Evaluating Language Models Planning Capabilities on Goal Ordering Challenges

Eran Hirsch[1,2] **Guy Uziel**[1] **Ateret Anaby-Tavor**[1]
[1]IBM Research [2]Bar-Ilan University
eran.hirsch@ibm.com guy.uziel1@ibm.com atereta@il.ibm.com

Abstract

Planning involves the composition of primitive actions to achieve specific goals within a given environment. Classical planning research has well-established different types of goal-ordering challenges which have implications on the planning heuristics. In this study, we investigate the performance of Large Language Models (LLMs) in identifying if an order between two goals hold. We distinguish between three types of goal orderings challenges: reasonable, necessary, and optimal. Our findings reveal that LLMs predominantly struggle with reasonable goal ordering tasks compared to necessary and optimal goal orderings. Advancing this area could lead to improvements in the planning abilities of LLMs.

1 Introduction

Planning is a compositional reasoning problem that involves sequencing primitive actions to achieve specific goals within a given environment. Classical planning domains transform real-world problems into algorithmically analyzable formats [1, 2, 3]. Their extensive history and well-established resources offer an ideal platform for evaluating the compositional reasoning capabilities of Large Language Models (LLMs) [4, 5, 6, 7]. For example, Agarwal et al. [4] investigate the impact of many-shot examples on the Logistics domain, while Lehnert et al. [5] examine LLMs' ability to replicate the A* algorithm within the Sokoban domain. However, despite these efforts, LLMs frequently encounter difficulties with well-defined classical planning tasks [8, 9, 10]. This limitation highlights the pressing need for a deeper analysis of the planning capabilities of LLMs.

An important aspect of classical planning problems is to decide which goal to tackle first. For example, the LAMA planning system landmark heuristic [11] uses this information to evaluate if a subgoal needs to be achieved again. Previous research in classical planning has identified two primary types of goal orderings: reasonable and necessary [12, 3]. Notably, classifying whether a specific ordering holds between two goals has been proven to be PSPACE-hard [12]. In our study, we investigate LLMs' ability to solve various goal ordering challenges. This task involves identifying the primitive actions necessary to achieve a desired end goal, while also accounting for the interdependencies between subgoals. We explore four distinct classical planning domains, and distinguish between three types of goal orderings: reasonable, necessary, and optimal. Understanding how LLMs handle these different goal ordering tasks can provide significant insights into their compositional reasoning capabilities and limitations. Our analysis reveals that LLMs particularly struggle with reasonable goal ordering, highlighting a key area for future research that could enhance their performance.

2 Goal Ordering Experiments

The following experiments aim to evaluate whether LLMs can effectively prioritize goals in scenarios with inherent dependencies, evaluating the model's compositional ability under the constraint of many goals.

NeurIPS 2024 Workshop on Compositional Learning: Perspectives, Methods, and Paths Forward.

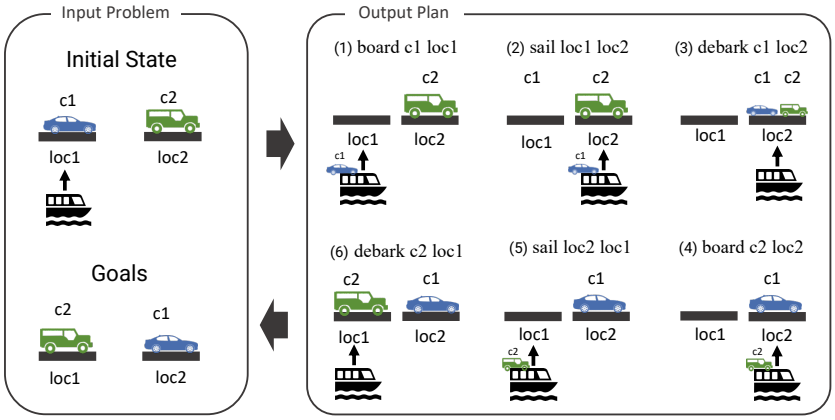

Figure 1: Example optimal goal ordering task. In the Ferry planning domain, a problem instance includes an initial state comprising the location of a ferry and several cars, with specified goals for placing the cars in specific locations (left). The ferry is capable of boarding a car and transporting it between locations. The goal ordering task requires the model to compare the two goals, and suggest that `car1 in location2` is the first goal to achieve in order to optimally solve this Ferry problem.

2.1 Task formulation

A state $s \in S$ represents the complete configuration of all objects at any given point, with S encompassing every conceivable configuration within the planning domain. An action $a \in A$ is applicable in state s if the state satisfies the action's preconditions, $s \models Pre(a)$, where $Pre(a)$ defines the necessary conditions for action applicability. Each action a is associated with an effect function $Eff(a) : S \rightarrow S$, which maps the current state to a new state resulting from the action's execution.

The input to the goal ordering task is an initial state $s_0 \in S$, a set of desired subgoals G, a description of the domain, and a task-specific instruction. The output of the goal ordering task is an ordered list of subgoals G', which reflects the sequence in which the subgoals should be achieved.

2.2 Experiment 1: Reasonable ordering

Reasonable ordering requires completing certain goals before others in order to achieve all objectives *simultaneously*. Formally, given two atomic goals $A, B \in G$, we say that $B \leq_r A$ iff for every state where goal A is achieved, there is no plan that achieves B without first destroying A [12].

For example, in the Blocksworld domain, blocks are stacked in towers, with the rule that only the top block of any stack can be moved at each step. Suppose the goals are to have on and on . Prioritizing the latter goal is considered a *reasonable* order, as completing it first is the only way to construct the tower. Otherwise, we must destroy one goal to achieve the other.[1]

2.3 Experiment 2: Necessary ordering

While reasonable ordering tests the ability to manage dependencies that arise when pursuing multiple goals simultaneously, necessary ordering evaluates the models' capacity to correctly identify that certain goals *cannot* be achieved before others. Formally, given two atomic goals $A, B \in G$, we say that $B \leq_n A$ iff for every state where goal A is achieved, goal B was already previously achieved.

For instance, in the Minigrid domain, an agent navigates a 2D map to pick up keys and unlock doors. Each step must be strategically planned to access progressively deeper sections of the map. Here, it becomes necessary to prioritize unlocking certain doors before others, as access to inner doors is dependent on first unlocking outer doors that block the path.

[1] In our experiments, we increase the complexity by using towers of at least four blocks, avoiding scenarios where the same block is mentioned in both goals.

Reasonable ordering

```
<DOMAIN_DESCRIPTION>
Given this state: <STATE>
and the following goals: <ALL_GOALS>
In order to complete the task all goals must be accomplished simultaneously.
Which one of the following goals needs to be achieved first?
The goals are: <TWO_GOALS>
In your answer only specify the goal with no explanation.
```

Necessary ordering

```
<DOMAIN_DESCRIPTION>
Given this state: <STATE>
Which one of the following goals can be achieved first?
The goals are: <TWO_GOALS>
In your answer only specify the goal with no explanation.
```

Optimal ordering

```
<DOMAIN_DESCRIPTION>
Given this state: <STATE>
Which one of the following goals can be achieved with fewer actions?
The goals are: <TWO_GOALS>
In your answer only specify the goal with no explanation.
```

Figure 2: Prompt templates used for the goal ordering experiments.

2.4 Experiment 3: Optimal ordering

We additionally identify an optimal ordering between goals. In optimal ordering, the current state determines the *most efficient* sequence of goals to minimize unnecessary actions. For example, as illustrated in Figure 1, with the ferry already at `location 1`, the optimal plan would involve boarding `car 1` and then sailing to `location 2`, rather than moving the ferry without `car 1`. The Ferry and Grippers domains are used in this experiment because they lack inherent necessary or reasonable orderings between goals, thus emphasizing the formulation of the most optimal plans based on the initial state.

3 Setup and Results

3.1 Models

We selected a diverse set of models for testing, including FALCON-180B, LLAMA-2-70B-CHAT, LLAMA-3-70B-INSTRUCT, CODE-LLAMA-34B-INSTRUCT, MISTRAL-7B-INSTRUCT-V0-2, MIXTRAL-8X7B-V0-1, and GPT-4 TURBO, chosen for their varying capacities and approaches in handling complex tasks [13, 14, 15, 16, 17, 18, 19].[2] To improve readability we use abbreviated names for the models throughout the paper rather than their full titles. The specific prompts used in these experiments are provided in Figure 2. All experiments were performed in a zero-shot fashion.

3.2 Dataset creation

We utilize classical planning domains, which transform real-world problems into structured formats suitable for algorithmic analysis. A widely used formalism in classical planning is the Planning Domain Definition Language (PDDL), developed to standardize the description of planning problems and domains [20]. We generate problem instances using the PDDL generators library [21]. In all experiments, the model is tasked with classifying which of two specific goals should be achieved

[2] In all experiments we used *gpt-4-0125-preview*.

Table 1: Success rate results for the goal planning experiments, testing for the model's ability to prioritize goals when imposed when imposed with reasonable, necessary or optimal ordering challenges.

Model	Reason. Blocks	Neces. Minigrid	Optimal Ferry	Optimal Grippers	Avg.
Falcon	.54	.55	.58	.54	.56
Mistral	.52	.69	.52	.66	.59
Mixtral	.54	.55	.56	.50	.53
Codellama	.58	.62	.54	.52	.53
Llama-2	.60	.59	.56	.60	.58
Llama-3	**.66**	.79	**.60**	**.80**	**.70**
GPT-4	.58	**.86**	.52	.54	.53

first (Section 2.1), establishing a baseline accuracy of 50% for these binary classification tasks. An example problem is provided in Figure 6.

To determine the correct goal order for evaluation purposes, we employ the landmarks graph generated by the LAMA planner [11]. The landmarks graph includes directed edges between subgoals, indicating the order in which they should be approached. These edges are labeled to denote the type of ordering, such as "n" for necessary and "r" for reasonable. By leveraging landmarks identified by the LAMA planner, our approach offers a novel method for constructing datasets that can systematically evaluate the goal ordering capabilities of LLMs.

3.3 Results

The results are detailed in Table 1. Among the models tested, Llama-3 demonstrated superior performance, outperforming the other models in two out of the three reasoning experiments. GPT-4 followed, but surpisingly underperformed in several domains, achieving low results outside of necessary goal ordering.

Notably, **LLMs struggle significantly with the reasonable ordering task**, as all models failed to achieve success rates meaningfully above the 50% baseline. This finding underscores a significant challenge for LLMs in managing goal dependencies, particularly in scenarios where a logical sequence between subgoals is necessary but not explicitly enforced, such as in the case of constructing a tower of blocks.

Interestingly, the two optimal ordering experiments, which utilized very similar domains differing primarily in their description of actions, yielded significantly different results. This discrepancy suggests that **the compositional reasoning capabilities of the models are sensitive to how the problem is described**. We conjecture that embodied agents problems, such as Grippers, are more common in the training data of LLMs.

4 Conclusions

In this work, we explored the differences between various classical planning domains through the lens of goal ordering. Our analysis of multiple goal ordering techniques revealed that reasonable goal ordering are more challenging for LLMs compared to those involving necessary or optimal orderings. Future research could focus on developing more advanced planning techniques designed to address these specific challenges and improve LLM performance in reasonable goal ordering tasks.

References

[1] Avrim L Blum and Merrick L Furst. Fast Planning Through Planning Graph Analysis. 1997.

[2] Joerg Hoffmann. FF: The Fast-Forward Planning System. *AI Magazine*, 22(3):57–57, September 2001.

[3] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22:215–278, November 2004.

[4] Rishabh Agarwal, Avi Singh, Lei M. Zhang, Bernd Bohnet, Stephanie Chan, Ankesh Anand, Zaheer Abbas, Azade Nova, John D. Co-Reyes, Eric Chu, Feryal Behbahani, Aleksandra Faust, and Hugo Larochelle. Many-shot in-context learning, 2024.

[5] Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping, 2024.

[6] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a llm, 2023.

[7] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with Language Model is Planning with World Model, May 2023.

[8] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency, May 2023.

[9] Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Can large language models really improve by self-critiquing their own plans?, 2023.

[10] Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change, November 2023.

[11] Silvia Richter and Matthias Westphal. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, September 2010.

[12] J. Hoffmann and J. Koehler. On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm. *Journal of Artificial Intelligence Research*, 12:338–386, June 2000.

[13] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The falcon series of open language models, 2023.

[14] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[15] AI@Meta. Llama 3 model card. 2024.

[16] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[17] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[18] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024.

[19] OpenAI. GPT-4 Technical Report, March 2023.

[20] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - The Planning Domain Definition Language.pdf, 1998.

[21] Jendrik Seipp, Álvaro Torralba, and Jörg Hoffmann. PDDL generators. `https://doi.org/10.5281/zenodo.6382173`, 2022.

A PDDL

```
(define (domain ferry)
    (:predicates (at-ferry ?l) (at ?c ?l) (empty-ferry) (on ?c))

    (:action sail
        :parameters (?from ?to)
        :precondition (at-ferry ?from)
        :effect (and (at-ferry ?to) (not (at-ferry ?from))))


    (:action board
        :parameters (?car ?loc)
        :precondition (and (at ?car ?loc) (at-ferry ?loc) (empty-ferry))
        :effect (and (on ?car) (not (at ?car ?loc)) (not (empty-ferry))))

    (:action debark
        :parameters (?car ?loc)
        :precondition (and (on ?car) (at-ferry ?loc))
        :effect (and (at ?car ?loc) (empty-ferry) (not (on ?car)))))
```

Figure 3: The PDDL Ferry domain definition. The domain definition specifies the predicates and actions, encapsulating the physics of the domain.

Classical planning is based on the notion that the entire planning domain and problem instance are described in a formalised, machine-readable format. One common format is the Planning Domain Definition Language (PDDL; 20). The domain definition (Figure 3) encodes the physics of the world, while the problem instance definition (Figure 4) specifies the initial state and desired goals, thereby customizing the domain to a specific scenario. The planning task entails generating a sequence of actions that facilitate a transition from the initial state to a goal state, thereby constituting a plan.

```
(define (problem ferry-l3-c2)
    (:domain ferry)
    (:objects l0 l1 l2 c0 c1
        )
    (:init
        (empty-ferry)
        (at c0 l1)
        (at c1 l2)
        (at-ferry l2)
    )
    (:goal
        (and
            (at c0 l0)
            (at c1 l0)
        )
    )
)
```

Figure 4: An example PDDL problem instance definition for the Ferry domain.

A domain consists of a pair `<P,A>`, where `P` represents a set of *predicates*, and `A` is a set of *actions*. Each predicate $p \in P$ includes a name and a set of variables denoted by a question mark. For instance, `at-ferry ?l` is a predicate, while `at-ferry(loc1)` is a specific truth-assignment to the predicate, indicating the ferry's presence at location 1. Predicates can also be negated (e.g., `not(at-ferry(loc1))`). An action $a \in A$ consists of a name, a set of variables, a set of effect predicates, and a set of precondition predicates. For instance, `board(?car ?loc)` is an action denoting the boarding of car `?car` on the ferry at location `?loc`. The action's preconditions include predicates such as `at_ferry(?loc)`, `at(?car,?loc)`, and `empty_ferry`, signifying that both the ferry and the car ?car are at location `?loc`, and that the ferry is empty. The action's effects include predicates such as `not(empty_ferry)`, `not(at(?car,?loc))`, and `on(?car)`, indicating that the ferry is no longer empty, that the car `?car` is no longer at location `?loc`, and that the car `?car` is now on the ferry.

A problem is a triple `<O,I,G>`, where `O` is a set of *objects*, `I` is a set of truth-assigned predicates that are currently true in the world model, and `G` is a set of truth-assigned predicates designated to be achieved.

B Experiment Details

In Section 3 we described a set of controlled experiments which test for specific reasoning abilities of LLM-based planners. In Appendix B.1 we provide more details about the prompts used.

B.1 Prompts

Our prompt is constructed as following: for each domain we start by describing the domain in natural language, such as exemplified by Figure 5. Then, we use the relevant prompt for each experiment (Figure 2).

```
We are dealing with the Blocksworld problem. In this domain we have 4 possible
    actions:
1. pickup - (?ob - object)
 - Preconditions: The object (?ob) must be clear, on the table, and the arm
     must be empty.
 - Effects: After executing the action, the object is now held, not clear, and
     not on the table.
 - Example: If you execute (pickup blockA), it means you pick up "blockA" from
     the table.
2. putdown - (?ob - object)
 - Preconditions: The object (?ob) must be currently held.
 - Effects: After executing the action, the object is now clear, the arm is
     empty, and the object is on the table.
 - Example: If you execute (putdown blockB), it means you put down "blockB" on
     the table.
3. stack - (?ob - object, ?underob - object)
 - Preconditions: The object that you want to stack (?ob) must be held, and
     the object underneath (?underob) must be clear.
 - Effects: After executing the action, the arm is empty, the stacked object
     (?ob) is clear, and it is now on top of the underneath object (?underob).
     The underneath object is no longer clear.
 - Example: If you execute (stack blockC blockD), it means you stack "blockC"
     on top of "blockD".
4. unstack - (?ob - object, ?underob - object)
 - Preconditions: The object that you want to unstack (?ob) must be on top of
     another object (?underob), and it (?ob) must be clear. Additionally, the
     arm must be empty.
 - Effects: After executing the action, the object (?ob) is now held, the
     underneath object (?underob) is clear, and the relationship "on" between
     (?ob) and (?underob) is broken. Also, (?ob) is no longer clear, and the
     arm is not empty.
 - Example: If you execute (unstack blockC blockD), it means you unstack
     "blockC" from on top of "blockD".
```

Figure 5: Explanation provided about the Blocksworld domain before each Blocksworld experiment.

B.2 Examples

C Domains Characteristics

C.1 Domains

Blocksworld. In this domain, the objects are blocks and the agent is a robotic arm that can pick them up and put them down on a table. Blocks can be stacked, and only a block that is clear (i.e., doesn't have a block on top of it) can be picked up. The goal predicates are to stack blocks on top of other blocks.

Ferry. In this domain the objects are cars and locations, and the agent is a ferry that can move between locations, board a car in one location or debark it in another location. The ferry can only

```
We are dealing with the Blocksworld problem. In this domain we have 4 possible
    actions:
...

In your answer only specify the action with no explanation.

Given this state:
b5 is on b4 and is clear,
b1 is on b2,
b2 is on b3,
b3 on-table,
b4 is on b1,
arm is empty

and the following goals:
on b1 b2,
on b2 b5,
on b3 b4,
on b5 b3

In order to complete the task both goals must be accomplished simultaneously.
Which one of the following goals needs to be achieved first?
The goals are:
on b1 b2, on b3 b4


In your answer only specify the goal with no explanation
<ANSWER>
on b3 b4
```

Figure 6: Example reasonable ordering experiment from the Blocksworld domain.

board one car at a time. The problem requires the ferry to transport the cars to their designated locations.

Grippers. In this domain the objects are balls and rooms, and the agent is a robot that can move between rooms, pick-up balls or drop a picked-up ball. The robot can hold two balls simultaneously with its left and right grippers. The problem requires the robot to transport the balls to their designated rooms.

Depots. Similar to Blocksworld, the objects are crates and the agent is a hoist that can pick them up or put them down. The difference from Blocksworld is that there are multiple "tables", which are called pallets, and each pallet has its own hoist. The pallets are located in different locations, called depots and distributors. The truck is another agent that can move the crates between locations. This is similar to Ferry and Grippers. The goal predicates are to stack the crates at specific locations, or on top of other crates.

Minigrid. In this grid domain, the objects are walls, keys and doors, and the agent is a robot that can traverse the grid, pick up keys and unlock doors. Doors can be locked, and their locks have certain shapes with keys having matching shapes. Only one key can be picked up at a time, thus the robot must first drop a key in order to pick up a different key. The more number of shapes there are, the more often the robot has to switch the key it is holding. The goal predicate is to reach a certain location in the board. The floor plan that we used for all Minigrid problems is depicted in Figure 7.

C.2 Domains Modifications

Some domains have a lot of predicates to describe the current state, thus creating a very long prompt for the LLM. This is a problem mostly in the Minigrid domain, which as seen in Figure 7, has a large 2D map which contains 64 places. This map is meticulously described in the PDDL format.

To sidestep this problem of having extremely long prompts, we make several changes to their PDDL definition which allow us to create a shorter state representation. Firstly, we remove predicates that contain typing information of objects. For example, we remove predicates that mention that the object `p1` is a place (`place p1`), that the object `key1` is a key (`key key1`), and that the object `shape1` is a shape (`shape shape1`). Given this change, it should still be possible to calculate applicable actions, as LLMs have different representations for each token, and they can infer the type of the object from the prefix, context and provided examples. In addition, we remove predicates which contain information about connected paths, such as `p1` is connected to `p2` (`conn p1 p2`). Originally, this is used in Minigird to represent the locations of walls. We remove this only for the fine-tuned models which can learn the map during training. Finally, in Minigrid, we remove predicates which indicate that a door is open (e.g., `open p3`). The predicates already describe which doors are locked (e.g., `locked p16`), and it should be possible for the LLM to see which places are locked and from it to infer the open doors.

Figure 7: Minigrid floorplan that we used for generating problem instances. L represents locked doors, w represents blocked places which can't be crossed, and all other dots represent places where the agent can walk. In each problem instance, the robot start and end location are chosen randomly. The keys are spread around the map, not depicted in this figure.