ProRAC: A Neuro-symbolic Method for Reasoning about Actions with LLM-based Progression and Search

Anonymous ACL submission

Abstract

Reasoning about actions and change (RAC) plays an important role in AI. It involves reasoning about preconditions and effects of actions, and has applications in planning. Despite recent advances of large language models (LLMs) in natural language processing, evaluations on several RAC benchmarks demonstrate that LLMs face significant challenges in RAC. Besides, effective methods for improving their RAC capabilities, especially neuro-symbolic approaches, remain largely unexplored. In this paper, we propose **ProRAC** (**Pro**gression-based **R**easoning about **A**ctions and **C**hange), a neuro-symbolic framework that leverages LLMs to tackle RAC and planning problems through an agentic, modular design. Central to ProRAC is a unified framework that operates in two modes: a reasoning mode that sequentially executes actions to validate states and answer queries, and a searching mode that integrates this progression into an A* search guided by a fine-tuned heuristic model to find plans. Extensive evaluations across multiple RAC and planning bechmarks demonstrate that ProRAC significantly outperforms existing methods, exhibiting robust performance across different model backbones and task types. [1]

1 Introduction

Reasoning about actions and change (RAC) plays an important role in artificial intelligence. It involves reasoning about the preconditions and effects of actions. To achieve autonomous intelligence, agents must act independently in the world: perform purposeful actions, predict the effects of actions, and compose multiple actions to achieve complex goals. The last and most challenging task is planning, which is an application of RAC and an important research area in classical AI.

In classical AI, many formalisms have been proposed for RAC, including Situation Calculus (Reiter, 2001), Event Calculus (Kowalski and Sergot, 1989), and Action Languages (Gelfond and Lifschitz, 1993; Giunchiglia and Lifschitz, 1998). Two fundamental reasoning methods to address RAC problems within these formalisms are Regression and Progression (Reiter, 2001). Roughly, regression reduces a query about the future to a query about the initial state. Progression, on the other hand, changes the initial state according to the effects of each action and then checks whether the formula holds in the resulting state. One advantage of progression compared to regression is that after a state has been progressed, many queries about the resulting state can be processed without extra overhead.

While RAC problems have been extensively studied within classical symbolic approaches, LLMs have demonstrated remarkable capabilities in recent years, offering new possibilities for solving RAC problems in natural language. Recently, several benchmarks for RAC over natural language have been proposed. He et al. (2023) introduced TRAC, a benchmark targeting RAC problems that includes some basic tasks such as projection, executability, plan verification, but focuses solely on the Blocksworld domain. Another RAC benchmark is ActionReasoningBench (Handa et al., 2025), which covers multiple classical planning domains, such as Depots and Grippers. ACPBench (Kokel et al., 2025) is another benchmark in parallel with ActionReasoningBench. ACPBench also covers multiple planning domains and features both multiple-choice and true/false questions. Finally, PlanBench (Valmeekam et al., 2023) is a challenging benchmark that is specifically designed to evaluate the planning generation capabilities of LLMs.

The success of LLMs in various reasoning tasks has also led to growing interest in neuro-symbolic (NeSy) AI approaches. Notably, neuro-

[1] Code available at: https://anonymous.4open.science/r/ProRAC_public-E788

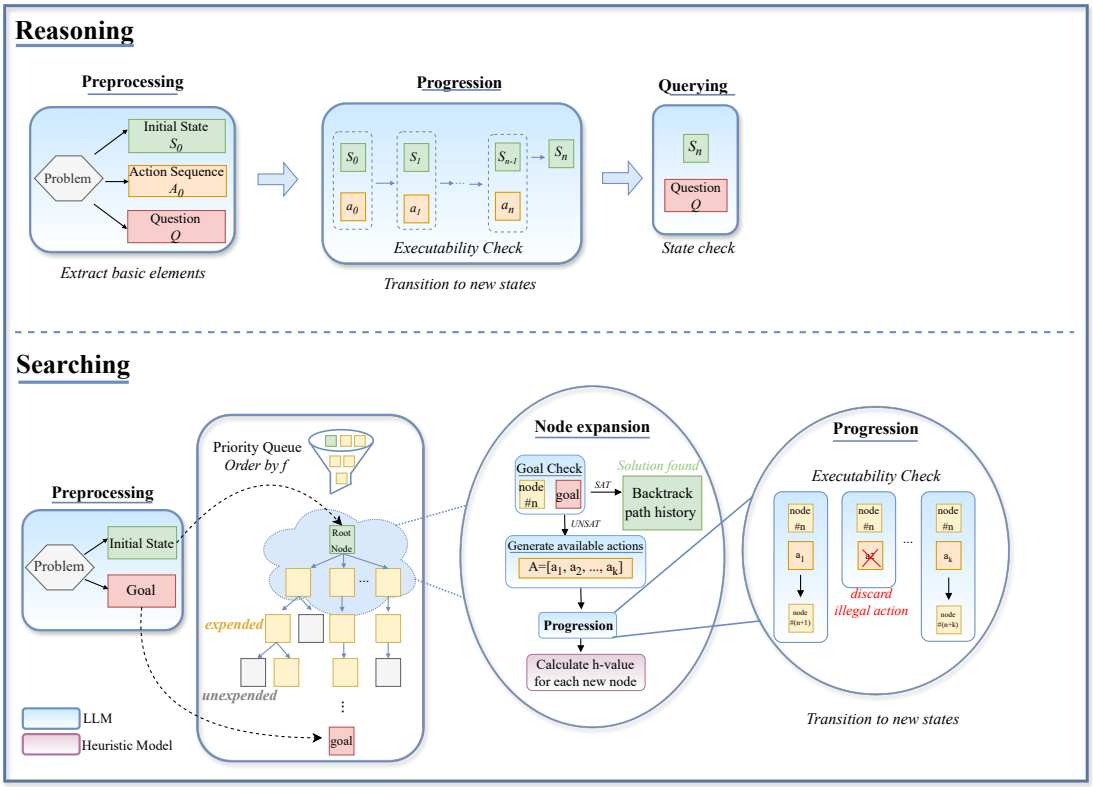

Figure 1: The framework of ProRAC. The framework has two modes: reasoning mode (for RAC tasks) and searching mode (for plan generation task).

symbolic AI goes beyond a mere combination of neural networks and symbolic solvers. In fact, neuro-symbolic approaches encompass a spectrum of integration paradigms, ranging from neural networks processing symbolic representations to purely neural systems that exhibit symbolic-like reasoning behaviors (Kautz, 2022). In particular, Symbolic[Neuro] is an architecture that integrates symbolic as a structural or functional blueprint to guide neural network design. In particular, the Symbolic[Neuro] architecture embeds neural components within a symbolic problem solver, and the Neuro[Symbolic] architecture embeds symbolic components within a neural network system.

While several benchmarks have been introduced, comprehensive neuro-symbolic frameworks for solving RAC problems remain scarce. LLM+AL (Ishay and Lee, 2025) is a neuro-symbolic framework leveraging LLMs and symbolic solvers to solve planning problem. The LLM functions as a semantic parser, translating problems expressed in natural language into action language $\mathcal{BC}+$. Then a symbolic solver performs reasoning to produce the final solution based on translation results. While effective on planning tasks, the framework requires human intervention, as in some cases the LLM consistently fails to accurately translate problems into a formal language. Besides, it evaluates only a small number of classical planning problems and does not address the aforementioned RAC benchmarks.

Beyond symbolic translation approaches, recent work has investigated search-based neuro-symbolic approaches to general reasoning tasks. Tree of Thoughts (Yao et al., 2023) formulates problem solving as a tree search over intermediate reasoning states, where each node corresponds to a coherent textual thought and branching reflects alternative progression paths. Similarly, Hao et al. (2023) treats LLMs as world models that can simulate state transitions and uses Monte Carlo Tree Search to explore reasoning trajectories. These methods demonstrate that LLMs can perform progression-based reasoning without explicit symbolic translation. However, ToT relies on direct LLM-based prompting for heuristic scoring, which can be unreliable, while RAP incurs prohibitive computational cost.

To enhance the reasoning capabilities of LLMs in RAC problems, we propose **ProRAC**, a neuro-symbolic framework targeting RAC and planning problems. For RAC tasks, ProRAC follows three steps — *Preprocessing*, *Progression*, and *Query-*

ing. First, we extract the initial state, action sequence, and query from the natural language problem. Then, we iteratively call an LLM to execute each action based on the current state, updating the state accordingly. Finally, the LLM evaluates the query against the final state to generate answers. For planning tasks, we integrate the Progression module into an A* search. An LLM generates possible actions from the current state, and then the LLM checks their executability to ensure only executable actions are considered. The Progression module computes the resulting states, and a fine-tuned small LLM provides heuristic values based on the current and goal states, guiding the search efficiently.

In order to validate the effectiveness of ProRAC, we conducted experiments on several RAC and planning benchmarks: TRAC, ACPBench, ActionReasoningBench and PlanBench. Experimental results indicate that our framework is more efficient than direct LLM prompting, autoformalization methods, and uninformed search approaches.

To the best of our knowledge, ProRAC is the first unified framework to integrate reasoning and searching to systematically solve problems across both RAC-oriented benchmarks and PlanBench.

To summarize, our main contributions are:

- **A Unified Neuro-Symbolic Framework**: We propose ProRAC, the first framework capable of solving both RAC and planning benchmarks within a single neuro-symbolic architecture, without relying on external symbolic solvers.
- **Learning-based Heuristic from Symbolic Supervision**: We propose a paradigm that trains a dedicated LM-based heuristic via automatically generated symbolic supervision, distilling exact heuristic signals from classical planners into a neural heuristic for natural language planning.
- **Extensive Empirical Validation**: We demonstrate that ProRAC significantly outperforms existing methods across four RAC and planning benchmarks, showing robust results in both reasoning and planning.

2 Related Work

In classic AI, researchers addressed RAC problems by translating them into formal representations and employing symbolic solvers for reasoning. Although symbolic solver-based approaches are capable of solving RAC problems, such methods inevitably suffer from two major issues. First, symbolic methods often require manually translating problems into formal languages, this formalization process is expensive and demands a high level of expert knowledge from translators. Second, in symbolic methods, actions are strictly defined, which limits the generalization ability of symbolic solvers and makes them difficult for handling varied natural language expressions.

As LLMs have exhibited powerful semantic parsing capabilities, many researchers have proposed combining LLMs with symbolic solvers. In this neuro-symbolic paradigm, also known as autoformalization, LLMs serve as translation modules that translate natural language described problems into formal representations, which are subsequently solved by symbolic solvers (Pan et al., 2023; Olausson et al., 2023; Callewaert et al., 2025). However, the ability of LLMs to accurately translate problems from natural language into formal languages remains limited. Many works based on autoformalization demonstrate that LLMs still face significant challenges in the formalization process, particularly on complex reasoning problems (Zhong et al., 2021).

In classical AI, planning problems are commonly formulated as search problems over a state space S, defined by an initial state S_0, a transition model T, a goal G, and a path cost function c (Russell and Norvig, 2020). A* search, a representative of informed search algorithms, extends uninformed search by incorporating a heuristic function. The efficiency of A* depends on the quality of the heuristic function, a well-designed heuristic can dramatically reduce the search space.

Designing effective heuristics has always been a challenging problem. In classical AI, domain-independent heuristics are typically constructed using the idea of relaxation. One of the most influential relaxation techniques is delete relaxation, which simplifies the planning problem by ignoring the delete effects of actions (Bonet and Geffner, 1999). Key examples include the additive heuristic h_{add} and the max heuristic $h_{\max}$.

For planning problems based on natural language, it is intuitive to employ LLMs to compute heuristic values. Some researchers use LLMs to compute heuristic values directly (Liu et al., 2024; Ling et al., 2025), iteratively refining the heuristics via prompt-based methods and evaluating them on small, manually curated test sets with ground-

Model	PR	EXE	PV
4o	94.73	96.58	88.69
4o-0CoT	96.19	97.22	87.51
4o-2CoT	97.77	95.56	93.33
4o-SC	97.78	95.56	93.33
4o-ProRAC	**100**	**97.77**	**95.56**
4o-mini	**95.56**	73.33	88.89
4o-mini-0CoT	95.56	68.88	84.44
4o-mini-2CoT	93.33	73.33	86.67
4o-mini-SC	88.89	**84.44**	**91.11**
4o-mini-ProRAC	93.33	80	84.44
v3.2	97.77	100	88.88
v3.2-0CoT	100	100	95.56
v3.2-2CoT	100	100	95.56
v3.2-SC	97.77	88.89	88.89
v3.2-ProRAC	**100**	**100**	**100**

Table 1: Performance (%) of ProRAC and baseline methods on TRAC benchmark. PR=Projection, EXE=Executability, PV=Plan Verification. **Bold** indicates the best performance.

truth heuristic values for the given problem. This process can eventually yield a heuristic function capable of solving the given problem. However, these itertative-refiniment approaches suffer from significant API costs. Besides, constructing test sets manually with ground-truth heuristic values is difficult for complex problems. Distinct from prior works, we fine-tuned a small LM to learn these heuristics from automatically generated symbolic data. This approach drastically reduces API cost while maintaining high accuracy.

3 Overview of ProRAC

Figure 1 illustrates the overview of our framework. To address RAC and plan generation problems, the framework operates in two modes: **Reasoning** (the action sequence is given) and **Searching** (heuristic search for complex planning). Crucially, the *Progression* module—originally designed for reasoning, serves as the fundamental state transition operator within the search process.

The entire reasoning/searching process involves three key components: *Preprocessing*, *Progression*, and *Search/Querying*. The entire process is driven by LLMs, eliminating the need for external symbolic solvers. To minimize our framework's reliance on extensive prompt engineering, we adopt a domain-independent strategy, employing a minimal set of two fixed demonstrations for both executability and state checks, regardless of the domain. Moreover, the instructions for each module remain consistent regardless of the specific task.

3.1 Preprocessing

The preprocessing step initializes the problem solving. For the Reasoning mode, the LLM extracts the initial state S_0, the given action sequence A, and the query q. For the Searching mode, where the action sequence is unknown, the LLM extracts the initial state S_0 and the goal state G. This step is technically necessary because some benchmarks (such as ActionReasoningBench) embed the action sequences within the questions.

3.2 Progression

The core component shared by both modes is **Progression**. In this step, the current states S_i and each action a_i in action sequence A are sequentially provided to the LLM to generate the corresponding new states S_{i+1}, that is:

$$S_{i+1} = T_\theta(S_i, a_i) \quad (i = 0, \ldots, k-1) \quad (1)$$

where $T_\theta(\cdot)$ represents the transition function implemented by the LLM, corresponding to the single-step progression from the current state to the next state.

This process is performed iteratively, with the LLM explicitly instructed to return the states of all objects, reflecting the progression principle in symbolic methods, where the initial state is continuously updated based on the effects of each action.

Besides, it is possible that certain actions in the given sequence cannot be executed because their preconditions are not satisfied in the current state. We add an additional executability check before taking each action. The executability check asks the LLM whether all preconditions of the action are satisfied in the current state. This module is also used in the Searching mode.

In the Reasoning mode, this function is applied iteratively to a fixed sequence A. In the Searching mode, it is invoked dynamically during node expansion to generate successor nodes.

3.3 Querying

After step two, we obtain the final state S_n. In the third step, we ask the LLM whether S_n satisfies

Model	Depots				Driverlog				Mystery				Grippers			
	AE	EFF	FT	ST	AE	EFF	FT	ST	AE	EFF	FT	ST	AE	EFF	FT	ST
4o	**97.74**	79.1	78.13	73.68	77.14	80.95	70	80	75	45.5	89.48	74.1	95.24	90.38	95.92	83.33
4o-0CoT	97.37	48.84	84.38	39.47	82.86	90.95	76.67	80	82.1	54.55	88.9	64	95.24	90.38	93.88	83.33
4o-2CoT	97.37	74.41	75	78.94	**94.28**	90.47	73.33	92	80	52.2	79	69	95.24	89	90	83.3
4o-SC	97.37	69.77	75	81.58	91.42	90.47	**90**	**92**	75	43.47	84.21	65.51	95.24	88.46	93.87	**90**
4o-ProRAC	92.10	**93.02**	**93.75**	**84.21**	80	**100**	80	84	**82.5**	**69.56**	**89.48**	**79.31**	**97.61**	**100**	**97.59**	86.67
4o-mini	**94.7**	58.14	67.74	81.6	77.14	90.5	80	56	65	52.2	68.42	58.62	**92.9**	84.62	89.6	66.67
4o-mini-0CoT	89.5	53.5	71	79	71.43	95.24	86.67	**84**	65	39.13	**84.21**	55.17	83.33	86.54	85.71	70
4o-mini-2CoT	92.1	76.64	65.62	60.52	85.71	85.72	**90**	64	70	39.13	79	**59**	91	79	88	70
4o-mini-SC	89.47	51.16	71.87	**86.84**	**80**	95.23	90	84	72.5	39.13	73.68	51.72	92.85	84.61	89.79	73.33
4o-mini-ProRAC	81.57	**87.5**	**84.38**	73.68	77.14	**100**	80	68	**75**	**65.22**	73.68	**65.51**	88.1	**92.30**	**89.79**	**79.67**
v3.2	84.73	72.09	78.13	71.05	91.42	85.71	84.61	73.33	77.5	39.13	89.47	58.62	95.24	94.23	93.87	80
v3.2-0CoT	92.10	72.09	84.37	68.42	85.29	85.71	82.61	78.94	77.5	47.82	**94.73**	48.27	90.47	96.15	100	80
v3.2-2CoT	97.36	76.64	81.25	73.68	82.85	90	88.46	75	80	52.17	89.47	48.27	83.33	96.15	93.87	83.33
v3.2-SC	**94.73**	84.21	86.84	89.47	**94.29**	90.47	86.67	56	77.5	47.83	94.73	55.17	95.23	96.15	97.95	83.33
v3.2-ProRAC	92.10	**90.69**	**100**	**89.47**	91.42	**100**	**90**	**80**	**90**	**82.60**	84.21	**82.75**	**97.61**	**100**	**100**	**93.33**

Table 2: Performances(%) of ActionReasoningBench on Depots, Driverlog, Mystery and Grippers. AE=Action Executability, EFF=Effects, FT=Fluent tracking, ST=State Tracking. **Bold** indicates the best performance.

the query q obtained from step one. The query may contain multiple propositions, and the LLM returns true only if S_n satisfies all propositions in q. If there is any proposition that cannot be satisfied, the LLM returns false as the answer.

3.4 Searching

To tackle complex planning tasks, we introduce a **Searching mode** based on the A* algorithm, where the heuristic is provided by a fine-tuned language model rather than hand-designed or prompt-based heuristics. This enables the agent to perform deliberative planning by integrating the LLM as a transition model and a fine-tuned model as a heuristic evaluator.

Heuristic Model Fine-tuning We fine-tuned a Qwen3-1.6B model (Yang et al., 2025) as the heuristic model with LoRA fine-tuning (Hu et al., 2022). The heuristic function $h(n)$ is trained on a symbolic corpus generated via an automated pipeline. We first create diverse problem instances using PDDL instance generators and solve them with Fast Downward (Helmert, 2006) to obtain optimal plans. These plans are then decomposed into state-action trajectories, where each intermediate state is annotated with its goal distance (heuristic value) calculated by PyperPlan (Alkhazraji et al., 2020). Then we translate the data from PDDL language to natural language. By training on these samples, the model learns to map the semantic context of a state and its corresponding goal to a heuristic value. More details about the data generation process can be found in the appendix A.2.

Search Process The agent explores the state space via a priority queue ordered by

$$f = g + h \tag{2}$$

where g denotes the length of the action sequence (path) from the initial state S_0 to the current state.

Each node expansion follows a reflect-then-act cycle: (1) **Node Selection & Goal Check**: The queue pops the node with the minimum f value and invokes the LLM to verify if its state satisfies G. If so, the search terminates. (2) **Expansion**: For unsatisfied states, the LLM proposes candidate actions, which are validated via the executability check module and processed by the progression module to generate verified successor states ($S_{next} = T_\theta(S, a)$). (3) **Evaluation**: The heuristic model assigns h-values to new nodes to update the queue.

4 Experiments

4.1 Benchmarks

We evaluated ProRAC on several RAC and planning benchmarks. We selected diverse domains from different benchmarks, ensuring that these domains do not overlap with each other. These domains cover a wide range of scenarios, including logistics and transportation, block stacking, and grid-based navigation.

- **TRAC**. Focusing on the Blocksworld domain, we evaluate three core tasks: (1) **Executability**, verifying action preconditions; (2) **Projection**, tracking state changes after action se-

Model	Satellite				Spanner				Visitall			
	AE	EFF	FT	ST	AE	EFF	FT	ST	AE	EFF	FT	ST
4o	75.68	79.12	78.72	66.77	75	84.48	85	72.22	69.23	86.05	**93.33**	80
4o-0CoT	81.08	83.33	85.11	66.67	69.44	74.07	83	75	64.1	83.72	93.33	**86.67**
4o-2CoT	75.67	83.3	87.23	75	**94.44**	88.88	**93.61**	72.22	64.1	90.69	93.33	86.67
4o-SC	78.37	83.33	**91.48**	75	80.55	81.48	91.48	75	74.36	83.72	93.33	93.33
4o-ProRAC	**91.89**	**100**	89.36	**81.08**	91.66	**88.88**	85.10	**86.11**	**94.87**	**93.02**	86.67	86.67
4o-mini	56.8	62.5	72.34	61.11	77.8	74.1	89.4	**82.86**	66.67	74.42	86.67	66.67
4o-mini-0CoT	51.35	66.67	78.23	68.6	77.14	84.62	89.4	71.43	66.67	74.42	90	73.33
4o-mini-2CoT	75.67	62.5	**78.72**	66.67	85.71	**85.71**	**90**	64	53.84	69.76	**90**	60
4o-mini-SC	70.27	70.83	74.46	69.44	88.88	81.48	80	69.44	51.28	72.09	80	73.33
4o-mini-ProRAC	**86.48**	**91.66**	76.59	**77.78**	**88.88**	81.48	72.34	75	**82.05**	**80**	86.67	**73.33**
v3.2	86.49	75	82.97	69.44	86.11	85.18	85.18	66.66	82.05	81.39	83.33	60
v3.2-0CoT	72.97	79.16	78.72	63.88	86.11	85.18	85.18	**75**	87.17	81.39	80	60
v3.2-2CoT	72.97	62.5	80.85	61.11	88.88	85.18	85.18	63.88	84.61	81.39	80	53.33
v3.2-SC	81.9	79.17	82.97	72.22	94.44	88.46	88.46	72.22	89.74	79.07	83.33	53.33
v3.2-ProRAC	**86.48**	**100**	**92.9**	**78.26**	**100**	**92.59**	**91.48**	72.22	**92.30**	**86.04**	**86.67**	**86.67**

Table 3: Performances(%) of ActionReasoningBench on Satellites, Spanner and Visitall. AE=Action Executability, EFF=Effects, FT=Fluent tracking, ST=State Tracking. **Bold** indicates the best performance.

quences; and (3) **Plan Verification**, assessing if a sequence achieves the goal.

- **ActionReasoningBench**. This benchmark spans seven domains including Depots, Driverlog, Mystery, Grippers, Satellite, Spanner, and Visitall. While its tasks: executability, effects, fluent tracking and state tracking, align with those in TRAC, it poses significantly higher complexity due to the substantially larger number of propositions per query.
- **ACPBench**. We evaluate six distinct domains: Ferry, Swap, Logistics, Rovers, Floortile, and Goldminer. While its applicability and progression tasks align with TRAC's executability and projection, its validation task introduces a more rigorous tripartite classification. It requires the agent to distinguish whether an action sequence is valid (no undefiend actions), applicable (executable in the current state), or constitutes a complete plan (reaches the goal), presenting a higher diagnostic challenge.
- **PlanBench**. We conduct experiments on the Plan Generation task within the Blocksworld domain, the primary domain of PlanBench. In this task, a language model is provided with an initial state and a goal, and is required to generate a sequence of actions (a plan) that transforms the initial state into the goal state.

4.2 Methods

For RAC tasks, We used GPT-4o (OpenAI, 2024), GPT-4o-mini, and DeepSeek-v3.2 (Liu et al., 2025) as the underlying models for evaluation. For comparison, we adopted Zero-Shot, Zero-Shot-CoT (Kojima et al., 2022), Two-Shot-CoT (Wei et al., 2022), and Self-Consistency (Wang et al., 2023) prompting as baselines. In Two-Shot-CoT prompts, we followed a process similar to ProRAC: first identifying the corresponding objects, then checking the executability of actions, subsequently executing the actions to obtain the final state, and finally comparing the query with the final state to determine the answer. We didn't use ToT for reasoning tasks because the action sequences in reasoning tasks are pre-defined, making multi-path reasoning redundant.

For the plan generation task, we employ DeepSeek-v3.2 and GPT-5-mini as the base models. We compare our A* search strategy against four baselines: LLM+AL, ToT, BFS, and direct prompting (IO). We did not include RAP in our comparison because the MCTS it employs is better suited for simulation games and requires significant computational cost. To ensure a fair comparison, the node budget for A*, ToT, and BFS is strictly limited to 500, with a maximum search depth of 20 and a branch width of 3. For the LLM+AL baseline, we allow the agent to iterate autonomously for 3 rounds without human intervention. For all experiments, the sampling temperature is set to 0.

4.3 Main Results of RAC tasks

Table 1 presents the results of ProRAC and baseline methods on TRAC. While the LLM performs reasonably well on TRAC using pure prompting meth-

Model	Ferry			Swap			Logistics			Rovers			Floortile			Goldminer		
	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL
4o	100	100	100	**100**	100	100	**100**	90	70	**100**	90	80	90	90	40	90	90	90
4o-0CoT	100	100	90	100	90	100	100	90	80	90	90	80	100	80	70	60	90	90
4o-2CoT	100	100	90	100	100	100	100	100	60	70	**100**	60	90	90	70	100	100	**100**
4o-SC	70	80	80	80	50	70	50	90	70	60	80	60	50	80	40	60	80	60
4o-ProRAC	**100**	**100**	**100**	90	**100**	**100**	**100**	**100**	**90**	**100**	90	**80**	**100**	**90**	**90**	**100**	**100**	90
4o-mini	90	100	**90**	60	100	**80**	**100**	80	80	**100**	90	80	100	100	60	**100**	90	**90**
4o-mini-0CoT	100	100	60	70	100	70	100	**90**	50	80	**100**	80	90	100	40	100	**100**	90
4o-mini-2CoT	100	100	80	**90**	100	50	100	80	70	80	90	80	100	100	20	100	90	40
4o-mini-SC	60	80	80	40	60	50	40	90	50	60	90	40	50	80	60	70	90	50
4o-mini-ProRAC	**100**	**100**	70	70	**100**	60	90	**90**	**80**	80	70	**80**	**100**	**100**	**80**	**80**	70	70
v3.2	100	100	100	100	100	**100**	100	**100**	**100**	80	90	80	100	100	100	100	**100**	90
v3.2-0CoT	100	100	100	100	100	100	100	90	90	80	90	80	100	100	100	90	100	100
v3.2-2CoT	90	90	100	100	100	100	100	100	**100**	100	90	90	100	100	100	100	100	100
v3.2-SC	100	100	100	100	100	100	100	100	100	80	80	80	100	100	100	100	100	90
v3.2-ProRAC	**100**	**100**	**100**	**100**	**100**	90	**100**	90	90	**100**	**90**	**90**	90	**100**	**100**	**100**	90	**100**

Table 4: Performance(%) of ProRAC and baseline methods on the ACPBench, BOOL questions. APP=Applicability, PROG=Progression, VAL=Validation. **Bold** indicates the best performance.

ods, its performance improves markedly when enhanced with ProRAC. Considering that the TRAC benchmark is relatively easy, we are more interested in the performance of ProRAC on ActionReasoningBench, where action sequences can be as long as 19 steps.

The performance of ProRAC and baseline methods on ActionReasoningBench is shown in Table 2 and Table 3. As observed from the tables, neither the Zero-Shot nor the CoT methods achieved satisfactory results, whereas ProRAC attained significantly better performance across different domains and tasks, even reaching 100% accuracy in some cases. In some domains, such as Visitall, ProRAC underperforms compared to the baseline methods. We attribute this to the exhaustive connectivity descriptions in the Visitall domain's initial state. Such verbose, information-heavy input hinders LLMs from efficiently extracting state information and performing accurate progression.

Table 4 presents the performance of ProRAC and baseline methods on ACPBench. It can be observed that LLMs perform relatively well on this benchmark, which we primarily attribute to the fact that some of the tasks in ACPBench, such as applicability and progression, are relatively easier than those in the other two benchmarks. However, neither ProRAC nor the baseline methods perform particularly well on the validation task, and their performance in certain domains such as the Logistics and the Floortile declines compared to their performance on the other two tasks. In our experiments, we observed that determining the validity of an action sequence poses a particular challenge for LLMs and they occasionally misclassify valid action sequences as invalid.

ProRAC demonstrates strong performance across different domains and tasks, outperforming prompt-based methods in many scenarios. The results across all three benchmarks validate the robustness and effectiveness of our approach. One possible reason ProRAC is effective is that it decomposes a question that requires multi-step reasoning in an action-wise manner.

4.4 Main Results of The Planning Task

Table 5 reports the performance of various methods on PlanBench with following metrics:

- **Success Rate (SR)**: The percentage of successfully solved instances.
- **Optimality Rate (OR)**: The ratio of optimal plans among all successfully generated plans.
- **Optimal Successful Rate (OSR)**: A composite metric calculated as $SR \times OR$.
- **Average Expanded Nodes (AEN)**: The mean number of nodes expanded during A* search. The lower the better.
- **Average Generated Nodes (AGN)**: The mean number of nodes generated during the search. The lower the better.
- **Average Effective Branching Factor (AEBF)**: EBF is a metric used to evaluate the quality of the heuristic function, calculated only for successfully solved instances. An AEBF value closer to 1 (the lower, the better) indicates a more informed and powerful

Model	Method	SR	OR	OSR	AEN	AGN	AEBF
DeepSeek-v3.2	ProRAC	87.5	77.5	**67.81**	**71**	**141**	**1.65**
	ToT	0	0	0	–	500	–
	BFS	7.5	100.0	7.5	–	131.85	3.27
	LLM+AL	0	0	0	–	–	–
	IO	84.61	63.63	53.83	–	–	–
GPT-5-mini	ProRAC	90	80	**72**	**130**	**183**	**1.71**
	ToT	0	0	0	–	500	–
	BFS	7.5	0	0	–	129.54	3.00
	LLM+AL	0	0	0	–	–	–
	IO	90	70	63	–	–	–

Table 5: Performance(%) of ProRAC and baseline methods on the PlanBench. We randomly sample 40 samples to run the experiments. ToT=Tree-of-Thought, IO=direct prompting. SR=successful rate, OR=optimality rate, OSR=optimal successful rate, AEN=average expanded nodes, AGN=average generated nodes, AEBF=average effective branching factor. **Bold** indicates the best performance.

heuristic. Details about EBF can be found at Appendix A.3.

As illustrated by Table 5, the uninformed BFS fails to solve nearly any instances, and ToT exhibits poor performance, even falling short of direct LLM prompting. In contrast, guided by the heuristic values provided by our trained model, our A* search is capable of identifying solutions while generating significantly fewer nodes.

Regarding LLM+AL, we observe that both language models struggle with the translation task; without human intervention, all instances remain unsolved. The primary failure mode is the generated program's inability to pass the satisfiability (SAT) check. Furthermore, we observe that the LLMs frequently fail to rectify existing errors during multi-round iterations, highlighting the inherent challenges of auto-formalization.

5 Conclusion

In this work, we present ProRAC, the first neuro-symbolic framework that systematically solves RAC and planning problems across multiple benchmarks within a unified progression-based architecture. Extensive experiments on four benchmarks demonstrate that ProRAC achieves strong performance across both reasoning and planning tasks. A key contribution of ProRAC lies in its heuristic-guided search paradigm. Instead of relying on hand-crafted heuristics or prompt-based heuristic elicitation, we propose to train a dedicated heuristic model via an automated pipeline that leverages symbolic planners to generate large-scale training data with exact heuristic signals. This training-based approach establishes a new way of incorporating symbolic planning knowledge into LLM-driven planning systems. Moving forward, empowering LLMs with a deeper planning intuition to autonomously discover structural patterns in complex state spaces will be a cornerstone for next-generation planning systems.

Regarding evaluation, our study focuses on several existing RAC benchmarks based on structured "toy" domains. This choice reflects the current state of the field, as comprehensive and realistic RAC benchmarks remain scarce in the NLP community. However, we view this work as a necessary step toward addressing more complex scenarios: if LLMs cannot reliably solve basic RAC problems in well-defined settings, their success in open-ended, uncertain real-world contexts is even less likely. A crucial direction for future work lies in the development of more realistic and comprehensive RAC benchmarks.

Limitations

First, the heuristic model is fine-tuned on automatically generated symbolic data specific to the target domains. While this ensures high precision within those domains, its zero-shot generalization to entirely unseen planning environments remains a challenge. Second, constrained by computational resources and expenses, the models used in our evaluation are primarily proprietary models. Future iterations of ProRAC could investigate the efficacy of open-source LLMs, and explore techniques such as fine-tuning and reinforcement learning to enhance the RAC capabilities of these models.

References

Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. 2020. Pyperplan. https://doi.org/10.5281/zenodo.3700819.

Blai Bonet and Hector Geffner. 1999. Planning as heuristic search: New results. In *European Conference on Planning*.

Benjamin Callewaert, Simon Vandevelde, and Joost Vennekens. 2025. VERUS-LM: a versatile framework for combining llms with symbolic reasoning. *CoRR*, abs/2501.14540.

Michael Gelfond and Vladimir Lifschitz. 1993. Representing action and change by logic programs. *J. Log. Program.*, 17(2/3&4):301–321.

Enrico Giunchiglia and Vladimir Lifschitz. 1998. An action language based on causal explanation: Preliminary report. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*, pages 623–630. AAAI Press / The MIT Press.

Divij Handa, Pavel Dolin, Shrinidhi Kumbhar, Tran Cao Son, and Chitta Baral. 2025. ActionReasoningBench: Reasoning about Actions with and without Ramification Constraints. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 8154–8173. Association for Computational Linguistics.

Weinan He, Canming Huang, Zhanhao Xiao, and Yongmei Liu. 2023. Exploring the capacity of pretrained language models for reasoning about actions and change. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4629–4643. Association for Computational Linguistics.

Malte Helmert. 2006. The fast downward planning system. *ArXiv*, abs/1109.6051.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Adam Ishay and Joohyung Lee. 2025. LLM+AL: bridging large language models and action languages for complex reasoning about actions. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 24212–24220. AAAI Press.

Henry A. Kautz. 2022. The third ai summer: Aaai robert s. engelmore memorial lecture. *AI Mag.*, 43:93–104.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Harsha Kokel, Michael Katz, Kavitha Srinivas, and Shirin Sohrabi. 2025. ACPBench: Reasoning About Action, Change, and Planning. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 26559–26568. AAAI Press.

Robert A. Kowalski and Marek J. Sergot. 1989. A logic-based calculus of events. *New Generation Computing*, 4:67–95.

Hongyi Ling, Shubham Parashar, Sambhav Khurana, Blake Olson, Anwesha Basu, Gaurangi Sinha, Zhengzhong Tu, James Caverlee, and Shuiwang Ji. 2025. Complex llm planning via automated heuristics discovery. *ArXiv*, abs/2502.19295.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2024. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning*.

Theo Olausson, Alex Gu, Benjamin Lipkin, Cedegao E. Zhang, Armando Solar-Lezama, Joshua B. Tenenbaum, and Roger Levy. 2023. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5153–5176. Association for Computational Linguistics.

OpenAI. 2024. GPT-4o System Card. *CoRR*, abs/2410.21276.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. In *Findings of the*

Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023, pages 3806–3824. Association for Computational Linguistics.

Raymond Reiter. 2001. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT press.

Stuart J. Russell and Peter Norvig. 2020. *Artificial Intelligence: A Modern Approach*, 4 edition. Pearson, Hoboken, NJ.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *ArXiv*, abs/2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou, and Nan Duan. 2021. AR-LSAT: investigating analytical reasoning of text. *CoRR*, abs/2104.06598.

A Appendix

A.1 Complete experiment results of ACPBench

Table 6 reports results on the multiple-choice questions (MCQs) in ACPBench, comparing ProRAC with baseline methods.

A.2 Details About the Heuristic Model Training

Based on our data generation pipeline, we generate 10767 training data, the heuristic values are calculated by using h_{add}.

We use hugging face transformer to build the code. We employ Qwen3-1.7B as the backbone transformer model. The model is fine-tuned on the training data. To accommodate long-horizon sequences, the maximum sequence length is set to 2048 tokens. Other parameters are displayed in table 7. All experiments, including LoRA fine-tuning and model inference, were conducted on a single workstation equipped with an NVIDIA A100 GPU. The host system features 10 vCPUs powered by an Intel Xeon processor and 72GB of system RAM.

A.3 Details about EBF

Effective branching factor (EBF) is a metric used to evaluate the quality of the heuristic function (Russell and Norvig, 2020). If A* search generates n nodes for a problem with solution depth d, then b^* is the branching factor that a uniform tree of depth d must have to contain $n+1$ nodes, satisfying the following relationship:

$$n + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d \quad (3)$$

By solvng this equation we can obtain the EBF. We calculate the Average EBF (AEBF) for instances that can be solved using searching methods. In practice, we employ the bisection method to numerically solve for b^*.

An AEBF value closer to 1 indicates a more informed and powerful heuristic, representing near-optimal search efficiency with minimal redundant exploration.

Model	Ferry			Swap			Logistics			Rovers			Floortile			Goldminer		
	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL	APP	PROG	VAL
4o	90	100	40	**100**	90	**100**	80	100	40	40	80	30	90	90	40	90	100	**90**
4o-0CoT	90	100	40	80	100	60	70	100	50	50	80	40	90	80	60	60	90	50
4o-2CoT	90	100	70	90	100	90	80	100	50	50	80	**50**	90	90	60	90	80	30
4o-SC	100	100	40	100	90	50	80	100	50	90	80	50	90	90	70	90	100	60
4o-ProRAC	**100**	**100**	**90**	90	**100**	70	**100**	**100**	70	**100**	**80**	40	**100**	**90**	**70**	**100**	**100**	60
4o-mini	90	100	30	70	100	70	**60**	90	20	**70**	70	20	80	80	40	**100**	90	**80**
4o-mini-0CoT	90	100	30	70	100	70	60	90	20	70	60	20	80	90	30	70	90	40
4o-mini-2CoT	90	100	40	70	100	70	50	100	40	50	80	50	80	90	30	90	90	40
4o-mini-SC	90	90	40	80	100	60	90	80	40	70	80	30	90	80	40	100	80	30
4o-mini-ProRAC	**100**	**100**	**70**	**80**	90	**70**	**90**	**100**	**60**	**90**	**80**	**50**	**100**	**90**	**60**	**100**	**100**	60
v3.2	100	100	100	100	80	**100**	100	100	90	100	**100**	**100**	100	100	90	**100**	100	90
v3.2-0CoT	100	100	100	80	90	80	90	90	**100**	80	90	90	100	90	90	100	100	**100**
v3.2-2CoT	100	100	100	100	70	100	100	100	90	100	100	90	100	90	100	90	100	100
v3.2-SC	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
v3.2-ProRAC	**100**	**100**	**100**	**100**	**100**	90	**90**	**100**	90	**100**	90	90	**100**	**100**	**100**	90	**100**	90

Table 6: Performance(%) of ProRAC and baseline methods on the ACPBench, MCQ questions. APP=Applicability, PROG=Progression, VAL=Validation. **Bold** indicates the best performance.

LoRA Specs	LoRA rank	64
	LoRA alpha	16
	Dropout Prob.	0.05
Optimization	Optimizer	AdamW
	Peak Learning Rate	2e-4
	Warmup Ratio	0.03
	Learning Rate Scheduler	Cosine
Training Setup	Total Epochs	3
	Per-device Batch Size	4
	Gradient Accumulation Steps	4
	Mixed Precision	FP16
	Random Seed	42

Table 7: Hyperparameters of LoRA fine-tuning.

A.4 Domain Independent Instructions

Initial State Extractor Prompt

You are a PDDL expert, and you are fimilar with the classical IPC domains. Based on the domain description, the current state and the example, extarct all objects' states. The domain of the exmaple may be different from the current state, but the idea of finishing the task is the same. The format of the example must be always the same. Your answer should use the same format as the example to think and generate your answer. Your answer should contain all objects' states, and each object's state should contain all predicates belonging to this object mentioned in the PDDL domain description. Pay attention that different kinds of objects have different predicates, don't mix them up. Besides, Some objects may have some predicates that are not mentioned in the domain description, that means these predicates are false (not true) in the current state. You should also include these predicates correctly with the correct negation. Don't miss any predicates.
So, first you need what kind of objects are mentioned in the PDDL domain description, and what predicates belongs to each object. Second, you need to confirm how many objects are mentioned in the current state. Third, extract all objects' states with all predicates belonging to each object mentioned in the PDDL domain description. Remember you need to return all all objects' states with all predicates belong to them. Finally, organize all objects' states into a

new paragraph as the end of your answer. Each sentences should be a simple sentence that only contain one object, divided by a period. Your last paragraph must use this format: 'box' with no other content, and put all objects' states inside the box. Remember, don't use any markdown formatting and latex format, the box is a strict format requirement, you can't change it, and your final answer must put inside the box.

Question Extractor Prompt

The raw text above contains two parts: an action sequence and a question. Based on the the example, your task is to extract and organize the question from the raw text. The domain of the exmaple may be different from the current state, but the format of the example must be always the same. Your answer should use the same format as the example to think and generate your answer. Normally the text has an action sequence in the front, followed by the question, but not all the time. Sometimes the question will contain an if clause, where another action is mentioned. You should ignore the if-clause and only extract the question.
So, first you need to confirm which parts of the raw text is the action sequence, and which parts is the question. Second, discard the action sequence and extract the question and organize the question into a new paragraph as the end of your answer. Your last paragraph must use this format: 'box' with no other content, and put the question you extracted inside the box. Remember, don't use any markdown formatting and latex format, the box is a strict format requirement, you can't change it, and your final answer must put inside the box.

Executability Check Prompt

You are a PDDL expert, and you are fimilar with the classical IPC domains. Based on the domain description, the example, the current state and the action, determine whether this action is executable or not. The domain of the exmaple may be different from the current state, but the idea of finishing the task is the same. The format of the example must be always the same. Your answer should use the same format as the example to think and generate your answer. An action is executable if and only if all preconditions of this action is satisfied in the current state. If any preconditions of the action is not satisfied, the action is not executable.
So, first you need to confirm objects that are affected by action. Second, confirm the preconditions of this action. Third, check all preconditions one by one.
After your analysis, please your final answer in a new paragraph as the end of your answer. If executable, return True, else return False. Your last paragraph should use this format: 'boxTrue' with no other content, and put your final answer inside the box. Remember, don't use any markdown formatting and latex format, the box is a strict format requirement, you can't change it, and your final answer must put inside the box.

Progression Prompts

You are a PDDL expert, and you are fimilar with the classical IPC domains. Based on the domain description, the current state, the action and the example, take the action and return the new state. The domain of the exmaple may be different from the current state, but the idea of finishing the task is the same. The format of the example must be always the same. Your answer should contain all objects' states, and each object's state should contain all predicates that belongs to this object mentioned in the PDDL domain description, including the properties that is not affected by the action. Remember, your answer should use natural language to describe the state of each object. Besides, some predicates of an object may not be mentioned in the current state, that means these predicates are false (not true) in the current state. You should also include these predicates correctly with the correct negation. Don't miss any predicates.

So, first you need to confirm the effects of this action. Second, you need to confirm which objects are effected by the action. Third, take the action and return all objects' state with all predicates mentioned in the PDDL domain description. Remember you need to return all predicates of all objects, including these that are not affected by the action.
After your analysis, please organize all objects' state as the new state in a new paragraph as the end of your answer. Each sentences should be a simple sentence that only contain one object, divided by a period. Your last paragraph must use this format: 'box' with no other content, and put all objects' states inside the box. Remember, don't use any markdown formatting and latex format, the box is a strict format requirement, you can't change it, and your final answer must put inside the box.

Generate Available Actions

You are a PDDL expert and you are finilar with classical IPC planning domain. Your task is to generate all executable actions available in the current state. An executable action is defined as an action whose execution prerequisites are all satisfied in the current state. Your answer must satisfy all the following requirements:
(1)Each action must be written in PDDL symbolic format, and actions are separated by a period, such as 'pick-up(A). stack(B, C)'. Each action is parallel and does not interfere with one another. (2)The action you give must strictly follow the domain description, especially the preconditions of each action. DON'T generate any action that violates the preconditions at the current state. (3) Do not generate actions that have already been executed in the previous step, and do not generate the inverse actions of the actions executed in the previous step. For example, if the previous action is "(unstack red blue)", then its inverse action would be "(stack red blue)". Such inverse actions should not be generated in order to avoid producing duplicate states.

After your analysis, you should organize all available actions into a new paragraph as the end of your answer without any other content. Your last paragraph must be in the following format: 'box' and put all actions in the box. For example, 'box(pick-up red). (stack yellow blue)'. Remember, don't use any markdown and latex formatting. The only allowed format in your last paragraph of your answer is box.... You can't change the format, this is the strict requirement. DON'T USE ANY MARKDOWN and latex format.