1)
(stack block1 block13)
(pickup block2)
(stack block2 block8)
(pickup block5)
(stack block5 block1)
(pickup block14)
(stack block14 block5)
```

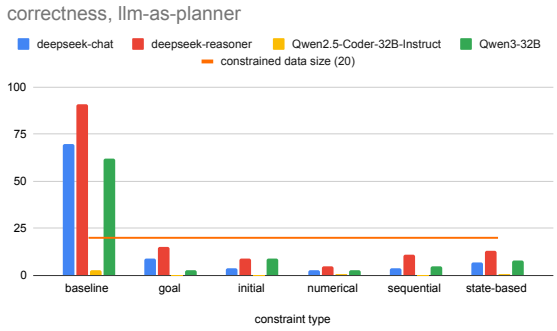

Figure 6: Natural BlocksWorld

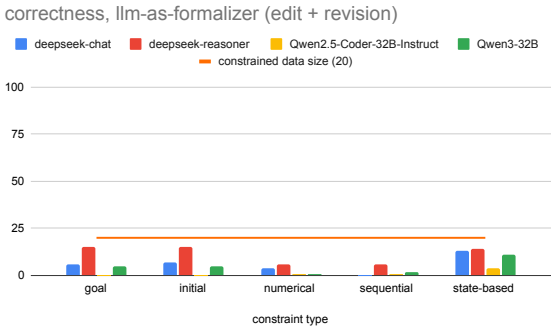

Figure 10: Natural BlocksWorld

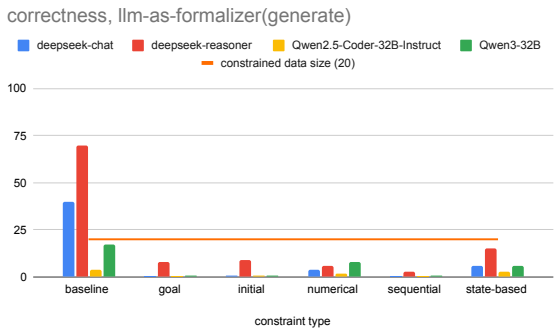

Figure 7: Natural BlocksWorld

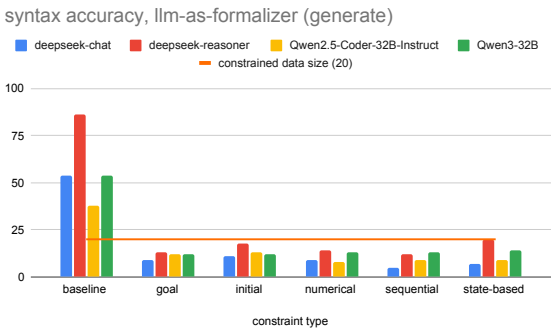

Figure 11: Natural BlocksWorld

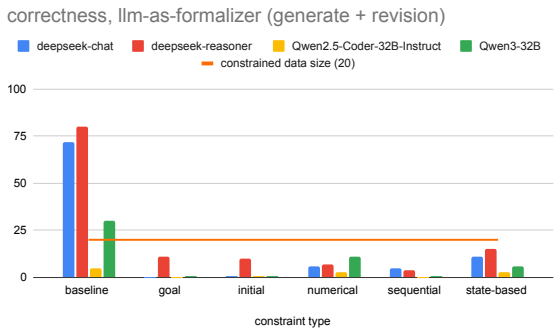

Figure 8: Natural BlocksWorld

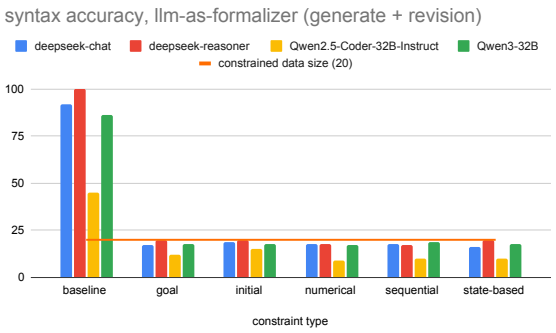

Figure 12: Natural BlocksWorld

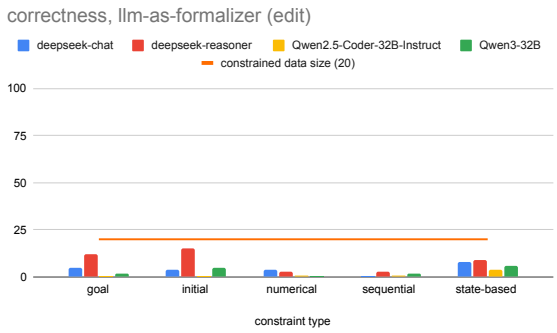

Figure 9: Natural BlocksWorld

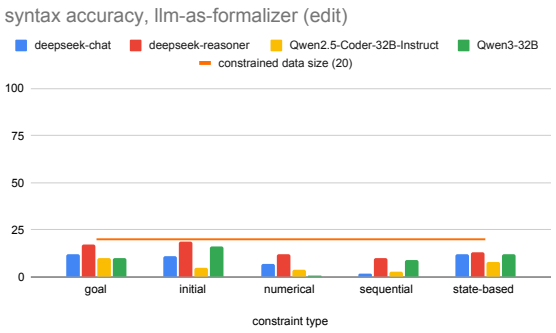

Figure 13: Natural BlocksWorld

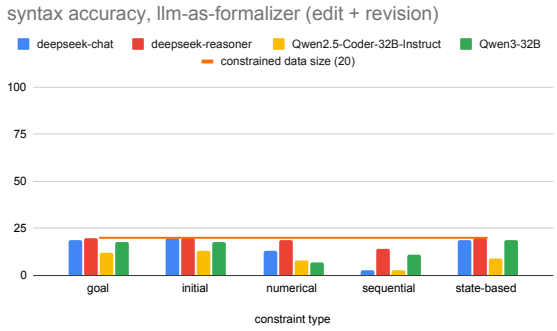

Figure 14: Natural BlocksWorld

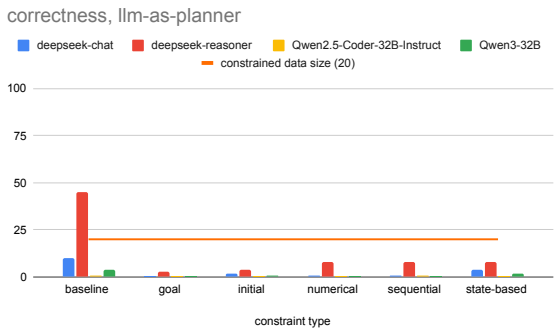

Figure 15: BlocksWorld-XL

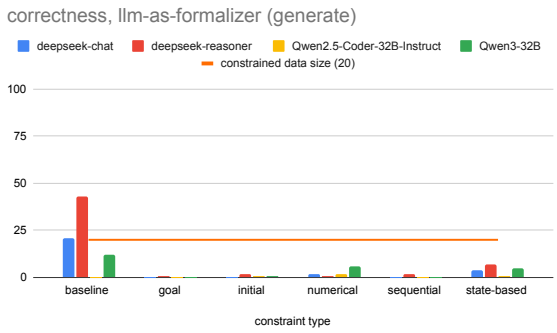

Figure 16: BlocksWorld-XL

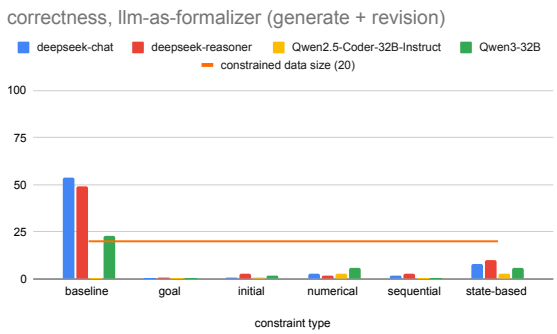

Figure 17: BlocksWorld-XL

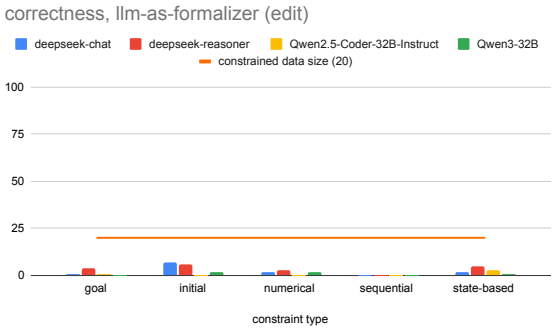

Figure 18: BlocksWorld-XL

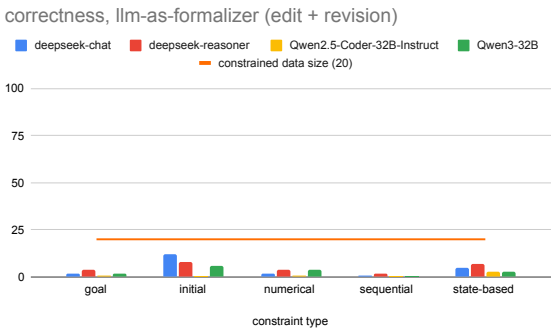

Figure 19: BlocksWorld-XL

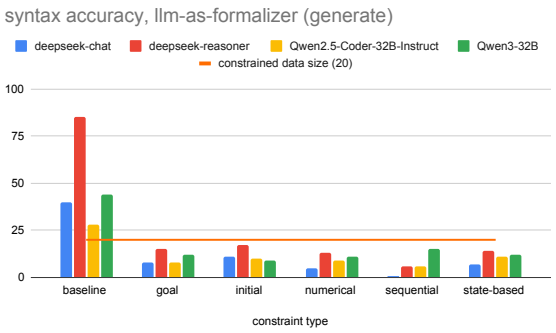

Figure 20: BlocksWorld-XL

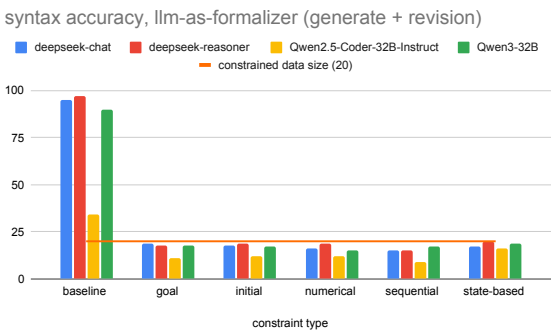

Figure 21: BlocksWorld-XL

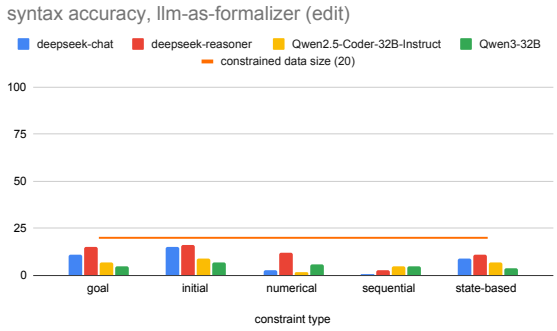

Figure 22: BlocksWorld-XL

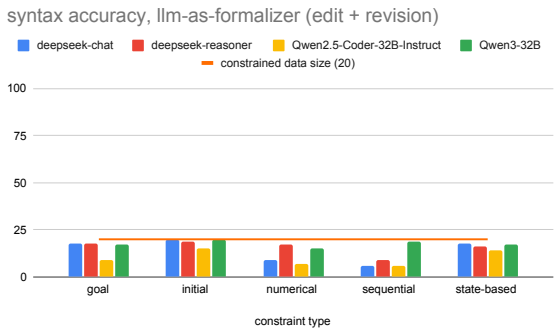

Figure 23: BlocksWorld-XL

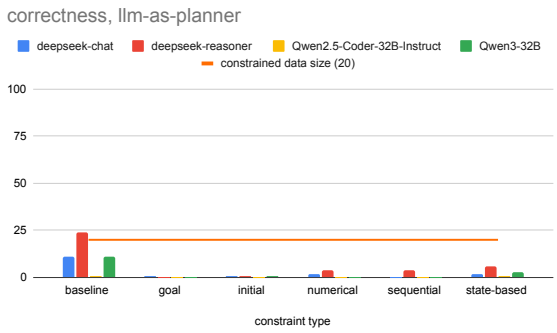

Figure 24: Mystery BlocksWorld

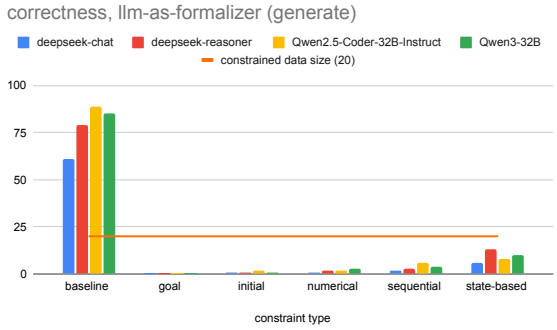

Figure 25: Mystery BlocksWorld

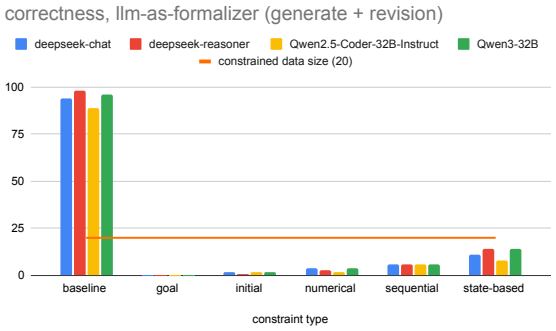

Figure 26: Mystery BlocksWorld

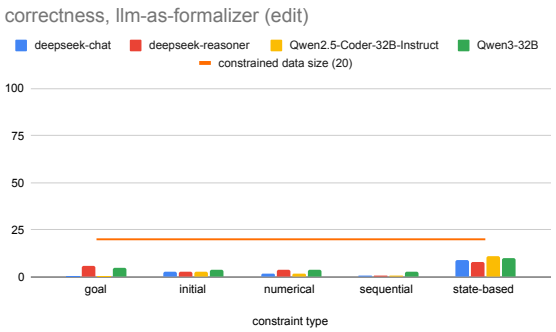

Figure 27: Mystery BlocksWorld

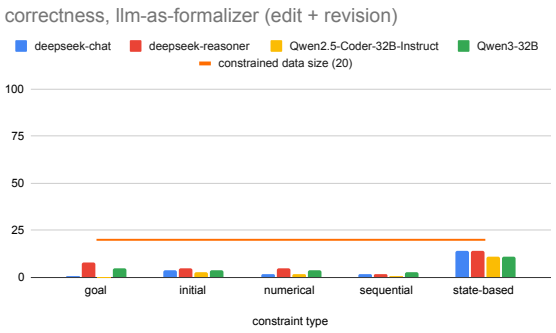

Figure 28: Mystery BlocksWorld

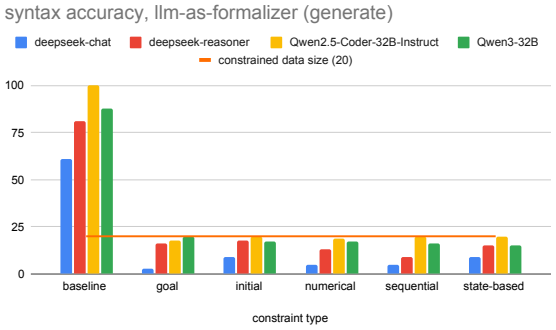

Figure 29: Mystery BlocksWorld

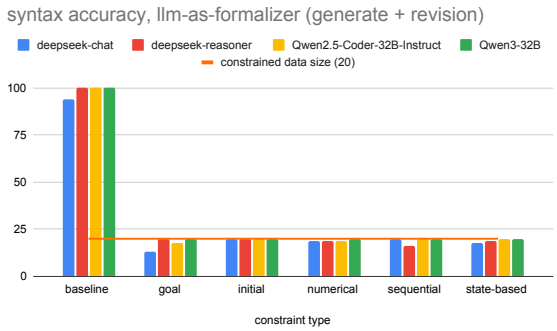

Figure 30: Mystery BlocksWorld

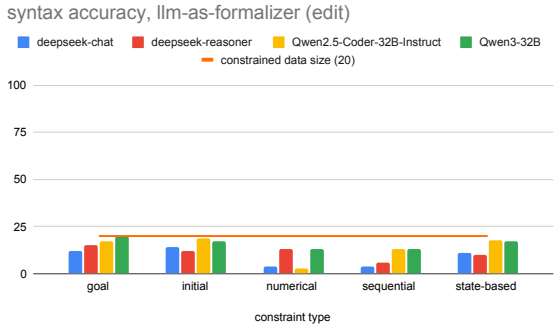

Figure 31: Mystery BlocksWorld

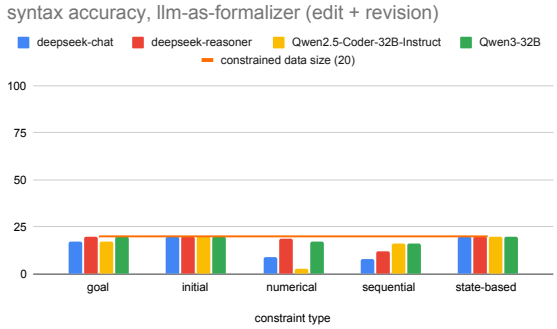

Figure 32: Mystery BlocksWorld