# Towards More Likely Models for AI Planning

Anonymous submission


## Abstract

This is the first work to look at the application of large language models (LLMs) for model space edits in automated planning tasks. We look at two quintessential model-space reasoning tasks: unsolvability and explanations. We empirically demonstrate how the performance of an LLM contrasts with combinatorial search (CS) – an approach that has been traditionally used to solve model space tasks in planning – with the increasing complexity of model edits and the increasing complexity of plans, both with the LLM in the role of a standalone model-space reasoner as well as in concert with the CS approach as part of a two-stage process. Our experiments show promising results suggesting further forays of LLMs into the exciting world of model space reasoning for planning tasks in the future.


## 1 Introduction

AI planning or automated planning (used interchangeably) is the task of synthesizing the goal-directed behavior of autonomous agents. Traditionally, the AI planning community has looked at the classical planning problem as one of generating a plan given a model of the world (Ghallab, Nau, and Traverso 2004). Here, "model" or a "planning problem" refers to a collection of constraints describing the state of the world (initial state), the actions available to the agent along with the conditions under which the agent can do those actions and the effect of doing those actions on the environment, and a target (goal) state for the agent to achieve. The plan is a sequence of actions with which the agent can transform the state of its world to the desired goal state.

Typically, these models are represented using the planning domain definition language or PDDL (Haslum et al. 2019; McDermott et al. 1998) – we will use the same for our studies in this paper. All the information to derive this solution (plan) is contained in the input model which remains static during the planning task. *But what if the model itself needs to be changed?* This may be because it is incorrect, or incomplete, or even unsolvable. It may be because it needs to be changed to support some new behaviors. It may also be because the model is being used to describe a world that itself needs to change through the actions of an agent. In practice, the deployment of systems that can plan involves a whole gamut of challenges in authoring, maintaining, and meta-reasoning about models of planning tasks.

This realization has led to the development of a large class of methods that can be broadly referred to as model space

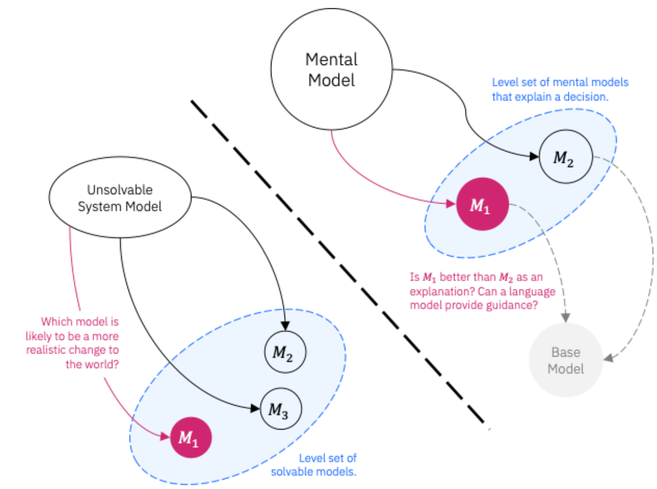


Figure 1: A conceptual illustration of model space problems in AI planning. Instead of the classical planning task of computing a plan given a model, a model space task starts with a starting model $\mathcal{M}$ and a target criterion to satisfy, and the solution is a new model $\mathcal{M}_1$ where that criterion is satisfied. That criterion on the left is that the initially unsolvable model becomes solvable (or an initially invalid plan in $\mathcal{M}$ becomes valid in the new model $\mathcal{M}_1$). On the other hand, the starting model on the right is the mental model of the user that needs to be updated and the target is a new model that can explain a given plan (or refute a given foil).

problems. All of them involve a starting model which has something wrong with it and the solution is a new model where the problem has been resolved or the required criterion has been met. Most existing solutions to these problems leverage some form of search over the space of models. While these methods can provide soundness guarantees with regard to the underlying criteria, they generally overlook the fact that not all models that satisfy the criteria are equally preferred with respect to the end user. Preference for a specific model is domain and task-dependent and as such an aspect overlooked by most work in this area. However, the emergence of pre-trained LLMs, provides us with a unique opportunity to identify potentially preferred model updates in at least the set of "real-worldly" domains. Broadly speaking, domains that cover all manner of human enterprise – and consequently (planning) models describing them wherever relevant (sequential decision-making tasks) – that are

| Problem | LLM-only | | | | LLM + Search | | | |
|---|---|---|---|---|---|---|---|---|
| | GPT-3.5-turbo | | GPT-4 | | GPT-3.5-turbo | | GPT-4 | |
| | Sound | Preferred | Sound | Preferred | Sound | Preferred | Sound | Preferred |
| Unsolvability | 7/18 | 2/7 | 14/18 | 10/14 | %100 | 3/14 | %100 | 11/14 |
| Explanation | 0/5 | 0/5 | 0/5 | 0/5 | %100 | 4/5 | %100 | 1/5 |

Table 1: A set of preliminary results on the use of LLM to direct model space search. In particular, we focus on creating solvable variants in a simple travel domain and the problem of generating model reconciliation explanation for blocksworld.

described on the public internet i.e. domains concerning our worldly matters. In this report, we share some preliminary insights into how well a state-of-the-art pre-trained LLM can help automatically select models and model updates that may be preferred for a given context.

## 2 Model Space Problems in AI Planning

While the literature has considered many different types of model space problem, this paper will focus on two main ones, namely, unsolvability and explanation generation. For readers new to the subject, we provide a conceptual illustration of these topics in Figure 1.

**Unsolvability** Perhaps the most difficult of model space problems, especially with humans in the loop, is that of unsolvability. This is because when a model is unsolvable, there is no artifact (such as an outputted plan) to look at for debugging purposes. Efforts at addressing this issue has included excuse generation (Göbelbecker et al. 2010; Herzig et al. 2014) and using explanations as a way to empower users to correct the model (Sreedharan et al. 2020, 2019).

**Explanations** While, for unsolvability, we deal with one model in isolation, when working with humans in the loop, AI systems are often required to provide explanations of their behavior. This task is formulated as one of "model reconciliation" (Chakraborti et al. 2017) – an explanation is the model update that justifies a particular plan i.e. if both models justify a plan then there is no need for explanations. For the purposes of this paper, we will focus on cases where the system needs to ensure that the plan will be optimal in the updated model so as to refute all possible foils.

As we mentioned previously, state-of-the-art solutions for both these problems use CS to end up with many logically equivalent solutions with no guidance on the likelihood of each in the context of the domain – we know they are not equally likely when presented to the user (Zahedi et al. 2019; Miller 2019). In the following, we will explore whether we can leverage a pre-trained LLM to give us such guidance.

## 3 Preliminary Results

In our evaluation, we will make use of LLMs in two separate capacities. Firstly, to test whether LLMs can act as direct model space reasoner and second it's ability to filter out preferred solutions from a given set of solutions generated through a combinatorial search. For the first setting, we measure whether the solution is sound (i.e., whether updated model is solvable or if the generated explanation is valid) and whether the generated solution will be considered a likely or preferred one. In the case of explanation, there are works that have helped establish guidelines on what explanation would be preferred over other (cf. (Zahedi et al. 2019)), but this less clear in the case of unsolvability. To address this we created a novel planning domain, which implicitly encodes commonsense constraints about what model-changes may be more reasonable. In particular, we created a travel domain where the agent needs to go from a source city to a destination ones. The domain definition includes information about what cities are neighboring and only includes bus and taxi service between neighboring cities. Next, we created a set of unsolvable problems such that they can be made solvable by only starting taxi or bus service between neighboring cities. First row of table 1, presents the results from the two settings. In the case of the LLM + Search setting for unsolvability, we used a model space search that returned the first twenty minimal solution (measured in terms of the number of model updates), while disallowing any solutions that are supersets of previously identified solutions. The model updates were additionally restricted to just initial state changes. In total, we considered 18 problems with total number of objects varying from nine to twenty and optimal plans ranging from two to seven steps. The problems were made unsolvable by randomly deleting a subset of taxi and bus services between cities. Out of the 18 problems, we found that the bounded search was only able to retrieve a solution that would considered reasonable per the domain constraints in only 14 problems. This further highlights one of the challenges of model space search, namely the extremely large space of potential solutions and also points to the need to incorporate feedback from the LLM into the search process itself. For most problems where the search was able to identify a solution set that included a reasonable solution, GPT-4 seems to be able to identify a preferred solution correctly outperforming LLM only strategy. However we would caution against making a strong conclusion here given the small dataset.

For explanation, we considered the Blocks World domain. We created a human model by randomly deleting a number of model components from the original IPC domain model (a total of 9 edits). We considered five different problems in the domain and considered multiple explanations of equal length for each domain. Following Zahedi et al. (2019), we consider an explanation to be preferred over others if it contains higher number of effect updates. For purely LLM based generation, we provided robot model, human model and plan as part of the prompt and asked it to generate an explanation in terms of model difference. In general, we saw that LLMs were quite bad at generating sound explanation (in so that these are model differences that exist and whose inclusion in the human model will render the target plan optimal), let alone select preferred explanation. We again see a small uptick when we combine search with LLMs as a mechanism to select explanations from the list.

All results listed in this paper are meant to be a set of initial experiments to evaluate the possibility of using LLMs to aid model space search. Going forward, we hope to replicate the experiments in a number of additional domains and model space tasks. We would also like to investigate other potential ways of incorporating LLMs in model search tasks. Example prompts used are provided in the appendix.

## References


Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *IJCAI*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Elsevier.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming Up With Good Excuses: What to do When no Plan Can be Found. In *ICAPS*.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. An Introduction to the Planning Domain Definition Language. *Synthesis Lectures on Artificial Intelligence and Machine Learning*.

Herzig, A.; de Menezes, M. V.; De Barros, L. N.; and Wassermann, R. 2014. On the Revision of Planning Tasks. In *ECAI*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR98003/DCS TR1165, New Haven, CT: Yale Center for Computational Vision and Control.

Miller, T. 2019. Explanation in Artificial Intelligence: Insights from the Social Sciences. *Artificial intelligence*.

Sreedharan, S.; Chakraborti, T.; Muise, C.; Khazaeni, Y.; and Kambhampati, S. 2020. D3WA+ – A Case Study of XAIP in a Model Acquisition Task for Dialogue Planning. In *ICAPS*.

Sreedharan, S.; Srivastava, S.; Smith, D.; and Kambhampati, S. 2019. Why Can't You Do That HAL? Explaining Unsolvability of Planning Tasks. In *IJCAI*.

Zahedi, Z.; Olmo, A.; Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2019. Towards Understanding User Preferences for Explanation Types in Explanation as Model Reconciliation. In *HRI Late Breaking Report*.

# 4 Sample Prompts

## 4.1 LLM only Setting for Unsolvability

```
"given the following problem and
domain files: " + domain_content
+ "," + problem_content +
"Come up with most reasonable changes
that you can make to the initial state
that will make it solvable.
I want you to only list the new
initial states without any
explanation or additional sentences
in the beginning.Just follow this
rule: (:init...)"
domain_content:
(define (domain domaingotocity)
(:requirements :typing)
(:types city - object)
(:predicates
    (at ?x - city)
    (has_taxi ?x ?y - city)
    (has_bus ?x ?y - city)
    (neighboring ?x ?y - city)
)
(:action use_taxi
    :parameters (?from ?to - city)
    :precondition (and
      (at ?from)
      (has_taxi ?from ?to)
    )
    :effect (and
      (not (at ?from))
      (at ?to)
    )
  )
(:action use_bus
    :parameters (?from ?to - city)
    :precondition (and
      (at ?from)
      (has_bus ?from ?to)
    )
    :effect (and
      (not (at ?from))
      (at ?to)
    )
  )
)
problem_content:
(define (problem problemgotocity)
    (:domain domaingotocity)
    (:objects city_a - city
    city_b - city
    city_c - city city_d - city
    city_e - city
    city_f - city city_j - city
    city_l - city
    city_o - city city_r - city
    city_s - city
    city_t - city city_v - city
    city_x - city)
    (:init
    (at city_a)
    (has_bus city_a city_b)
    (has_bus city_a city_d)
    (has_bus city_d city_j)
    (has_bus city_l city_v)
    (has_bus city_t city_e)
    (has_taxi city_e city_o)
    (has_taxi city_e city_x)
    (has_taxi city_f city_s)
    (has_taxi city_r city_l)
    (has_taxi city_s city_c)
    (neighboring city_a city_b)
    (neighboring city_a city_d)
    (neighboring city_b city_c)
    (neighboring city_d city_j)
    (neighboring city_e city_o)
    (neighboring city_e city_x)
    (neighboring city_f city_s)
    (neighboring city_l city_v)
    (neighboring city_r city_l)
    (neighboring city_s city_c)
    (neighboring city_t city_e)
    (neighboring city_t city_o))
    (:goal (at city_c))
    )
output:
(:init (at city_a)
(has_bus city_a city_b)
(has_bus city_a city_d)
(has_bus city_d city_j)
(has_bus city_l city_v)
(has_bus city_t city_e)
(has_taxi city_e city_o)
(has_taxi city_e city_x)
(has_taxi city_f city_s)
(has_taxi city_r city_l)
(has_taxi city_s city_c)
(neighboring city_a city_b)
(neighboring city_a city_d)
(neighboring city_b city_c)
(neighboring city_d city_j)
(neighboring city_e city_o)
(neighboring city_e city_x)
(neighboring city_f city_s)
(neighboring city_l city_v)
(neighboring city_r city_l)
(neighboring city_s city_c)
(neighboring city_t city_e)
(neighboring city_t city_o)
(at city_o)
(at city_x))
```

## 4.2 LLM + Search Setting for Unsolvability

```
"Given the following problem,
domain files, and options list:
- Problem: {uns_problem_string}
- Domain: {domain_string}
- Options: {option_list}
Pick the most reasonable option
from the list that you can apply
```

```
to theinitial state to make the
problem solvable. Only provide
the number of the option selected
and no other information
(exclude even the term option)."

uns_problem_string:
(define (problem problemgotocity)
    (:domain domaingotocity)
    (:objects city_a - city
    city_b - city city_c - city
    city_d - city city_e - city
    city_f - city city_j - city
    city_l - city city_o - city
    city_r - city city_s - city
    city_t - city city_v - city
    city_x - city)
    (:init (at city_a)
    (has_bus city_a city_b)
    (has_bus city_a city_d)
    (has_bus city_d city_j)
    (has_bus city_l city_v)
    (has_bus city_t city_e)
    (has_taxi city_e city_o)
    (has_taxi city_e city_x)
    (has_taxi city_f city_s)
    (has_taxi city_r city_l)
    (has_taxi city_s city_c)
    (neighboring city_a city_b)
    (neighboring city_a city_d)
    (neighboring city_b city_c)
    (neighboring city_d city_j)
    (neighboring city_e city_o)
    (neighboring city_e city_x)
    (neighboring city_f city_s)
    (neighboring city_l city_v)
    (neighboring city_r city_l)
    (neighboring city_s city_c)
    (neighboring city_t city_e)
    (neighboring city_t city_o))
    (:goal (at city_c))
)
domain_string:
(define (domain domaingotocity)
    (:requirements :typing)
    (:types city)
    (:predicates (at ?x - city)
    (has_bus ?x - city ?y - city)
    (has_taxi ?x - city ?y - city)
    (neighboring ?x - city
    ?y - city))
    (:action use_bus
        :parameters
        (?from - city ?to - city )
        :precondition
        (and (at ?from)
        (has_bus ?from ?to))
        :effect
        (and (not (at ?from))
        (at ?to))
    )
     (:action use_taxi
        :parameters
        (?from - city ?to - city )
        :precondition (and (at ?from)
        (has_taxi ?from ?to))
        :effect (and (not (at ?from))
        (at ?to))
    )
)

options:

["Option 1: {'has_taxi city_a city_c'}",
"Option 2: {'has_taxi city_b city_c'}",
"Option 3: {'has_bus city_d city_c'}",
"Option 4: {'has_taxi city_a city_s'}",
"Option 5: {'has_bus city_a city_s'}",
"Option 6: {'has_bus city_b city_c'}",
"Option 7: {'has_taxi city_b city_s'}",
"Option 8: {'has_taxi city_d city_s'}",
"Option 9: {'has_bus city_b city_f'}",
"Option 10: {'at city_s'}",
"Option 11: {'has_bus city_a city_c'}",
"Option 12: {'has_bus city_a city_f'}",
"Option 13: {'has_taxi city_j city_s'}",
"Option 14: {'has_bus city_j city_f'}",
"Option 15: {'has_bus city_d city_f'}",
"Option 16: {'has_taxi city_d city_c'}",
"Option 17: {'has_taxi city_j city_f'}",
"Option 18: {'at city_f'}",
"Option 19: {'has_bus city_d city_s'}",
"Option 20: {'has_bus city_j city_s'}"]

output:
4
```

### 4.3 LLM only Setting for Explanations

```
"Consider a situation, where your
understanding of a task may be
best represented by the following
pddl files:
- Problem: {problem_string}
- Model: {original_domain}
- The optimal solution for the
task is given as: {initial_opt_plan}
If a user's understanding of the task
is given as
- Problem: {problem_string}
- Domain: {uns_domain_string}
come up with the set of model updates
that will help the user understand
why the plan is optimal. Ensure that
these updates not only enable the
execution of the plan but also ensure
its optimality within the modified model.
Here is example model updates you can
give to me: {examples_for_the_prompt}
You need to come up with exactly
{budget} edits. All the edits need to
be stored in one list with opening
and closing square brackets, where each
element corresponds to one edit. Your
```

response should only be this list,
a list with opening and closing
square brackets.No further explanation
is required."

problem_string:
```
(define (problem BLOCKS-8-1)
(:domain BLOCKS)
(:objects B A G C F D H E - block)
(:init (CLEAR E) (CLEAR H) (CLEAR D)
(CLEAR F) (ONTABLE C) (ONTABLE G)
 (ONTABLE D) (ONTABLE F) (ON E C)
 (ON H A) (ON A B) (ON B G)
 (HANDEMPTY))
(:goal (and (ON C D) (ON D B)
(ON B G) (ON G F) (ON F H)
(ON H A) (ON A E)))
)
```
original_domain:
```
(define (domain BLOCKS)
  (:requirements :strips
  :typing)
  (:types block)
  (:predicates
    (on ?x - block ?y - block)
    (ontable ?x - block)
    (clear ?x - block)
    (handempty)
    (holding ?x - block)
   )

  (:action pickup
     :parameters (?x - block)
     :precondition (and (clear ?x)
     (ontable ?x) (handempty))
     :effect
     (and (not (ontable ?x))
       (not (clear ?x))
       (not (handempty))
       (holding ?x)))

  (:action putdown
     :parameters (?x - block)
     :precondition
     (and (holding ?x))
     :effect
     (and (not (holding ?x))
       (clear ?x)
       (handempty)
       (ontable ?x)))
  (:action stack
     :parameters
     (?x - block ?y - block)
     :precondition
     (and (holding ?x)
     (clear ?y))
     :effect
     (and (not (holding ?x))
       (not (clear ?y))
       (clear ?x)
       (handempty)
       (on ?x ?y)))

  (:action unstack
     :parameters (?x - block ?y - block)
     :precondition
     (and (on ?x ?y)
     (clear ?x) (handempty))
     :effect
     (and (holding ?x)
       (clear ?y)
       (not (clear ?x))
       (not (handempty))
       (not (on ?x ?y))))
)
```

initial_opt_plan:
```
    (unstack e c)
    (putdown e)
    (unstack h a)
    (stack h c)
    (unstack a b)
    (stack a e)
    (unstack h c)
    (stack h a)
    (pickup f)
    (stack f h)
    (unstack b g)
    (stack b c)
    (pickup g)
    (stack g f)
    (unstack b c)
    (stack b g)
    (pickup d)
    (stack d b)
    (pickup c)
    (stack c d)
```

uns_domain_string:
```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates
  (on ?x - block ?y - block)
       (ontable ?x - block)
       (clear ?x - block)
       (handempty)
       (holding ?x - block)
       )

  (:action pickup
     :parameters (?x - block)
     :precondition
     (and (clear ?x)  )
     :effect
     (and (not (ontable ?x))
       (not (handempty))
       (holding ?x)))

  (:action putdown
     :parameters (?x - block)
     :precondition
     (and (holding ?x) )
     :effect
```

```
      (and (not (holding ?x))
        (clear ?x)
        (handempty)
        ))
   (:action stack
      :parameters
      (?x - block ?y - block)
      :precondition
      (and  (clear ?y))
      :effect
      (and (not (holding ?x))
        (not (clear ?y))

        (handempty)
        (on ?x ?y)))
   (:action unstack
      :parameters
      (?x - block ?y - block)
      :precondition
      (and (handempty))
      :effect
      (and (holding ?x)
        (clear ?y)
        (not (clear ?x))
        (not (handempty))
        ))
)

examples_for_the_prompt:
['unstack-has-delete-effect-on ?x ?y',
'stack-has-add-effect-clear ?x',
'unstack-has-precondition-on ?x ?y',
'pickup-has-precondition-handempty']

budget:
7

output:
['unstack-has-delete-effect-on ?x ?y',
'stack-has-add-effect-clear ?x',
'unstack-has-precondition-on ?x ?y',
'pickup-has-precondition-handempty',
'pickup-has-precondition-ontable ?x',
'putdown-has-add-effect-ontable ?x',
'unstack-has-add-effect-not-on ?x ?y']
```

### 4.4 LLM + Search Setting for Explanations

```
"Consider a situation, where your
understanding of a task may be
best represented by the following
pddl files:
- Problem: {problem_string}
- Domain: {uns_domain_string}
The system chooses the following
plan as the optimal solutions for
the task:{initial_opt_plan}
Given the following set of potential
explanations {option_list}
Select the option you would think
would be the most reasonable
explanation. Only provide the number
of the option selected and no other
information (exclude even the
term option)."

problem_string:
(define (problem BLOCKS-8-1)
(:domain BLOCKS)
(:objects B A G C F D H E - block)
(:init (CLEAR E) (CLEAR H) (CLEAR D)
(CLEAR F) (ONTABLE C) (ONTABLE G)
 (ONTABLE D) (ONTABLE F) (ON E C)
 (ON H A) (ON A B) (ON B G)
 (HANDEMPTY))
(:goal (and (ON C D) (ON D B)
(ON B G) (ON G F) (ON F H)
(ON H A) (ON A E)))
)

uns_domain_string:
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates
    (on ?x - block ?y - block)
    (ontable ?x - block)
    (clear ?x - block)
    (handempty)
    (holding ?x - block)
   )

  (:action pickup
     :parameters (?x - block)
     :precondition
     (and (clear ?x)  )
     :effect
     (and (not (ontable ?x))
       (not (handempty))
       (holding ?x)))

  (:action putdown
     :parameters (?x - block)
     :precondition
     (and (holding ?x) )
     :effect
     (and (not (holding ?x))
       (clear ?x)
       (handempty)
       ))
  (:action stack
     :parameters
     (?x - block ?y - block)
     :precondition
     (and  (clear ?y))
     :effect
     (and (not (holding ?x))
       (not (clear ?y))
       (handempty)
       (on ?x ?y)))
  (:action unstack
     :parameters
     (?x - block ?y - block)
     :precondition
```

```
      (and (handempty))
      :effect
      (and (holding ?x)
        (clear ?y)
        (not (clear ?x))
        (not (handempty))
        ))
)
```

original_domain:

```
(define (domain BLOCKS)
  (:requirements :strips
  :typing)
  (:types block)
  (:predicates
    (on ?x - block ?y - block)
    (ontable ?x - block)
    (clear ?x - block)
    (handempty)
    (holding ?x - block)
   )

  (:action pickup
     :parameters (?x - block)
     :precondition (and (clear ?x)
     (ontable ?x) (handempty))
     :effect
     (and (not (ontable ?x))
       (not (clear ?x))
       (not (handempty))
       (holding ?x)))

  (:action putdown
     :parameters (?x - block)
     :precondition
     (and (holding ?x))
     :effect
     (and (not (holding ?x))
       (clear ?x)
       (handempty)
       (ontable ?x)))
  (:action stack
     :parameters
     (?x - block ?y - block)
     :precondition
     (and (holding ?x)
     (clear ?y))
     :effect
     (and (not (holding ?x))
       (not (clear ?y))
       (clear ?x)
       (handempty)
       (on ?x ?y)))
  (:action unstack
     :parameters (?x - block ?y - block)
     :precondition
     (and (on ?x ?y)
     (clear ?x) (handempty))
     :effect
     (and (holding ?x)
       (clear ?y)
       (not (clear ?x))
       (not (handempty))
       (not (on ?x ?y))))
)
```

initial_opt_plan:

```
    (unstack e c)
    (putdown e)
    (unstack h a)
    (stack h c)
    (unstack a b)
    (stack a e)
    (unstack h c)
    (stack h a)
    (pickup f)
    (stack f h)
    (unstack b g)
    (stack b c)
    (pickup g)
    (stack g f)
    (unstack b c)
    (stack b g)
    (pickup d)
    (stack d b)
    (pickup c)
    (stack c d)
```

option_list:

["Option 1: {'Action unstack has precondition (on ?x ?y)', 'Action pickup has precondition (handempty)', 'Action pickup has precondition (ontable ?x)', 'Action unstack has precondition (clear ?x)', 'Action stack has precondition (holding ?x)', 'Action stack has add effect (clear ?x)', 'Action unstack has delete effect (on ?x ?y)'}", "Option 2: {'Action unstack has precondition (on ?x ?y)', 'Action pickup has precondition (handempty)', 'Action unstack has precondition (clear ?x)', 'Action stack has precondition (holding ?x)', 'Action stack has add effect (clear ?x)', 'Action putdown has add effect (ontable ?x)', 'Action unstack has delete effect (on ?x ?y)'}", "Option 3: {'Action unstack has precondition (on ?x ?y)', 'Action pickup has precondition (handempty)', 'Action pickup has delete effect (clear ?x)', 'Action unstack has precondition (clear ?x)', 'Action stack has precondition (holding ?x)', 'Action stack has add effect (clear ?x)','Action unstack has delete effect (on ?x ?y)'}"]

output:

2