Learning and Leveraging Verifiers to Improve Planning Capabilities of Pre-trained Language Models

Anonymous submission

Abstract

There have been wide spread claims in the literature about the emergent reasoning capabilities of Pretrained Large Language Models. However, recent studies, have found that their ability to plan remains questionable. Through our experiments using GPT-2, we empirically demonstrate that the performance of a finetuned baseline remains poor because it violates pre-conditions of actions in the plans that it generates. To improve the planning capabilities of a finetuned LLM, we train a verifier, which can classify actions as being valid or invalid in a particular state. By randomly sampling actions from the same dataset, we generate examples of invalid actions which are then used to train a verifier which can check for action applicability. In the presence of diverse sampling from a generator and a verifier which can prune invalid trajectories, we show significant gains in the success rate on the Blocksworld domain. Additionally, we show that finetuning the GPT-2 generator itself to create the verifier generalizes better than finetuning the base GPT-2. Lastly, we investigate the role of the sampling temperature which can be used to control the exploration-exploitation tradeoff.

Introduction

Pre-trained Large Language Models which employ Transformers(Vaswani et al. 2017) as the backbone architecture have made huge strides in the field of Deep Learning and become the de-facto architecture for a variety of tasks in NLP. These models posses a rich amount of knowledge about the world, as a by-product of their pre-training. Recent works have hinted at their emergent abilities to reason (Kojima et al. 2023). Sometimes, the reasoning can be elicited by prompting the model in the correct fashion, referred to as Chain-of-Thought Prompting(Wei et al. 2023).

This leads to a natural question: can these models plan? A recent work (Valmeekam et al. 2023) investigates the planning abilities of various LLMs like GPT-3, InstructGPT and BLOOM. In this work, the authors prompt the model with a textual description of the domain and an example plan, in hope that the LLM could generalize to other instances. However, the results show very poor performance on test instances. Even after finetuning the models on plans generated by a SOTA planner, the performance remains sub-par.

The PDDL framework provides a convenient abstraction for planning by specifying relations over objects of different types. Actions are parameterized by objects and have pre-conditions and effects. The ability of an LLM like architecture to plan subsumes the ability to build a world-model of a domain. In the context of PDDL, knowing a world-model would be equivalent to knowing the effects and pre-conditions of an action. Our investigation of the poor performance of finetuned models reveals that they don't perform well because eve the finetuned models often take actions which are invalid in a particular state, leading to failed plans. This leads to poor success rate of such planners on test instances. In other words, when trained on trajectories, LLMs are bad at capturing pre-conditions of actions.

In order to alleviate this problem of finetuned models, we train a verifier which classifies a generated action as valid or invalid in a given state. Verifiers have been proposed in the context of reasoning and problem-solving for Language Models. For instance, (Cobbe et al. 2021) train a verifier for mathematical problem solving in natural language by explicitly labeling solutions generated by a finetuned generative language model. The "good" examples are present in the finetuning dataset itself. However, the "bad" examples requires human annotation which can be cumbersome and expensive. Along similar lines, Yang et al propose verifier-guided search(Yang, Deng, and Chen 2022), where each proof step is a (premise, conclusion) pair. For training a verifier, they propose augmentations, such as negating the conclusion, removing parts of the premise, etc.

In the context of PDDL planning, we propose a cheap source of negative examples. For a state in the dataset, we generate a negative example by sampling any random action present in the dataset. Equipped with this simple strategy, we can train a verifier. We demonstrate that sampling a diverse set of trajectories and using the verifier to prune them is critical for improving the success rate of these planners.

For our experiments, we use Blocksworld as the testbed and finetuned GPT-2(Radford et al. 2019) models as the generator and the verifier. Results show that given the same number of attempts a generator equipped with a verifier achieves a gain of 27% over the pure-generator baseline. Additionally, the rate of bad trajectories decreases from 52% to a mere 5%. We also investigate the role of the sampling temperature of the generator and demonstrate it's role in balancing the exploration-exploitation tradeoff. Finally, we show that finetuning the generator itself for verification generalizes better as compared to finetuning vanilla GPT-2.

Verifier Augmented Plan Generation in LLMs

The PDDL framework (Ghallab et al. 1998) defines a class of planning problems where each problem can be defined using a domain $\mathcal{D}$, a start state $\mathcal{I}$ and a goal specification $\mathcal{G}$. The domain consists of a set $\mathcal{F}$ of predicates and a set $\mathcal{A}$ of action templates. An instantiation of the domain consists of grounding the predicates over a set of objects $\mathcal{O}$. A grounded action consists of an action template a and a mapping $\mathcal{V}$ of each argument of an action to an object. Such a mapping induces a set of pre-conditions $pre(a[\mathcal{V}])$ and a set of effects $eff(a[\mathcal{V}])$. A grounded action is said to be applicable in a state if all its preconditions are true. When a grounded action is applied to a state, all the effects of the action become true in the next state.

We aim to take a **generate-test** loop approach to planning using LLMs, where the **generate** part is a finetuned LLM and the **test** part is also a finetuned LLM. We assume an offline dataset of successful trajectories $\{(s_1, a_1, s_2, a_2..., s_g)\}$ where each s_i contains a textual representation of the state. The generator and verifier are both learnt using the same dataset. Since the domain is Markovian and we work with goal-conditioned policies, policy learning can be formulated as an auto-regressive generation task wherein the model has to generate an action and the corresponding next state given a state and the goal. For this purpose, we finetune a pre-trained LLM GPT-2 (Radford et al. 2019) to be the generator. The generator is an LLM which is finetuned to generate the action and the next state a_i, s_{i+1} given the goal state and the previous state s_g, s_i. The verifier is also a LLM which is finetuned to predict whether the action a_i is applicable in state s_i.

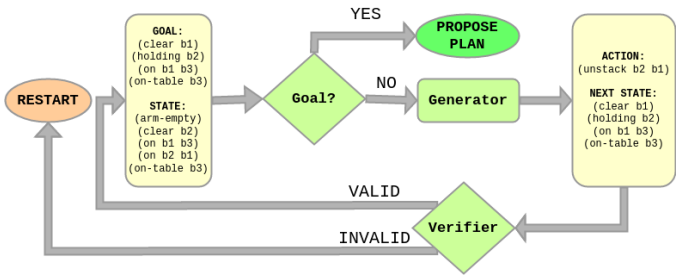

Figure 1: The figure shows a prompt which contains s_i and s_g, which is given to the generator as input and the output it generates, which contains a_i and s_{i+1}. The verifier taken the current state s_i and generated action a_i as input. After each transition generation, if the verifier approves the action, the generated next state s_{i+1}, is given as input for the next transition along with the goal s_g. If the verifier disapproves of the action, planner restarts from the initial state s_0.

Training the Generator

To create the generator, we finetune the pretrained GPT-2 model on transitions extracted from successful execution traces. From a trajectory $(s_1, a_1, s_2....s_g)$, where each s_i and a_i is the textual representation of the state and action respectively and s_g is the goal, we train GPT-2 on a transition, containing (s_g, s_i, a_i, s_{i+1}). During inference, we provide with (s_g, s_i) as the prompt and the model generates (a_i, s_{i+1}). [1] An example of the prompt can be seen in Figure 1.

Training the Verifier

In the context of planning, a verifier needs to learn which actions are applicable in a particular state. We hypothesize that preconditions can be implicitly learnt by a high-capacity model like GPT-2, given negative samples. Learning a verifier from successful trajectories is an interesting problem, since we do not have explicit knowledge about actions which are inapplicable in a particular state. We generate the training dataset for the verifier in the following manner: for every transition (s_i, a_i, s_{i+1}), (s_i, a_i) is a positive sample. To generate a negative sample, we use (s_i, a') where a' is a random action samples from the dataset of trajectories. We note that a' may well be executable in the state s_i, however, the probability of such an event is low.

To train the verifier, we finetune the generator on the verification dataset. Since the GPT-2 architecture is a auto-regressive decoder architecture, the last token's contextual embedding contains the summary of the entire sentence, which is then passed through an MLP to perform a prediction task. In our case, the verifier is passed a state and an action, and the verifier has to do a classify the action as valid or invalid in the state.

Verifier-augmented Generation

Figure 1 shows the overall verifier augmented plan generation process. The GPT-2 Generator takes the initial state $s_0 \leftarrow s_I$ and goal s_g. At the ith transition the generator incrementally generates the next action a_i and state s_{i+1} from s_i, s_g. The verifier then checks whether the transition $s_i \xrightarrow{a_i} s_{i+1}$ is valid. If it is valid, then the generator continues with s_{i+1}, s_g. If the transition is not valid, the generator is reset to start from s_0, s_g.[2] It should be noted that the learned verifier is not guaranteed to be correct in checking transition validity–it can have both false positives and false negatives.

Experimental Evaluation

Generating the training data

Using a randomized Blocksworld instance generator, with the same parameters, we generate 10,000 initial states with varying number of blocks and 200 test instances. To generate training trajectories, we use a **random-exploring**[3] agent which selects any action uniformly randomly from

[1] There are two reasons why it isn't preferred to include the entire history into the prompt. Firstly, the context window is of limited size. Secondly, if generation is done step by step, the inference time scales as $\mathcal{O}(n)$, whereas if the entire history is included in the prompt, it scales as $\mathcal{O}(n^2)$, n being the # of transitions.

[2] An alternative to resetting the generator all the way back to s_0, s_g would be to just reset it to start from the previous state s_i. One difference is that s_i, s_g may be an unsolvable planning problem even when s_0, s_g is solvable.

[3] Any planner can be used to generate such trajectories and is not important to demonstrate the utility of verifiers

the set of applicable actions in a state. In order to generate some diversity, we prune actions in which the next state has already been explored. This gives us access to an offline dataset of valid and diverse trajectories $D = \{T_i = (s_1, a_1, s_2, a_2 ... s_g)\}_{i=1}^{|D|}$. To generate training data for GPT-2 from these trajectories, we choose the end state as the goal and each individual transition is taken as an input sentence[4]. Additional details can be found in the Appendix.

Baselines and Metrics

For all the methods, we consider a maximum plan length L_{max}. To measure the utility of a verifier we consider the following methods:

- **generator@k**: This agent generates at most k plans of length at most L_{max}. In the first plan that the agent's generated next state matches the goal, the plan is proposed. The inference algorithm is given in Algorithm 1
- **generator+verifier@k**: This agent generates at most k plans of length atmost L_{max}. As soon as a plan is found which reaches the goal and the verifier approves all transitions, it is proposed. The inference algorithm is given in Algorithm 2

For all baselines, we use top-p sampling for the next token during generation (Holtzman et al. 2020) with $p = 0.99$, with a temperature $\tau = 1$. L_{max} is set to 40 steps.

Each planner proposes a plan for an instance. To assess the planners, we consider the following metrics:

- **bad-transition-rate(BTR)**: Does the plan have an illegal action in the proposed plan?
- **goal-reaching-rate(GRR)**: Is the goal achieved in the proposed plan?

Algorithm 1: Inference algorithm for generator@k

Require: G: generator, V : verifier, L_{max}: Max plan length, k: Max number of plans, s_0: Start state, s_g: Goal.
$i \leftarrow 0$
while $i < L_{max}$ **do**
  $j \leftarrow 0$
  while $j < k$ **do**
    if $s_j == g$ **then**
      return $(a_1, a_2 ... a_{j-1})$
    end if
    $a_j, s_{j+1} \leftarrow G(s_j, s_g)$
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while

Results

How does generator+verifier do? For the purpose of this experiment, we set $k = 25$ and the trained model is used for generating plans on the 200 test instances. For each test instance, there can be three possible outcomes:

1. The proposed plan is valid and reaches the goal.[5]
2. The proposed plan fails because of illegal actions in any intermediate step.
3. The planner fails to propose a plan.

Method	GRR	BTR
generator@25	0.375	0.525
generator+verifier@25	0.655	0.05

Table 1: Results of generator@25 and generator+verifier@25 on 200 Blocksworld test instances.

As we see from Table 1, generator@25 can only solve 37.5% of the instances, and has a bad transition in 52.5% of the total instances(Note that if even one action is inapplicable, the entire trajectory is invalid). However, generator+verifier@25 is able to gain substantially over generator@25 and achieve a score of 65.5%. Additionally, we see that BTR goes down significantly and is present in only 5% of the proposed plans in the test instances.

Does the performance improve with more attempts? Since the generator is non-deterministic, we expect that the GRR of the planner would increase with increasing number of attempts that it gets at generating a plan. In order to verify this hypothesis, we vary k in the range $\{1, 8, 16, 25\}$, and measure the performance of generator@k and generator+verifier@k. The results can be seen in Figure 2. It is evident that the performance of both models increases with increasing number of trajectories, but only generator+verifier@k is able to improve consistently unlike generator@k which plateaus around 16 trajectories.

[4]We only consider complete specification of the goal state in our work for simplicity, although a partial specification is an easy extension.

[5]An example plan can be found in the Appendix

Algorithm 2: Inference algorithm for generator+verifier@k

Require: G: generator, V : verifier, L_{max}: Max plan length, k: Max number of plans, s_0: Start state, s_g: Goal.
$i \leftarrow 0$
while $i < L_{max}$ **do**
  $j \leftarrow 0$
  while $j < k$ **do**
    if $s_j == g$ **then**
      return $(a_1, a_2 ... a_{j-1})$
    end if
    $a_j, s_{j+1} \leftarrow G(s_j, s_g)$
    if $V(s_j, a_j)$ fails **then**
      break
    end if
    $j \leftarrow j + 1$
  end while
  $i \leftarrow i + 1$
end while

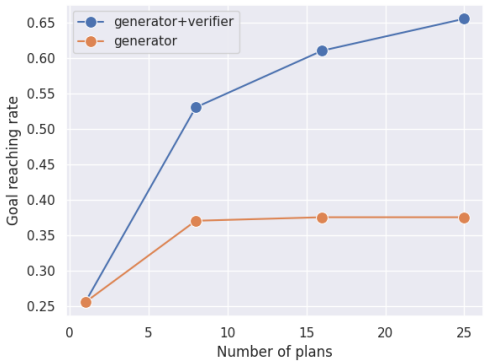

Figure 2: GRR of generator+verifier@k and generator@k with increasing number of attempts at generating a plan

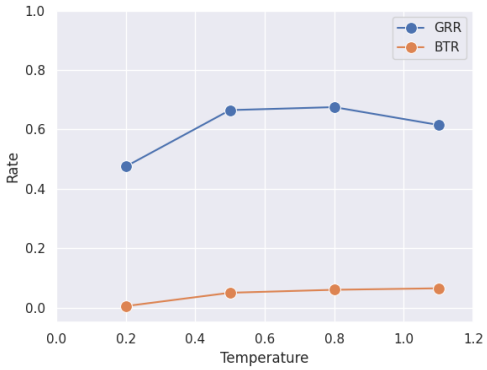

Figure 3: Plot comparing GRR of generator+verifier@25 with respect to varying values of temperature

Finetuning the generator to train the verifier leads to better generalization. Which model should be finetuned to create the verifier? The first option is to finetune GPT-2 itself, referred to as V_{base}. The other option is to finetune the generator, referred to as the $V_{generator}$[6]. From results in Table 2, we observe that finetuning the generator leads to better results. Intuitively, this could be because the generator has already been trained on a related task, which makes it more robust.

Method	GRR	BTR
generator+verifier(V_{base})@25	0.635	0.105
generator+verifier($V_{generator}$)@25	0.655	0.5

Table 2: Results of using a verifier finetuned on GPT-2 versus the generator.

How does the sampling temperature affect results? The sampling temperature τ of the LM controls the diversity in the output by scaling the un-normalized log probabilities of each token by a factor of τ. A temperature of 0 would lead to greedy sampling and a temperature of infinity would lead to a uniform distribution over next tokens. To investigate the role of temperature in the performance of the architecture, we do a sweep over $\tau \in \{0.2, 0.5, 0.8, 1.1\}$. The results can be seen in Figure 3.

The inverted U-shaped curve can be explained by the exploration-exploitation tradeoff where there is an optima reached at a particular temperature. At a lower temperature($\tau = 0.2$), the generator is very conservative and doesn't have a lot of diversity in the generated plans. Suprisingly, it also makes a much lower number of bad transitions. On the other hand, at a higher temperature($\tau = 1.1$), the generator is very exploratory and starts making more bad transitions, which is why the goal reaching rate starts decreasing. This is also suggestive of the fact that τ can be tuned on a validation set to find the optimal value.

[6]Note that the generator itself was finetuned from GPT-2.

Related Work

Unlike methods such as SayCan (Ahn et al. 2022), which use LLMs for heuristic guidance, and depend instead on external simulators for correctness, our verifier is learned from the same trajectory data used for finetuning the LLM in the first place. The idea of learning to act from an offline dataset of trajectories ties in to the field of offline RL. The ability to plan from the "logs" of a system provides efficiency by reducing the number of interactions required with the environment to learn the dynamics/policy in the environment. Even for PDDL planning, the efforts to leverage previous trajectories to provide planning capabilities pre-dates the rise of transformers. For example, (Zhuo et al. 2020) use word vector embeddings and RNNs to support plan recognition. Recent works such as Decision Transformer(Chen et al. 2021) also employ large Transformer models to learn policies purely from trajectories. The work by (Pallagani et al. 2022) attempts to learn plans directly from action sequences generated by a planner. They too represent the states and actions in the form of text which acts as input for an LLM. Neither of these works however leverage independently trained verifiers like we do.

Conclusions and Future Work

From our study, we discover that the usage of a verifier is critical for using Large Language Models for the purposes of planning, since it can prune plans which have illegal actions. It is able to successfully unlock scaling with respect to an increasing number of attempts at generating a plan. An interesting question might be how to employ the feedback of verifiers into the generator during test-time. Also, how does the technique of verification scale to the level of pre-trained models as large as GPT-3 or GPT-4. Is it possible that a verifier could be built merely by prompt engineering via in-context learning? All these questions seem worth investigating, which we leave for future work. The appendix contains additional details on the training and test data, the hyper-parameter settings for the training of the generator and verifier, as well as an example of a 40 transition plan generated by the generator+verifier@25.

References

Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; Herzog, A.; Ho, D.; Hsu, J.; Ibarz, J.; Ichter, B.; Irpan, A.; Jang, E.; Ruano, R. J.; Jeffrey, K.; Jesmonth, S.; Joshi, N. J.; Julian, R.; Kalashnikov, D.; Kuang, Y.; Lee, K.-H.; Levine, S.; Lu, Y.; Luu, L.; Parada, C.; Pastor, P.; Quiambao, J.; Rao, K.; Rettinghouse, J.; Reyes, D.; Sermanet, P.; Sievers, N.; Tan, C.; Toshev, A.; Vanhoucke, V.; Xia, F.; Xiao, T.; Xu, P.; Xu, S.; Yan, M.; and Zeng, A. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. arXiv:2204.01691.

Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; and Mordatch, I. 2021. Decision Transformer: Reinforcement Learning via Sequence Modeling. arXiv:2106.01345.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; Hesse, C.; and Schulman, J. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168.

Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. PDDL - The Planning Domain Definition Language.

Holtzman, A.; Buys, J.; Du, L.; Forbes, M.; and Choi, Y. 2020. The Curious Case of Neural Text Degeneration. arXiv:1904.09751.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2023. Large Language Models are Zero-Shot Reasoners. arXiv:2205.11916.

Pallagani, V.; Muppasani, B.; Murugesan, K.; Rossi, F.; Horesh, L.; Srivastava, B.; Fabiano, F.; and Loreggia, A. 2022. Plansformer: Generating Symbolic Plans using Transformers. arXiv:2212.08681.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language Models are Unsupervised Multitask Learners.

Seipp, J.; Torralba, Á.; and Hoffmann, J. 2022. PDDL Generators. https://doi.org/10.5281/zenodo.6382173.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023. Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change). arXiv:2206.10498.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. arXiv:1706.03762.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; and Zhou, D. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

Yang, K.; Deng, J.; and Chen, D. 2022. Generating Natural Language Proofs with Verifier-Guided Search. arXiv:2205.12443.

Zhuo, H. H.; Zha, Y.; Kambhampati, S.; and Tian, X. 2020. Discovering Underlying Plans Based on Shallow Models. *ACM Trans. Intell. Syst. Technol.*, 11(2): 18:1–18:30.

Appendix

Training data

The training set consists of 10000 initial states with 3 to 8 blocks. The generators used were taken from (Seipp, Torralba, and Hoffmann 2022). Using these, we generate trajectories of length at most 20, by sampling a random action from the valid set of actions in a state. Whenever a state is visited which has already been visited, we stop the trajectory. Then, we collect all the transitions in that trajectory with the goal being the end state. This allowed us to generate almost 61,000 trajectories. We create a similar validation dataset of 6000 trajectories. To convert the PDDL state and action to its textual representation, we convert all the true propositions to text and append them in lexicographical order.

Test instances

With a process similar to the training data generation, we generate 200 trajectories on different initial states of length atmost 30. The end state is sampled from the second half of the generated trajectory, so that the end state requires a non-trivial number of actions to reach.

Generator

The generator is a finetuned GPT-2 model trained on 2 Nvidia V100 GPUs. The hyperparameters for training the generator are given in Table 3

Parameter	Value
Batch size	16
Training epochs	20
Learning rate	5e-6
Warmup steps	50

Table 3: Training hyperparameters for the generator

Verifier

The verifier is finetuned on the generator which itself is a GPT-2. It was trained on 1 Nvidia V100 GPU. The hyperparameters for training the verifier are given in Table 4

Parameter	Value
Batch size	8
Training epochs	1
Learning rate	5e-6
Warmup steps	20

Table 4: Training hyperparameters for the verifier

An Example Plan

The following is an example plan generated by generator+verifier@25. Note that the plan will not be (near-)optimal, because the training data doesn't use an (near-)optimal planner for trajectories. If (near-)optimal trajectories are used for training, the quality of plans will also be improved.

Transition 1:

```
GOAL: (arm-empty) (clear b1) (clear
b2) (on b1 b3) (on b3 b4) (on-table b2)
(on-table b4)

STATE: (arm-empty) (clear b1) (on b1
b4) (on b3 b2) (on b4 b3) (on-table b2)

ACTION: (unstack b1 b4)

NEXT STATE: (clear b4) (holding b1) (on
b3 b2) (on b4 b3) (on-table b2)
```

Transition 2:

```
GOAL: (arm-empty) (clear b1) (clear
b2) (on b1 b3) (on b3 b4) (on-table b2)
(on-table b4)

STATE: (clear b4) (holding b1) (on b3
b2) (on b4 b3) (on-table b2)

ACTION: (putdown b1)

NEXT STATE: (arm-empty) (clear b1)
(clear b4) (on b3 b2) (on b4 b3)
(on-table b1) (on-table b2)
```

Transition 3:

```
GOAL: (arm-empty) (clear b1) (clear
b2) (on b1 b3) (on b3 b4) (on-table b2)
(on-table b4)

STATE: (arm-empty) (clear b1) (clear
b4) (on b3 b2) (on b4 b3) (on-table b1)
(on-table b2)

ACTION: (pickup b1)

NEXT STATE: (clear b4) (holding b1) (on
b3 b2) (on b4 b3) (on-table b2)
```

Transition 4:

```
GOAL: (arm-empty) (clear b1) (clear
b2) (on b1 b3) (on b3 b4) (on-table b2)
(on-table b4)

STATE: (clear b4) (holding b1) (on b3
b2) (on b4 b3) (on-table b2)

ACTION: (putdown b1)

NEXT STATE: (arm-empty) (clear b1)
(clear b4) (on b3 b2) (on b4 b3)
(on-table b1) (on-table b2)
```

Transition 5:

```
GOAL: (arm-empty) (clear b1) (clear
b2) (on b1 b3) (on b3 b4) (on-table b2)
```

(on-table b4)

STATE: (arm-empty) (clear b1) (clear b4) (on b3 b2) (on b4 b3) (on-table b1) (on-table b2)

ACTION: (unstack b4 b3)

NEXT STATE: (clear b1) (clear b3) (holding b4) (on b3 b2) (on-table b1) (on-table b2)

Transition 6:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (holding b4) (on b3 b2) (on-table b1) (on-table b2)

ACTION: (stack b4 b1)

NEXT STATE: (arm-empty) (clear b3) (clear b4) (on b3 b2) (on b4 b1) (on-table b1) (on-table b2)

Transition 7:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b3) (clear b4) (on b3 b2) (on b4 b1) (on-table b1) (on-table b2)

ACTION: (unstack b3 b2)

NEXT STATE: (clear b2) (clear b4) (holding b3) (on b4 b1) (on-table b1) (on-table b2)

Transition 8:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b2) (clear b4) (holding b3) (on b4 b1) (on-table b1) (on-table b2)

ACTION: (putdown b3)

NEXT STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b4 b1) (on-table b1) (on-table b2) (on-table b3)

Transition 9:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b4 b1) (on-table b1) (on-table b2) (on-table b3)

ACTION: (unstack b4 b1)

NEXT STATE: (clear b1) (clear b2) (clear b3) (holding b4) (on-table b1) (on-table b2) (on-table b3)

Transition 10:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b2) (clear b3) (holding b4) (on-table b1) (on-table b2) (on-table b3)

ACTION: (putdown b4)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (clear b4) (on-table b1) (on-table b2) (on-table b3) (on-table b4)

Transition 11:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (clear b4) (on-table b1) (on-table b2) (on-table b3) (on-table b4)

ACTION: (pickup b2)

NEXT STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

Transition 12:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

ACTION: (stack b2 b1)

NEXT STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b2 b1)

(on-table b1) (on-table b3) (on-table b4)

Transition 13:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b2 b1) (on-table b1) (on-table b3) (on-table b4)

ACTION: (unstack b2 b1)

NEXT STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

Transition 14:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

ACTION: (stack b2 b3)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b4) (on b2 b3) (on-table b1) (on-table b3) (on-table b4)

Transition 15:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b4) (on b2 b3) (on-table b1) (on-table b3) (on-table b4)

ACTION: (unstack b2 b3)

NEXT STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

Transition 16:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

ACTION: (stack b2 b4)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

Transition 17:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

ACTION: (pickup b3)

NEXT STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

Transition 18:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

ACTION: (stack b3 b2)

NEXT STATE: (arm-empty) (clear b1) (clear b3) (on b2 b4) (on b3 b2) (on-table b1) (on-table b4)

Transition 19:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b3) (on b2 b4) (on b3 b2) (on-table b1) (on-table b4)

ACTION: (unstack b3 b2)

NEXT STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

Transition 20:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

ACTION: (putdown b3)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

Transition 21:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

ACTION: (pickup b3)

NEXT STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

Transition 22:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

ACTION: (putdown b3)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

Transition 23:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

ACTION: (pickup b1)

NEXT STATE: (clear b2) (clear b3) (holding b1) (on b2 b4) (on-table b3) (on-table b4)

Transition 24:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b2) (clear b3) (holding b1) (on b2 b4) (on-table b3) (on-table b4)

ACTION: (putdown b1)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

Transition 25:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

ACTION: (pickup b3)

NEXT STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

Transition 26:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b2) (holding b3) (on b2 b4) (on-table b1) (on-table b4)

ACTION: (putdown b3)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

Transition 27:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b2 b4) (on-table b1) (on-table b3) (on-table b4)

ACTION: (unstack b2 b4)

NEXT STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

Transition 28:
GOAL: (arm-empty) (clear b1) (clear

b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

ACTION: (stack b2 b1)

NEXT STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b2 b1) (on-table b1) (on-table b3) (on-table b4)

Transition 29:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b2 b1) (on-table b1) (on-table b3) (on-table b4)

ACTION: (unstack b2 b1)

NEXT STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

Transition 30:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

ACTION: (stack b2 b3)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b4) (on b2 b3) (on-table b1) (on-table b3) (on-table b4)

Transition 31:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b4) (on b2 b3) (on-table b1) (on-table b3) (on-table b4)

ACTION: (unstack b2 b3)

NEXT STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

Transition 32:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (clear b4) (holding b2) (on-table b1) (on-table b3) (on-table b4)

ACTION: (stack b2 b1)

NEXT STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b2 b1) (on-table b1) (on-table b3) (on-table b4)

Transition 33:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b2) (clear b3) (clear b4) (on b2 b1) (on-table b1) (on-table b3) (on-table b4)

ACTION: (pickup b3)

NEXT STATE: (clear b2) (clear b4) (holding b3) (on b2 b1) (on-table b1) (on-table b4)

Transition 34:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b2) (clear b4) (holding b3) (on b2 b1) (on-table b1) (on-table b4)

ACTION: (stack b3 b4)

NEXT STATE: (arm-empty) (clear b2) (clear b3) (on b2 b1) (on b3 b4) (on-table b1) (on-table b4)

Transition 35:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b2) (clear b3) (on b2 b1) (on b3 b4) (on-table b1) (on-table b4)

ACTION: (unstack b2 b1)

NEXT STATE: (clear b1) (clear b3)

(holding b2) (on b3 b4) (on-table b1) (on-table b4)

Transition 36:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (holding b2) (on b3 b4) (on-table b1) (on-table b4)

ACTION: (stack b2 b3)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (on b2 b3) (on b3 b4) (on-table b1) (on-table b4)

Transition 37:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (on b2 b3) (on b3 b4) (on-table b1) (on-table b4)

ACTION: (unstack b2 b3)

NEXT STATE: (clear b1) (clear b3) (holding b2) (on b3 b4) (on-table b1) (on-table b4)

Transition 38:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b1) (clear b3) (holding b2) (on b3 b4) (on-table b1) (on-table b4)

ACTION: (putdown b2)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b3 b4) (on-table b1) (on-table b2) (on-table b4)

Transition 39:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (arm-empty) (clear b1) (clear b2) (clear b3) (on b3 b4) (on-table b1) (on-table b2) (on-table b4)

ACTION: (pickup b1)

NEXT STATE: (clear b2) (clear b3) (holding b1) (on b3 b4) (on-table b2) (on-table b4)

Transition 40:
GOAL: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)

STATE: (clear b2) (clear b3) (holding b1) (on b3 b4) (on-table b2) (on-table b4)

ACTION: (stack b1 b3)

NEXT STATE: (arm-empty) (clear b1) (clear b2) (on b1 b3) (on b3 b4) (on-table b2) (on-table b4)