# Acquiring Useful Transitions in Online Relational Model-Based Reinforcement Learning

by

Annie Feng

S.B. Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2023)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

### MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

#### MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

© 2025 Annie Feng. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by:	Annie Feng
	Department of Electrical Engineering and Computer Science
	December 3, 2024
Certified by:	Tomás Lozano-Pérez
	Professor of Electrical Engineering and Computer Science, Thesis Supervisor
Accepted by:	Katrina LaCurts
	Chair, Master of Engineering Thesis Committee

### Acquiring Useful Transitions in Online Relational Model-Based Reinforcement Learning

by

#### Annie Feng

Submitted to the Department of Electrical Engineering and Computer Science on December 3, 2024 in partial fulfillment of the requirements for the degree of

### MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

#### ABSTRACT

Reinforcement learning in environments with large state and action spaces and sparse rewards demands sophisticated exploration strategies. This thesis introduces a practical exploration method for model-based relational reinforcement learning designed for largescale relational domains with sparse rewards. Inspired by goal-literal babbling, our approach utilizes teacher-provided subgoals and demonstrations to guide exploration. We evaluate the method in the Baking-Large domain, which features vast state and action spaces and a challenging narrow-passage problem. Furthermore, we investigate the potential of Large Language Models to assist exploration in this domain and simpler ones, highlighting their capabilities and limitations.

Thesis supervisor: Tomás Lozano-Pérez

Title: Professor of Electrical Engineering and Computer Science

# Acknowledgments

I would like to express my sincere gratitude to Tomas and Leslie for granting me this incredible opportunity. Their kindness, invaluable wisdom, and insightful guidance have been instrumental in shaping my journey. I am deeply thankful to Nishanth, my mentor and collaborator, for his invaluable guidance in refining the research direction and for the many thought-provoking and productive discussions that enriched this work. Finally, I extend my deepest appreciation to my mom, my dad, and my brother, whose unwavering encouragement and support have been the foundation of all my endeavors.

# Contents

Li	st of	Figures	11
Li	st of	Tables	13
1	1 Introduction		
2	Pro	blem Setting	17
	2.1	Environments	17
	2.2	Tasks	18
	2.3	Model: PDDL Operators With Actions	19
	2.4	PDDLGym Environments	19
	2.5	Our Large-Scale Domain: Baking-Large	20
3	$\mathbf{Rel}$	ated Work	21
4	Insi	ghts From Goal-Literal Babbling Exploration	23
	4.1	How GLIB Works	23
		4.1.1 GLIB Relies on Random Actions	23
	4.2	GLIB's Reliance on Random Actions Inspired Demonstrations	24
	4.3	GLIB's Failed Plan Executions Inspired Precondition Targeting	24
	4.4	GLIB's Goal Sampling Inspired Curricula in Our Algorithm	26
	4.5	Why GLIB Is Effective: Working Hypotheses	27
		4.5.1 What Are Useful Transitions to Collect for Learning?	27
5	Ou	r Algorithm For Exploration in Relational Domains	<b>29</b>
	5.1	Overview	29

	5.2	.2 Initialization: Lines 2-9					
	5.3	Training Loop: Lines 10-44	34				
		5.3.1 Automatic Action Selection	34				
		5.3.2 Detecting When to Request the Teacher to Intervene	37				
		5.3.3 Updating the Learned Model	38				
	5.4	How to Select Curricula and Demonstrations	38				
		5.4.1 Selecting curriculum to correct weak preconditions	39				
		5.4.2 Selecting demonstrations	41				
	5.5	Summary	44				
6	Res	sults	45				
	6.1	Exploration Methods Evaluated	45				
	6.2	Results on Long-Horizon Test Planning Problems	46				
		6.2.1 Analysis of Baseline Method Failures	46				
	6.3	Data Efficiency	47				
7	Fut	Future Work 49					
	7.1	Integrating GLIB With Operators From GPT-4	49				
	7.2	Planning Open-Loop with GPT-4	52				
	7.3	Takeaways and Future Work	54				
8 Conclusion		nclusion	55				
A	Dor	nains	57				
	A.1	Baking-Small	57				
		A.1.1 Predicates	57				
		A.1.2 Operators	58				
		A.1.3 Largest Set of Objects in a Problem	62				
	A.2	A.1.3 Largest Set of Objects in a Problem	62 62				
	A.2	A.1.3 Largest Set of Objects in a Problem         Blocks         A.2.1 Predicates	62 62 62				
	A.2	A.1.3       Largest Set of Objects in a Problem         Blocks	<ul> <li>62</li> <li>62</li> <li>62</li> <li>63</li> </ul>				

A.3	Doors		65		
	A.3.1	Predicates	65		
	A.3.2	Operators	66		
	A.3.3	Largest Set of Objects in a Problem	67		
A.4	Minec	raft	70		
	A.4.1	Predicates	70		
	A.4.2	Operators	70		
	A.4.3	Largest Set of Objects in a Problem	73		
A.5	A.5 Baking-Large				
	A.5.1	Predicates	74		
	A.5.2	Operators	77		
	A.5.3	Largest Set of Objects in a Problem	90		
B Pro	mpt T	emplates	93		
<b>B Pro</b> B.1	<b>mpt T</b> Promp	emplates of Templates for Generating Operators	<b>93</b> 93		
<b>B Pro</b> B.1	<b>mpt T</b> Promp B.1.1	emplates of Templates for Generating Operators	<b>93</b> 93 95		
<ul><li>B Pro</li><li>B.1</li><li>B.2</li></ul>	<b>mpt T</b> Promp B.1.1 Predic	emplates         ot Templates for Generating Operators         Few-Shot Chain-of-Thought Prompt Template         ate Descriptions for Baking-Large	<b>93</b> 93 95 104		
<ul><li>B Pro</li><li>B.1</li><li>B.2</li></ul>	mpt T Promp B.1.1 Predic B.2.1	emplates         ot Templates for Generating Operators         Few-Shot Chain-of-Thought Prompt Template         ate Descriptions for Baking-Large         Grounded Literal Descriptions	<ul> <li>93</li> <li>93</li> <li>95</li> <li>104</li> <li>104</li> </ul>		
<ul><li>B Pro</li><li>B.1</li><li>B.2</li></ul>	mpt T Promp B.1.1 Predic B.2.1 B.2.2	emplates         ot Templates for Generating Operators         Few-Shot Chain-of-Thought Prompt Template         ate Descriptions for Baking-Large         Grounded Literal Descriptions         Action Predicate Descriptions	<ul> <li>93</li> <li>93</li> <li>95</li> <li>104</li> <li>104</li> <li>108</li> </ul>		
<ul><li>B Pro</li><li>B.1</li><li>B.2</li></ul>	<b>mpt T</b> Promp B.1.1 Predic B.2.1 B.2.2 B.2.3	emplates         ot Templates for Generating Operators         Few-Shot Chain-of-Thought Prompt Template         ate Descriptions for Baking-Large         Grounded Literal Descriptions         Action Predicate Descriptions         Lifted Action Literal Variable Descriptions	<ul> <li>93</li> <li>93</li> <li>95</li> <li>104</li> <li>104</li> <li>108</li> <li>109</li> </ul>		
<ul><li>B Pro</li><li>B.1</li><li>B.2</li></ul>	<b>mpt T</b> Promp B.1.1 Predic B.2.1 B.2.2 B.2.3 B.2.4	emplates of Templates for Generating Operators	<ul> <li>93</li> <li>93</li> <li>95</li> <li>104</li> <li>104</li> <li>108</li> <li>109</li> <li>112</li> </ul>		
<ul> <li>B.1</li> <li>B.2</li> <li>B.3</li> </ul>	<b>mpt T</b> Promp B.1.1 Predic B.2.1 B.2.2 B.2.3 B.2.4 Promp	emplates         ot Templates for Generating Operators         Few-Shot Chain-of-Thought Prompt Template         ate Descriptions for Baking-Large         Grounded Literal Descriptions         Action Predicate Descriptions         Lifted Action Literal Variable Descriptions         Goal Descriptions         ot Templates for Generating Plan Sketches Open-Loop in Baking-Large	<ul> <li>93</li> <li>93</li> <li>95</li> <li>104</li> <li>104</li> <li>108</li> <li>109</li> <li>112</li> <li>113</li> </ul>		
<ul> <li>B.1</li> <li>B.2</li> <li>B.3</li> <li>B.4</li> </ul>	mpt T Promp B.1.1 Predic B.2.1 B.2.2 B.2.3 B.2.4 Promp Promp	emplates         ot Templates for Generating Operators         Few-Shot Chain-of-Thought Prompt Template         ate Descriptions for Baking-Large         Grounded Literal Descriptions         Action Predicate Descriptions         Lifted Action Literal Variable Descriptions         Goal Descriptions         ot Templates for Generating Plan Sketches Open-Loop in Baking-Large         ot Templates for Grounding Plans	<ul> <li>93</li> <li>93</li> <li>95</li> <li>104</li> <li>104</li> <li>108</li> <li>109</li> <li>112</li> <li>113</li> <li>116</li> </ul>		

### List of Figures

- 4.1 In "GLIB\_G1 with demos" and "GLIB\_L2 with demos", the first 61 actions are demonstrations from a teacher. From these demonstrations, initial operators for every action predicate are learned and used for planning in GLIB<sup>1</sup>. GLIB with demonstrations uses fewer environment interactions to solve the same tasks and solves harder tasks than GLIB without demonstrations. . . . 25
- 4.2 Success rate vs. number of environment interactions with failed actions in plans marked with a green dot. This plot shows the performance of GLIB on the toy Baking domain in PDDLGym. Notice that out of all the points where success rate increases, more often than not, green dots are located at those points. This shows that an increase in the success rate is often directly caused by a model update after a plan executes unexpectedly because of incorrect preconditions in the executed operator.
  26
- 6.2 Success rates on 12 short-horizon tasks (solvable in three or fewer actions) with goals reachable in the training episodes. Our method demonstrates superior data efficiency compared to both GLIB variants and random action selection.
  48

- 7.1 Success rate curves comparing vanilla GLIB to GLIB initialized with LLMgenerated operators. In domains where the LLM generates almost all ground truth operators accurately (Minecraft and Blocks), learning is significantly accelerated. However, in the Baking and Doors domains, the LLM generates fewer correct operators or none at all, leading to fewer useful transitions collected (Baking) or even distraction from better action selection (Doors).
- 7.2 ChatGPT's proposed action sequence to bake a cake in Baking-Large. In this example shown, the proposed sequence of action predicate names can form a valid plan to solve the goal. However, the groundings of the predicates in the later responses (not shown) make the plan fail.
  53

51

# List of Tables

2.1	Domain size comparison by number of predicates, number of operators, and	
	maximum number of objects.	20

# Chapter 1

## Introduction

Learning the transition model is fundamental in model-based reinforcement learning. In this approach, an agent interacts with its environment to learn an approximate model of the environment's dynamics. This learned model can then given as input to a planner to achieve a goal.

In complex environments with large state and action spaces, learning effective models for planning is challenging. For instance, consider a robot navigating an unfamiliar environment or an autonomous AI agent attempting a new task on the web. In both scenarios, efficiently exploring to learn a useful model for planning to complete tasks requires directed trial-anderror in small subspaces of the state space. Relational transition models, which define lifted relations between objects, offer a promising solution. By design, these models generalize to new problems with different objects and compactly represent large-scale domains, enabling the handling of environments with numerous objects. Following on prior work (Chitnis, Silver et al., 2020; Lang, Toussaint, and Kersting, 2012), we adopt *noisy deictic rules* [13] as the transition model representation in relational model-based reinforcement learning.

This thesis proposes a new exploration algorithm for relational model learning, inspired by experiments with the goal-literal babbling (GLIB) algorithm [6]. Originally designed for small, simple domains where effective actions can be discovered through near-exhaustive enumeration, GLIB struggles in larger, more complex environments. To demonstrate this limitation, we introduce a new domain, Baking-Large, with over 2<sup>1600</sup> grounded states and over 2700 grounded actions. Baking-Large also features a narrow-passage problem, requiring a long sequence of specific actions to access critical parts of the state space. In our experiments, we show that our method is much more effective than GLIB on this domain.

Our method combines planning and teacher guidance to direct exploration, drawing inspiration from goal-literal babbling (GLIB) [6]. Like GLIB, our approach alternates between planning and another mode—teacher guidance in our case. We plan to the agent's learned operators' preconditions, and unlike GLIB, which defaults to random actions when no plans are found, our method requests teacher-provided demonstrations or subgoals. Furthermore, instead of planning to uniformly sampled random goals, our method alternates between planning to teacher-defined subgoals and learned preconditions.

We also explore the potential of Large Language Models (LLMs) for efficient exploration in relational domains, examining how they can propose symbolic operators for goal-literal babbling and generate plans to provide demonstrations that accelerate model learning. These experiments, conducted in Baking-Large and simpler environments from PDDLGym [34], highlight promising directions for future research.

The rest of the thesis is organized as follows. First, we describe our problem setting and introduce the domains used in our experiments. Next, we discuss related work. We then present our insights into GLIB that led to our algorithm. Following this, we detail our algorithm and the teacher's role, illustrating with examples from Baking-Large. We then present the results of our algorithm on Baking-Large. For future work, we explore initial experiments with Large Language Models, highlighting their potential for gathering demonstrations. Finally, we conclude with a summary and suggestions for future research.

# Chapter 2

### **Problem Setting**

Our problem setting features discrete states and discrete actions. It is online, object-oriented, deterministic, goal-based, and fully observable. In this section, we formally define the environments and tasks, followed by an introduction to the domains used in our experiments, including our own Baking-Large environment.

### 2.1 Environments

An environment is defined as  $\langle S, A, I, T, P, O, P, Q \rangle$ .

As in a typical RL environment, S is the state space, A is the action space,  $\mathcal{I}$  is the initial state distribution, T is the maximum episode length as the maximum number of actions, and P(s'|s, a) is the deterministic transition model where  $s, s' \in S$  and  $a \in A$ . The agent does not know the transition model, but it does know S and A. As it takes actions in the environment, the agent perfectly observes states sampled from the transition model P. The agent interacts with the environment episodically.

 $\mathcal{O}$  is the finite set of objects for the environment. Between episodes, the set of objects  $o \subset \mathcal{O}$  may differ, but within each episode, o is fixed. Each object has a semantically meaningful name, such as "pan-0".

The state space S is relational with respect to a known, finite set of predicates  $\mathcal{P}$ , and the action space  $\mathcal{A}$  is relational with respect to a known, finite set of predicates  $\mathcal{Q}$ . Predicates are Boolean-valued functions. A predicate applied to objects or variables is a ground literal

or lifted literal, respectively. Under the closed world assumption, each state  $s \in S$  is a conjunction of positive ground literals constructed from the predicates in  $\mathcal{P}$ . Each action  $a \in \mathcal{A}$  is a ground literal constructed from a predicate in  $\mathcal{Q}$ . Each predicate has a semantically meaningful name, such as "is-raw-egg-yolk".

An environment is associated with a set of tasks that the agent sees at training time, and a set of tasks that the agent sees at testing time. The test tasks are generally harder than the train tasks, containing more objects and requiring longer horizon plans to achieve goals. In the Baking-Large domain, none of the test tasks' goals can be achieved within the training episodes.

### 2.2 Tasks

A task is specified as  $\langle s_0, o, g \rangle$ , where  $s_0 \in S$  is the initial state,  $o \subset O$  is the set of objects that stays constant throughout the episode, and  $g \in S$  is the goal state.

During training, the agent attempts training tasks episodically. An episode begins in state  $s_0$ , and the agent takes a sequence of actions  $a_0, a_1, ..., a_n \in \mathcal{A}$  until a termination condition is satisfied: the agent either reaches the goal state g or the episode ends after the  $T^{th}$  action. Unless we specify otherwise, the next episode is randomly sampled from the training tasks and begins. In our experiments, we stop training once a maximum number of actions have been taken or when the agent achieves 100% on the test tasks.

As the agent explores and learns, we periodically evaluate the learned model on test tasks. For each test task, if the model generates a plan based on the training episodes, we execute it open-loop and consider it successful if the goal is achieved. We use test tasks to report the effectiveness of our methods, but in an application, the agent could do without periodic evaluation and just keep exploring and improving the learned model.

### 2.3 Model: PDDL Operators With Actions

The model that the agent learns is a set of PDDL operators with actions<sup>1</sup>. Each PDDL operator is associated with a lifted action literal defined as an action predicate applied to a subset of the operator's parameters. Our operators' preconditions are conjunctions at the top level. For each operator, we append the lifted action literal to the overall conjunction in the preconditions of the operator.

We define ground truth operators, which the agent does not have access to, for each environment. Executing the action literal in the environment causes deterministic effects defined by the ground truth operator whose preconditions are satisfied. If no ground truth operator has its preconditions satisfied, then the executed action doesn't change the state of the environment.

### 2.4 PDDLGym Environments

We use four existing domains from PDDLGym[34] in our experiments with Large Language Models: Baking (which we call Baking-Small), Minecraft, Blocks, and Doors. These domains are relatively small-scale compared to Baking-Large. The operators, predicates, and objects for these domains are detailed in Appendix A.

Baking-Small presents a simplified version of the narrow-passage problem. Here, the agent bakes cakes and souffles, requiring specific, sequential actions like adding ingredients to a pan, mixing them, putting the pan in the oven, and starting the oven. Requiring a precise and long sequence of actions makes random action selection not as effective in this domain as in the Blocks, Minecraft, and Doors domains.

Baking-Large scales up this challenge significantly, introducing more ingredients and additional steps for baking cakes and souffles. The extended narrow-passage amplifies the difficulty posed in Baking-Small.

<sup>&</sup>lt;sup>1</sup>We use PDDL 1.2. Our requirements are typing and disjunctive preconditions.

### 2.5 Our Large-Scale Domain: Baking-Large

We designed a domain called Baking-Large for our experiments, where the agent bakes cakes and souffles and serves them on plates. To bake a cake, the agent must preheat the oven with the appropriate cake settings, mix five ingredients in a bowl, transfer the mixture to a pan, place the pan in the oven, and start the oven with cake-baking settings. To bake a souffle, the agent must preheat the oven with souffle settings, crack and separate eggs, mix several ingredients in a bowl, beat egg whites, fold them into the mixture, pour the mixture into a pan, place the pan in the oven, and start the oven with souffle-baking settings.

We list the operators, predicates, and objects for this domain in Appendix A. A comparison of the scale of Baking-Large with other domains is provided in Table 2.1. To further demonstrate the scale of this domain, consider the following:

- 1. The largest operator has 21 literals in the precondition and 25 literals in the effects.
- 2. Each task has 30-40 objects.
- 3. The minimum length of plans that solve test tasks range from 24 to 28 actions.

Domain Name	State Predicates	Action Predicates	Operators	Objects
Blocks	6	4	4	7
Baking-Small	14	8	8	11
Minecraft	9	5	5	35
Doors	5	2	2	93
Baking-Large (ours)	50	18	22	41

Table 2.1: Domain size comparison by number of predicates, number of operators, and maximum number of objects.

# Chapter 3

## **Related Work**

Our work is related to several existing lines of work in the literature.

**Exploration in Reinforcement Learning (RL).** Efficient exploration of state spaces to enable rapid learning is a core challenge in RL [2, 19]. Various exploration strategies have been developed to improve learning efficiency [5, 8, 15, 24, 29, 33, 38]. Of these, our work is most related to the *model-based* approaches [24, 33, 38], which attempt to learn a transition model during exploration, which is then used for planning tasks at test time. Our work differs by addressing the *relational* setting, where state spaces are represented as relationships among objects so that symbolic planners can be used to plan with a given transition model.

Exploration in relational domains. Recent work has investigated exploration in relational domains that support efficient symbolic planning [6, 18, 20, 21, 26, 31, 32]. Among these, our work builds on GLIB [6], which explores relational domains by randomly sampling possible goals and attempting to plan to achieve them. However, GLIB and most similar works primarily focus on small-scale domains with limited predicates and objects. By contrast, we are interested in large scale domains, where the space of possible goals that could be sampled is extremely large. We empirically evaluate GLIB's performance in such settings and focus on developing a new exploration approach to handle these challenges effectively.

We explore the potential of Large Language Models (LLMs) [28] to enhance exploration in relational domains. Several recent works have studied using LLMs for exploration in nonrelational domains [10, 22, 27]. Another line of work has studied using LLMs for planning directly [14, 37, 39] and discovered that symbolic planners generally outperform LLMs at planning tasks. Consequently, we use LLMs to aid exploration rather than perform direct planning to achieve goals. Similar to Wong et. al. [41], we investigated using an LLM to propose symbolic operators; however, we use the operators only for exploration and learn operators from data for evaluation, while [41] evaluates and discards operators proposed by the LLM.

We also investigate leveraging a small set of human demonstrations to enable more efficient exploration. This idea has been explored previously in the context of deep reinforcement learning [25, 30, 40]. In relational reinforcement learning, using human demonstrations to aid exploration has been proposed for model-free methods [9] and for model-based methods [23]. Our work has similarities with Martinez et. al. [23]: we ask for teacher demonstrations and/or subgoals when planning fails. However, while Martinez's method builds on REX, ours is inspired by GLIB and addresses the larger and more complex Baking-Large domain. Learning to plan. Recent years have seen significant interest in learning models from data to enable effective planning [1, 4, 7, 12, 16, 17, 35, 36, 41]. Several previous works attempt to learn symbolic predicates, operators, and other planning components for longhorizon tasks [12, 16, 35]. Many of these works take place entirely in the offline setting, where only demonstrations are provided [4, 17, 35]. The works that operate in the online setting generally do not focus on exploration challenges and rely on naive strategies for exploration [12, 16]. Our work is most related to approaches that assume given predicates and focus on learning symbolic operators [3, 13, 17, 41]. We adopt the symbolic operatorlearning algorithm from 13 (though other operator-learning algorithms could certainly be used here instead) and focus on collecting transition data online so that useful operators can be learned with as little data as possible.

# Chapter 4

# Insights From Goal-Literal Babbling Exploration

Our method was inspired by our empirical study of goal-literal babbling exploration (GLIB) [6]. In this section, we present our insights that motivated our algorithm.

### 4.1 How GLIB Works

GLIB is an exploration strategy for relational domains that uses planning to guide exploration [6]. To select an action, the agent uniformly samples a goal from the unvisited state space and attempts to find a plan under the learned model. If a plan is found, GLIB executes it until it completes or fails. If no plan is found after N attempts at sampling and planning (where N is a hyperparameter), GLIB defaults to randomly selecting an action from the action space.

#### 4.1.1 GLIB Relies on Random Actions

GLIB's strategy is a mixture of selecting random actions and planning to randomly-selected, unvisited goal states. At the beginning of online learning, there are no operators, so GLIB relies on random actions to get a nontrivial transition upon which to learn a first operator, which can then be used for planning. However, even after the first operators are learned, it's possible that many plans can't be made with those operators alone, which often happens in complex domains such as Baking-Large. As a result, GLIB will continue relying on random actions to find other nontrivial transitions.

In large domains, random actions have a very low probability of succeeding. We created Baking-Large to feature numerous action predicates, high predicate arities, many objects, and complex preconditions. Our experiments in Baking-Large confirmed that GLIB and random action selection perform poorly in such environments. Unlike GLIB, our method avoids relying on random actions entirely.

## 4.2 GLIB's Reliance on Random Actions Inspired Demonstrations

Unlike GLIB, which relies on random actions to learn initial operators, our method leverages teacher-provided demonstrations. This approach was motivated by an experiment to confirm our hypothesis that providing GLIB with transitions with unseen effects as initial demonstrations would accelerate operator learning. As shown in Figure 4.1, this hypothesis was validated in our Baking-Large domain. The same trend was observed in the following small-scale domains in PDDLGym: Blocks, Doors, Easygripper, Minecraft, Travel, and Baking [34].

# 4.3 GLIB's Failed Plan Executions Inspired Precondition Targeting

We hypothesized that the following event is a catalyst for model learning. We observed the following pattern in GLIB: first the agent makes a plan to a randomly-selected goal, then it executes an operator with incorrect preconditions, then the agent observes a transition with unexpected effects, and finally the agent calls the model learner to update the incorrectly

<sup>&</sup>lt;sup>1</sup>In our experiments using demonstrations, when sampling goals in GLIB, we do not rule out goals visited during the demonstrations unless they were achieved after the demonstrations.



(a) GLIB with initial demonstrations vs. vanilla GLIB on the long-horizon test tasks in Baking-Large.

(b) GLIB with initial demonstrations vs. vanilla GLIB on short-horizon training tasks in Baking-Large.

Figure 4.1: In "GLIB\_G1 with demos" and "GLIB\_L2 with demos", the first 61 actions are demonstrations from a teacher. From these demonstrations, initial operators for every action predicate are learned and used for planning in GLIB<sup>1</sup>. GLIB with demonstrations uses fewer environment interactions to solve the same tasks and solves harder tasks than GLIB without demonstrations.

learned operator. This model change then increases the success rate. We marked the observation of this pattern using green dots on success rate curves such as in Figure 4.2. Across many domains, we observed that this event often coincidences at iterations where the success rate increases. Inspired by this observation, we wanted our method to observe transitions that reveal incorrect preconditions as much as possible.

In our method, we plan to the preconditions of a learned operator before executing the operator. In this way, we purposefully cause the observed pattern. This is unlike GLIB, which relies on chance to sample goals that involve an operator with incorrect preconditions in the plan.



Figure 4.2: Success rate vs. number of environment interactions with failed actions in plans marked with a green dot. This plot shows the performance of GLIB on the toy Baking domain in PDDLGym. Notice that out of all the points where success rate increases, more often than not, green dots are located at those points. This shows that an increase in the success rate is often directly caused by a model update after a plan executes unexpectedly because of incorrect preconditions in the executed operator.

# 4.4 GLIB's Goal Sampling Inspired Curricula in Our Algorithm

The toy Baking domain presents a narrow passage problem: interesting parts of the state space can only be accessed after executing a lengthy sequence of prerequisite actions. We observed that GLIB addresses this challenge by serendipitously sampling a specific sequence of goals, achieving them one by one to get across the narrow passage, and then executing a random action in the newly accessible state space that enhances the model. Similarly, our method prescribes curricula—sequences of subgoals—for the agent to accomplish. Instead of relying on random actions, a teacher reasons about the situation and provides demonstrations.

### 4.5 Why GLIB Is Effective: Working Hypotheses

#### 4.5.1 What Are Useful Transitions to Collect for Learning?

We identify two types of useful transitions that an exploration strategy collects:

- 1. A transition that shows a counterexample to a learned operator's preconditions.
- 2. A transition with a previously unseen set of effects.

GLIB collects the first type of transitions by executing plans that fail and the second type primarily through random actions. Early in training, when no operators have been learned, random actions are used to obtain the second type of transitions. Once some operators are learned, plans can be made using those operators, and executing a random action immediately after achieving a goal can help uncover the second type of transitions, particularly those within narrow passages. The second type of transitions could also be collected when a mistake in a plan leads to discovering new effects, allowing the definition of a new operator. However, in almost all of our observations, failed actions in a plan did not produce any effects in our simulation.

When a plan fails, the agent improves the model because the observed effects are different than the effects that the operator in the plan describes. In these cases, the preconditions of the operator were wrong, and collecting the transition with unexpected effects informs the operator learning algorithm to update the failed operator's preconditions<sup>2</sup>.

We believe GLIB is more effective than random primarily because it collects failed plan executions that directly lead to improvements in preconditions (collecting the first type of transitions). To illustrate this, we created plots like Figure 4.2 to show that success rate sometimes increases immediately after the model is updated using transitions from the failed

<sup>&</sup>lt;sup>2</sup>Though depending on the learning algorithm, in some cases the preconditions may not update. We observed this problem when learning negated preconditions for our most complex operators in Baking-Large, which are the operators associated with the *use-stand-mixer* action predicate.

plans. We also observed this correlation in other small-scale domains: Baking-Toy, Blocks, Minecraft, and Travel in PDDLGym[34].

In addition, we believe that GLIB's goal sampling helped get across narrow passages by occasionally sampling good goal sequences in Baking-Small. After the agent achieves the last goal in the sequence, executing a random action helps get the second type of transitions.

We use the insights from studying GLIB to develop our method: we use planning to fix incorrectly learned preconditions and collect the first type of transitions, and we rely on curricula and demonstrations to get the second type of transitions.

# Chapter 5

# Our Algorithm For Exploration in Relational Domains

The Baking-Large domain is too large and complex for random actions or GLIB to gather enough useful transitions to learn a model capable of long-horizon planning. This motivated us to develop a new algorithm that performs well in Baking-Large. In this section, we present the main contribution of this work: an exploration method designed to efficiently learn action abstractions in large, complex domains like Baking-Large.

### 5.1 Overview

At a high level, our method builds on the structure of GLIB, which alternates between random action selection and planning to recognize mistakes in the learned model. Our algorithm alternates between two modes: (1) teacher guidance and (2) planning.

In the teacher guidance mode, exploration is directed using a combination of demonstrations and subgoals provided by a teacher. As discussed in Section 5.4, a well-designed curriculum can elicit plan execution errors that improve the model. Subgoals can also guide the agent to specific states where the teacher can provide critical demonstrations.

In planning mode, the algorithm focuses on planning to either a learned operator's preconditions or a teacher-provided subgoal. For each learned operator, we attempt to plan to its randomly grounded preconditions and execute the operator to identify precondition errors. If all learned operators have either been verified or lack plans to their preconditions, the state is reset to the last achieved subgoal, and planning continues toward the next teacher-provided subgoal.

We provide the high level algorithm in Algorithm 1. Following the pseudocode, we describe our algorithm in detail in this section<sup>1</sup>.

 $<sup>^{1}</sup>$ An implementation of this algorithm should not only follow the pseudocode but also take into account the remarks in this chapter that are not conveniently expressed in the pseudocode.

```
Algorithm 1 Learn and Explore with Expert Demonstrations and Curricula
 1 procedure LEARN_AND_EXPLORE(\mathcal{D}_{init}, Env)
 \mathbf{2}
        transition data \leftarrow \mathcal{D}_{init}
 3
        model \leftarrow LEARNMODEL(transition data)
        state \leftarrow Env.RESET()
 4
 5
        \mathbf{g} \leftarrow []
                                                                                    \triangleright List of subgoals
 6
        last achieved subgoal state \leftarrow state
        prev plan \leftarrow None
  7
 8
        prev action \leftarrow None
 9
        generalized to test tasks \leftarrow False
10
        while not generalized to test tasks do
            (action, plan, finished plan to preconds, is planning to preconds)
11
                                                                                                     \leftarrow
    GETACTIONBYPLANNING(model, state, \mathbf{g})
            if not AGENTISSTUCK(action, plan, prev_action, prev_plan) then
12
13
                next state \leftarrow Env.STEP(action)
                transition data.APPEND((state, action, next state))
14
               plan to precords failed \leftarrow CHECKPLANFAILED(is planning to precords,
15
    state, next state)
               state \leftarrow next\_state
16
               if (not is planning to preconds) and SUBGOALACHIEVED(\mathbf{g}[0], state) then
17
                   \mathbf{g}.POP(0)
18
19
                   last achieved subgoal state \leftarrow state
                end if
20
               if finished plan to preconds or plan to preconds failed then
21
                   state \leftarrow Env.RESET(last achieved subgoal state)
22
23
               end if
                prev action \leftarrow action
24
               prev plan \leftarrow plan
25
            else
26
27
                (actions, \mathbf{g}_{new}) \leftarrow ASKEXPERT()
                for action in actions do
28
29
                   next state \leftarrow Env.STEP(action)
                   transition data.APPEND((state, action, next state))
30
31
                   state \leftarrow next state
               end for
32
33
               if \mathbf{g}_{new} is not None then
34
                   \mathbf{g} \leftarrow \mathbf{g}_{\text{new}}
                   last achieved subgoal state \leftarrow Env.RESET()
35
                end if
36
               state \leftarrow Env.RESET(last achieved subgoal state)
37
                generalized to test tasks \leftarrow EVALMODEL(model)
38
            end if
39
40
            if not model.FITSALLDATA(transition data) then
                model \leftarrow LEARNMODEL(transition data)
41
            end if
42
        end while
43
44 end procedure
                                                  31
```

### 5.2 Initialization: Lines 2-9

#### Initial demonstrations $\mathcal{D}_{init}$ and initial operators

#### Algorithm 1: Line 2

We initialize the dataset of transitions with demonstrations  $\mathcal{D}_{init}$  provided by a teacher. Each demonstration is a tuple: *(state, action, next state)*. The agent learns the initial operators from this set of demonstrations in a call to LearnModel.

As an example, in Baking-Large, we (as the teacher) provided these demonstrations in the following way. We created four training episodes. In each episode, we execute one plan to the goal for demonstrations. The descriptions of the goals are as follows:

- 1. Bake a cake.
- 2. Bake a souffle.
- 3. Bake a cake and place it on a plate.
- 4. Bake a cake and a souffle.

After the initial set of demonstrations were collected, we chose to have the agent collect all subsequent data in the fourth episode. This episode is the most complicated of the training episodes: it has a state space that is a superset of all the other training episodes' state spaces.

#### Model learning

#### Algorithm 1: Line 3

Our implementation of LearnModel is a probablistic rule learner [13] which performs a greedy search over rule sets to maximize a score function. Since we are interested in deterministic domains, we determinize each learned probablistic rule using its most probable effect set to create an operator for each rule. The set of operators constitutes our model.

#### **Environment Reset**

#### Algorithm 1: Line 4

We call Env.Reset() to get the first observation in our episode. From this point onward, our algorithm remains within a single episode, but the environment can be reset to any state that is reachable from the episode's starting state. In Algorithm 1, we denote resetting to a specific state s as Env.Reset(s) and denote resetting to the episode's starting state as Env.Reset().

#### Initialize variables

#### Algorithm 1: Lines 5-9

We initialize five variables:

- 1. g: the list of subgoals that we will get from the teacher, initialized as an empty list.
- 2. *last\_achieved\_subgoal\_state*: the state at which the last achieved subgoal was first achieved, initialized to the starting state in the episode. We will reset the environment back to this state after executing a plan to a learned operator's preconditions. If there are no subgoals given by the teacher yet, the start of the episode is the reset point.
- prev\_plan: the previous plan found to a subgoal or preconditions, initialized as None. This is used for detecting if the agent should request intervention from the teacher later in the algorithm.
- prev\_action: the previous action taken in the environment, initialized as None. This
  is used for detecting if the agent should request intervention from the teacher later in
  the algorithm.
- 5. generalized\_to\_test\_tasks: a Boolean that signifies if the learned model passes the test tasks, initialized to False. If this variable is true, we exit the training loop.

### 5.3 Training Loop: Lines 10-44

In the training loop, the agent decides which actions to take, executes them, perfectly observes the effects, and updates the learned model. The main contribution of this thesis is the exploration strategy: we first try to automatically select the action by planning, but when the agent cannot plan to improve the learned model, we ask a teacher to provide demonstrations or curricula until the automatic strategy can take control again.

#### 5.3.1 Automatic Action Selection

#### Algorithm 1: Line 11

In the automatic strategy, the agent tries to automatically pick an action by planning<sup>2</sup> to learned preconditions or teacher-provided subgoals. We give pseudocode for the GetActionByPlanning procedure in Algorithm 2.

Here's a walkthrough of the pseudocode:

- 1. **Operator Selection** (*Algorithm 2, Lines 2, 26-27*): First try planning to the preconditions of the learned operators. Select an operator from the set of learned operators that has not yet been executed as part of a plan.
- 2. Grounding Preconditions (*Algorithm 2, Lines 31-37*): Randomly ground the operator's lifted preconditions in the current state. This grounded precondition becomes the goal to plan toward. Append the grounded action to the end of the plan to execute the operator after achieving the preconditions. If a plan is found, save its state (Lines 16, 31) and return it (Lines 33, 4).
- 3. Planning Attempts (Algorithm 2, Line 30): If no plan is found, attempt planning with up to N = 200 (a tunable hyperparameter) additional random groundings of the precondition.
- 4. Fallback to Teacher Subgoals (*Algorithm 2, Lines 7-13*): If still no plans are found, switch to planning toward the next subgoal provided by the teacher. If a plan is found,

<sup>&</sup>lt;sup>2</sup>In our implementation, we used the FastDownward [11] planning system.

return it.

- 5. Requesting Teacher Assistance (*Algorithm 2, Line 14*): If no subgoals are provided or no plan to the subgoal can be found, we return values that indicate that we need to request new subgoals or demonstrations from the teacher.
- 6. **Plan Execution** (*Algorithm 2, Lines 22-25*): When a plan is found, we save it and execute it action by action until it either successfully completes or fails.
- 7. Handling Plan Failures (*Algorithm 2, Line 19*): If an action in the plan fails to execute (e.g., it produces no effects in our simulation), stop following that plan.

This structured approach integrates planning to learned preconditions, planning to subgoals, and requesting teacher assistance for exploration. Algorithm 2 Get Action by Planning to Learned Preconditions or Expert-Given Subgoals

1 procedure GETACTIONBYPLANNING(model, state, g) action, plan, finished \_plan\_to\_preconditions  $\leftarrow$  TARGETPRECONDITIONS(model,  $\mathbf{2}$ state) if action is not None then 3 **return** (action, plan, finished plan to preconditions, True) 4 5end if  $plan \leftarrow None$ 6 7 if LEN(g) > 0 then 8  $plan \leftarrow GETPLANTOGOAL(model, state, \mathbf{g}[0])$ 9 end if if plan is not None then 10 action  $\leftarrow$  plan |0|11 return (action, plan, False, False) 1213end if 14 return (None, [], False, False) end procedure 1516**procedure** TARGETPRECONDITIONS(model, state) 17 **State Variables:** 18 19current plan: Plan to preconditions being executed. If the plan fails during execution, this is automatically set to an empty list. 20Hyperparameter: N TRIES  $\leftarrow 200$  $\triangleright$  Number of tries to sample a random grounding of the 21preconditions 22if LEN(current plan) > 0 then is last action in plan  $\leftarrow$  LEN(current plan) == 1 23return current plan.POP(0), is last action in plan 24end if 2526for operator in model do 27if HASEXECUTEDPLANFOROPERATOR(operator) then continue 28end if 2930 for i in range $(N\_TRIES)$  do (ground preconditions, ground action) SAM-31 $\leftarrow$ PLE GROUND PRECONDITIONS(operator.lifted preconditions) 32 $plan \leftarrow GETPLANTOGOAL(model, state, ground preconditions)$ if plan is not None then 33 current plan  $\leftarrow$  plan + [ground action] 34is\_last\_action\_in\_plan  $\leftarrow$  LEN(current plan) == 1 35 $a \leftarrow current plan.POP(0)$ 36 return a, current plan, is last action in plan 37 38 end if end for 39 end for 40return (None, None, False) 41

```
42 end procedure
```
### 5.3.2 Detecting When to Request the Teacher to Intervene

#### Algorithm 1: Line 12

If AgentIsStuck returns True, the agent requests teacher intervention; otherwise, it executes the action returned by GetActionByPlanning. If GetActionByPlanning returns None, indicating that the automatic strategy has failed, AgentIsStuck will also return True, prompting a request for teacher intervention. Additionally, if the agent repeatedly executes a failed plan to a subgoal without causing a model update, AgentIsStuck will return True.

procedure AGENTISSTUCK(action, plan prev_action, prev_plan)		
auto_explore_failed $\leftarrow$ (action is None)		
repeating_plan $\leftarrow$ (plan == prev_plan) and (action == prev_action)		
<b>return</b> auto_explore_failed or repeating_plan		
end procedure		

#### AgentIsStuck returns False: execute the next action in the plan

#### Algorithm 1: Lines 13-25

If the agent does not need guidance from the teacher, the agent proceeds to execute the next action in the plan.

Upon the completion or execution failure of a plan from TargetPreconditions in Algorithm 2, the state is reset to the last successfully achieved subgoal.

#### AgentIsStuck returns True: request intervention from a teacher

#### Algorithm 1: Lines 26-39

When AgentIsStuck returns True, the teacher is requested to provide additional demonstrations and/or design a curriculum following the guidelines in Section 5.4.

To collect demonstrations, the teacher provides a plan to be executed from the current state. The agent executes the sequence of actions in the environment, generating (state, action, next\_state) tuples as demonstrations. After receiving the demonstrations, the agent resets (Line 37) to the last achieved subgoal—or the start of the episode if no subgoals exist or a new curriculum was received—and attempts the automatic strategy again in the next iteration of the while loop (Line 12). Collecting effective demonstrations causes a model update (Lines 40-41), enabling the agent to plan successfully with the updated model.

The curriculum from the teacher is a sequence of subgoals that the agent plans to achieve in order. It is designed either to guide the agent to make plans that result in failures, which improve the model, or to lead the agent to specific states where plans from the teacher can be executed.

In our setting, the training loop concludes when evaluation confirms that the learned model enables an automated planner to generate open-loop plans that successfully solve planning problems with novel initial states and goals not encountered during training. In other settings, the loop may terminate based on different evaluation criteria to extract the learned model under varying conditions. Our method ensures that teacher intervention is requested after a finite time, at which point the model can be evaluated. In practice, our method requests teacher guidance periodically, so the model is periodically evaluated.

#### 5.3.3 Updating the Learned Model

#### Algorithm 1: Lines 40-41

The model is updated whenever the agent observes new transitions that are not explained by the learned rules from our operator learner [13]. After a call to LearnModel, the rule sets update so that all transitions are explained by the updated rules. For each nontrivial rule, we derive a deterministic operator by extracting the rule's preconditions and its most likely effects. This set of operators collectively form our learned model.

## 5.4 How to Select Curricula and Demonstrations

For Baking-Large, we defined a set of planning problems to help the teacher benchmark the agent. The tasks are arranged in order of increasing difficulty that roughly corresponds to increasing plan lengths. The more challenging tasks depend on the successful completion of simpler tasks or are composed of multiple simpler tasks.

The teacher's process for selecting curricula and demonstrations involved the following initial steps:

- 1. Model Evaluation: Test the agent on the benchmark, progressing from the easiest to the hardest tasks. A task is solved if executing the plan open-loop reaches the goal. Examine the easiest failed task. There are two things that may have happened. (1) If no plans were found, identify the demonstrations or curricula needed by noting the easier tasks that were solved and/or planning to subsets of the goal state. (2) If a plan was executed, we identify the first failed operator in the plan and design a curriculum to correct that operator's preconditions. The next steps discuss case (2). We discuss case (1) in Section 5.4.2.
- 2. **Operator Analysis:** Examine the learned operator associated with the failed action in the plan. Inspect the operator's incorrect preconditions and devise a curriculum that elicits a plan execution error using the operator.
- 3. What to do if the curriculum doesn't help improve the model: If the curriculum doesn't induce plan execution errors because no plans are found to a subgoal, then provide demonstrations to the subgoal. If a plan execution error does not cause a model update, we found that providing additional demonstrations or duplicating past demonstrations sometimes helped update the model. However, in general, this algorithm rests on the assumption that a plan execution error causes the operator learner to update the model. We found that this assumption held for most cases with our operator learner in Baking-Large; we leave investigating this issue for future work.<sup>3</sup>

We continue the discussion by illustrating two examples in Baking-Large: one on curriculum selection and the other on choosing demonstrations.

### 5.4.1 Selecting curriculum to correct weak preconditions

If an operator failed in a plan during the teacher's evaluation, the reason is that the preconditions were too weak. To correct the preconditions, the teacher creates a curriculum where

<sup>&</sup>lt;sup>3</sup>In our experiments, to learn simple operators, if a plan execution error didn't result in a model update, we reran the experiment after tuning hyperparameters in the operator learning algorithm. However, even after tuning, our operator learner did not learn negated literals in the preconditions for our most complex operators in Baking-Large (the operators in Appendix A.4 that are named "use-stand-mixer-in-bowl...") from a single plan execution error. Future work could focus on improving the operator learner to update upon plan execution error caused by missing negated literals in the preconditions for these operators.

the preconditions are met but the executing the operator results in unexpected effects.

#### Example: learning the *put-pan-in-oven* operator in Baking-Large

In this example, here is the learned operator that the teacher inspected:

```
(:action put-pan-in-oven0
:parameters (?x0 - container ?x1 - oven)
:precondition (and (put-pan-in-oven ?x0 ?x1)
    (is-pan ?x0)
    (not (oven-is-full ?x1)))
:effect (and
    (container-in-an-oven ?x0)
    (container-in-oven ?x0 ?x1)
    (oven-is-full ?x1))
)
```

The teacher identified that this operator fails in a plan during the benchmarking. The teacher thinks about how the learned precondition could be weaker than the ground truth, and notices that the agent hasn't realized that a pan has to be outside of ovens to be put into an oven. To facilitate learning the correct operator, the teacher devises the following curriculum:

- 1. Subgoal 1: (container-in-oven pan-0 oven-0)
- 2. Subgoal 2: (container-in-oven pan-0 oven-1)

After achieving the first subgoal, the agent immediately attempts to execute *put-pan-in-oven(pan-0, oven-1)* to achieve the second subgoal. However, this action fails. The learned precondition is missing the condition (not (container-in-an-oven ?pan)). By adding this failure to the dataset and invoking LearnModel(transition\_data), the agent updates the model and learns the correct preconditions.

Here is the operator that is now learned, which matches the ground truth:

### 5.4.2 Selecting demonstrations

When executing a plan to the next subgoal results in a failure that doesn't improve the model, or no plans to the next subgoal are found, the teacher can provide a plan to the next subgoal.

#### Example: learning the operator to place desserts on plates

In this Baking-Large example, we walk through learning the operators associated with the action *move-baked-good-in-container-to-different-container* and motivate the teacher's demonstration.

At the initialization stage in Section 5.2, the agent was given one demonstration for the action *move-baked-good-in-container-to-different-container*, where a cake has been placed on a plate. From this demonstration, the agent learned this operator:

The agent plans using the automatic strategy and improves the operator's precondition:

)

The teacher would like the agent to learn to put souffles on plates and provides the following goal:

```
and((is-souffle dessert-0), (dessert-in-container plate-0 dessert-0))
```

However, no plans are found to this subgoal, so the agent requests the teacher for guidance. Following the guidelines in Section 5.4, the teacher finds that the agent is able to plan and achieve a subset of the goal state: *(is-souffle dessert-0)*. The teacher realizes that the agent is only missing the step to place the souffle on the plate, and figures out that the operator describing this action is missing. The agent has learned how to place a cake on the plate but not how to place a souffle on a plate.

To learn the correct operators, the agent needs to observe a transition from placing a souffle on a plate. Instead of providing a plan to bake a souffle, to reduce the teacher's effort, the teacher specifies the subgoal for the agent to plan to make a souffle: *(is-souffle dessert-0)*. After the agent achieves this subgoal, the teacher provides the single demonstration of moving the souffle from the pan to the plate by executing action *move-baked-good-in-container-to-different-container(pan-0 plate-0 dessert-0)*. After a model update, the agent learns a general operator:

Afterwards, our automatic strategy refines the learned operator to more closely align with the ground truth so that it can now be used to achieve the original goal. Here is the final learned operator:

```
(:action move-baked-good-in-container-to-different-container0
:parameters (?x0 - container ?x1 - container ?x2 - dessert_hypothetical)
:precondition (and (not (container-in-an-oven ?x0))
    (dessert-in-container ?x0 ?x2)
    (is-plate ?x1)
```

```
(move-baked-good-in-container-to-different-container ?x0 ?x1 ?x2))
:effect (and
   (not (dessert-in-container ?x0 ?x2))
   (not (pan-is-full ?x0))
   (dessert-in-container ?x1 ?x2))
)
```

# 5.5 Summary

In this section, we introduced our exploration algorithm for large-scale relational domains, which alternates between automatic planning and teacher guidance. In the next section, we show that this approach enables generalization and achieves greater data efficiency compared to existing exploration methods in Baking-Large.

# Chapter 6

# Results

Our results demonstrate that our exploration method consistently solves long-horizon test tasks (minimum plan length: 24 actions) in Baking-Large, whereas both GLIB and random action selection fail on these tasks. Moreover, GLIB and random actions only succeed on simple tasks, which our method solves with significantly fewer environment interactions.

In our experiments, we used the same operator learning algorithm [13] across all exploration strategies. Each experiment was capped at 2,000 environment interactions.

## 6.1 Exploration Methods Evaluated

- **GLIB-G1**: A GLIB variant that samples grounded goals (indicated by "G" in "G1") consisting of a single ground literal (the "1" in "G1"). Following the original implementation, goal novelty for GLIB-G1 is defined per episode: grounded literals visited in previous episodes can be sampled again after episode resets.
- GLIB-L2: A GLIB variant that samples lifted goals (indicated by "L" in "L2") that are conjunctions of two lifted literals (the "2" in "L2"). Following the original implementation, goal novelty for GLIB-L2 is defined across episodes: lifted goals achieved in earlier episodes cannot be sampled again in subsequent ones.
- Random actions: This strategy uniformly samples actions from the action space (the set of all ground actions).

• Our method: Employs the exploration strategy described in the previous chapter (Algorithm 1).

# 6.2 Results on Long-Horizon Test Planning Problems

At test time, we used FastDownward to generate plans and evaluated success by executing the plans open-loop to see if they achieved the goals. The evaluation consisted of two longhorizon planning problems with goals unreachable during training:

- Task 1: Bake two souffles using raw ingredients and serve them on plates. The minimum solution requires 28 actions.
- 2. Task 2: Bake two cakes using raw ingredients and serve them on plates. The minimum solution requires 24 actions.

Figure 6.1 shows the success rates for these tasks across five seeds of our method and ten seeds for each baseline method. While none of the baseline methods solve the test tasks within the 2,000 actions limit, our method succeeds in solving both tasks using fewer than 834 interactions.

### 6.2.1 Analysis of Baseline Method Failures

Baseline methods fail on the test tasks because their learned models lack the critical operators required for generating valid plans. The successful executions of these operators are absent from the data because achieving their preconditions requires long-horizon plans that are unlikely under GLIB or random action selection. Random sampling of goals and actions lacks the directed exploration needed to meet those operators' preconditions.



Figure 6.1: Success rates on unseen long-horizon planning problems as training progresses. None of the baseline methods—GLIB-G1, GLIB-L2, or random actions—solve any test tasks within 2,000 interactions. Our method successfully solves all test tasks using fewer than 834 interactions.

## 6.3 Data Efficiency

Although GLIB and random action selection fail on long-horizon tasks, they can execute some actions successfully during training. To compare data efficiency, we evaluated performance on these tasks, which require plans of three or fewer actions. As shown in Figure 6.2, our method achieves higher success rates with fewer environment interactions.



Figure 6.2: Success rates on 12 short-horizon tasks (solvable in three or fewer actions) with goals reachable in the training episodes. Our method demonstrates superior data efficiency compared to both GLIB variants and random action selection.

# Chapter 7

# **Future Work**

Our method relies on a teacher to provide curricula and demonstrations. Automating the teacher's role could be a desirable enhancement. As a potential solution, we explored the use of Large Language Models (LLMs), such as GPT-4 [28], which has commonsense knowledge abilities that could be harnessed for general knowledge-based domains like Baking-Large.

We conducted initial experiments to investigate GPT-4's capabilities in acquiring demonstrations within relational domains. Specifically, we tested two approaches: (1) integrating GPT-4-generated operators into the GLIB framework, and (2) performing open-loop planning directly with GPT-4.

## 7.1 Integrating GLIB With Operators From GPT-4

We prompted GPT-4 to generate initial operators for planning to collect demonstrations. The prompt templates are listed in Appendix B. We use GLIB to plan with the LLMproposed operators and the learned operators, while we evaluate the learned operators only. Obtaining effective operators from the LLM allows the agent to plan and observe new sets of effects early during training. These transitions, similar to the initial demonstrations in our algorithm, accelerate the learning process. However, the operators from the LLM can also be distracting and waste interactions.

The LLM-generated operators were beneficial for some domains. For popular domains that the LLM may have encountered during its training (e.g., Minecraft and Blocks), the LLM generates many operators that are accurate for the domain (Figures 7.1a and 7.1b). For the domain (Baking-Small: Figure 7.1c) that was unfamiliar to the LLM but have predicates with semantically meaningful names, the LLM also proposed some correct operators, but it proposed more incorrect operators. During training, the agent discards an LLM operator once the agent observes a transition that contradicts the operator.

The LLM-generated operators were more distracting than helpful for other domains. When the LLM generates operators that are mostly or all incorrect, the generated operators can still be used for planning, but the plans all fail and provide no informative feedback. This wastes execution time that could have been spent using alternative exploration methods. For instance, in the Doors domain (Figure 7.1d), the LLM-proposed operators were so ineffective that vanilla GLIB performed better than GLIB with the LLM operators.



(a) Success rate vs. number of environment interactions in the Minecraft domain. Comparison of two methods: GLIB + LLMinitialized operators (blue) and GLIB (orange).



(c) Success rate vs. number of environment interactions in the Baking-Small domain. Comparison of two methods: GLIB + LLM-initialized operators (blue) and GLIB (orange).



(b) Success rate vs. number of environment interactions in the Blocks domain. Comparison of two methods: GLIB + LLM-initialized operators (blue) and GLIB (orange).



(d) Success rate vs. number of environment interactions in the Doors domain. Comparison of two methods: GLIB + LLM-initialized operators (blue) and GLIB (orange).

Figure 7.1: Success rate curves comparing vanilla GLIB to GLIB initialized with LLMgenerated operators. In domains where the LLM generates almost all ground truth operators accurately (Minecraft and Blocks), learning is significantly accelerated. However, in the Baking and Doors domains, the LLM generates fewer correct operators or none at all, leading to fewer useful transitions collected (Baking) or even distraction from better action selection (Doors).

# 7.2 Planning Open-Loop with GPT-4

Given the mixed success of using GPT-4 on predicates in toy domains in Figure 7.1, we designed Baking-Large with descriptive predicate names and augmented each predicate with a natural language description. Instead of prompting with predicates as in the previous experiments, we prompted GPT-4 with natural language descriptions to propose plans. The descriptions, predicates, and prompt templates are listed in Appendix B.

GPT-4 demonstrated promise in generating the correct sequence of action names (Figure 7.2). However, it struggled to reliably produce grounded open-loop plans for our training tasks. Errors emerged when grounding the actions due to challenges in reasoning about long-range dependencies within the plans. For instance, the bowl used by the agent to add ingredients should be the same bowl used later for mixing, but GPT-4 frequently failed to maintain this consistency. Here is the sequence of atomic actions that will achieve the goal of making a cake in pan-0 while ensuring it is not damaged:

#### Goal:

Make dessert-0 a real, physical cake and ensure pan-0 is not damaged.

#### Plan:

- 1. preheat-oven-with-cake-settings
  - Preheat the oven to 350°F, which is the appropriate temperature for baking a cake.
- 2. crack-egg-and-put-in-container (using egg-0 and putting it in bowl-0)
  - · Crack the first egg into bowl-0, preparing the egg for the mixture.
- 3. crack-egg-and-put-in-container (using egg-1 and putting it in bowl-0)
  - · Crack the second egg into the same bowl, combining the eggs for the cake mixture.
- 4. pour-powdery-ingredient-from-measuring-cup (using flour-1 and putting it in bowl-0)
  - Pour all of the couple of cups of flour (flour-1) from measuring-cup-1 into bowl-0.
- 5. pour-powdery-ingredient-from-measuring-cup (using sugar-0 and putting it in bowl-0)
  - Pour the sugar (sugar-0) from measuring-cup-3 into bowl-0 to add sweetness to the cake mixture.
- pour-powdery-ingredient-from-measuring-cup (using baking-powder-0 and putting it in bowl-0)
  - Pour the baking powder (baking-powder-0) from measuring-cup-2 into bowl-0 to help the cake rise during baking.
- 7. put-butter-in-container-from-measuring-cup (using butter-0 and putting it in bowl-0)
  - Add all of the butter (butter-0) from measuring-cup-4 into bowl-0, providing richness and moisture to the cake mixture.
- 8. use-stand-mixer (using bowl-0)
  - Use the electric stand mixer to mix all of the ingredients in bowl-0, creating a smooth, consistent cake batter.
- 9. pour-mixture-only (pour the mixture from bowl-0 into pan-0)
  - · Pour the smooth cake batter from bowl-0 into pan-0, preparing it for baking.
- 10. put-pan-in-oven (using pan-0)
  - · Place pan-0 with the cake batter into the preheated oven.
- 11. set-oven-with-cake-bake-time-and-press-start
  - Set the appropriate bake time for the cake and press 'start' to begin the baking process.
- 12. remove-pan-from-oven (once the cake is baked)
  - After the baking time has completed, remove pan-0 from the oven and place it on the counter to cool, ensuring it is undamaged.

This sequence will result in dessert-0 becoming a real, physical cake while pan-0 remains	
undamaged.	

Figure 7.2: ChatGPT's proposed action sequence to bake a cake in Baking-Large. In this example shown, the proposed sequence of action predicate names can form a valid plan to solve the goal. However, the groundings of the predicates in the later responses (not shown) make the plan fail.

## 7.3 Takeaways and Future Work

Our findings highlight the following insights and opinions:

- Natural Language Descriptions Are Essential: To better leverage GPT-4 for generating plans and operators, predicates should be elaborated on by natural language descriptions. GPT-4 performs better at proposing sequences of action names than at directly reasoning about predicates.
- 2. **Operators vs. Demonstrations**: Proposing effective operators for exploration has a similar benefit to planning directly with the LLM, as these operators enable the observation of new sets of effects that improve learning.
- 3. Challenges in Grounded Plans: GPT-4 struggles to generate open-loop plans for exploration tasks as the horizon increases. While the sequence of action predicate names often forms a reasonable plan, errors arise when grounding the plan, particularly due to long-range dependencies.
- 4. Brittleness of Handcrafted Prompts: A key challenge we observed is the sensitivity of GPT-4's outputs to variations in prompt templates or inputs. Developing systematic approaches for prompt creation and optimization could greatly enhance the robustness and effectiveness of prompting programs.

Future research could continue this line of work of using LLMs to acquire demonstrations that introduce new effects while leveraging planning to refine the preconditions.

# Chapter 8

# Conclusion

When action executions are costly, learning a model for planning to goals must minimize the number of actions taken. With this in mind, we proposed an algorithm for efficient exploration in relational model-based reinforcement learning. Drawing on insights from goalliteral babbling (GLIB), our algorithm focuses on correcting preconditions through planning and using demonstrations from a teacher to expedite learning. We evaluated our approach in a large-scale domain, Baking-Large, where our algorithm is able to learn models for longhorizon planning and achieve greater data efficiency than GLIB.

Additionally, we conducted an initial investigation into the role of Large Language Models (LLMs) in exploration within relational domains. Our experiments indicate that GPT-4 can be beneficial by proposing operators in several domains, but GPT-4's proposed operators can be distracting for other domains. GPT-4 also demonstrated potential as a heuristic for planning in the Baking-Large domain, but it made mistakes when grounding the plans.

Future work could extend our algorithm by developing automated guidance for the teacher, potentially by leveraging LLMs. A potential path could use LLMs that accept predicates annotated with natural language descriptions to propose plans closed-loop for demonstrations. Such systems could address potential brittleness from hand-crafted inputs and include systematic procedures for prompt and label generation to ensure robustness and scalability. A careful characterization of the challenges posed by large-scale, complex relational domains and the appropriate selection of teacher demonstrations and subgoals to address those challenges would further advance this direction.

# Appendix A

# Domains

In this appendix, we provide the operators, predicates, and objects for each domain used in the experiments presented in this thesis. The objects vary across problems. Rather than listing the objects for each problem individually, we include the set of objects for the problem with the largest object set in the domain.

The operators' parameters and the objects are annotated with their respective types. For instance, in the Blocks domain, a line such as "a - block" indicates an object named "a" of type "block."

# A.1 Baking-Small

### A.1.1 Predicates

Action Predicates

```
(putegginpan ?egg - ingredient ?pan - pan)
(putflourinpan ?flour - ingredient ?pan - pan)
(mix ?pan - pan)
(putpaninoven ?pan - pan ?oven - oven)
(removepanfromoven ?pan - pan)
(bakecake ?new - ingredient ?oven - oven)
(bakesouffle ?new - ingredient ?oven - oven)
```

#### **State Predicates**

```
(isegg ?egg - ingredient)
      (isflour ?flour - ingredient)
      (panhasegg ?pan - pan)
      (panhasflour ?pan - pan)
      (panisclean ?pan - pan)
      (paninoven ?pan - pan)
      (inpan ?x - ingredient ?pan - pan)
      (inoven ?pan - pan ?oven - oven)
      (ovenisfull ?oven - oven)
g
      (hypothetical ?new - ingredient)
      (ismixed ?pan - pan)
      (iscake ?new - ingredient)
12
      (issouffle ?new - ingredient)
13
      (soapconsumed ?soap - soap)
```

### A.1.2 Operators

1 (:action putegginpan 2 :parameters (?egg - ingredient ?pan - pan) 3 :precondition (and (putegginpan ?egg ?pan) 4 (isegg ?egg) 5 (not (panhasegg ?pan)) 6 (not (ismixed ?pan)) 7 (panisclean ?pan) 8 (not (paninoven ?pan)) 9 ) 10 :effect (and (panhasegg ?pan) 11 (inpan ?egg ?pan)

```
)
12
      )
13
14
      (:action putflourinpan
15
      :parameters (?flour - ingredient ?pan - pan)
16
      :precondition (and (putflourinpan ?flour ?pan)
17
                         (isflour ?flour)
18
                         (not (panhasflour ?pan))
19
                         (not (ismixed ?pan))
20
                         (panisclean ?pan)
21
                          (not (paninoven ?pan))
22
                    )
23
      :effect (and (panhasflour ?pan)
24
                   (inpan ?flour ?pan)
25
              )
26
      )
27
28
      (:action mix
29
      :parameters (?egg - ingredient ?flour - ingredient ?pan - pan)
30
      :precondition (and (mix ?pan)
31
                         (inpan ?egg ?pan)
32
                         (inpan ?flour ?pan)
33
                         (isegg ?egg)
34
                         (isflour ?flour)
35
                         (not (paninoven ?pan))
36
                    )
37
      :effect (and (ismixed ?pan)
38
                   (not (isegg ?egg))
39
                   (not (isflour ?flour))
40
                   (not (inpan ?egg ?pan))
41
                   (not (inpan ?flour ?pan))
42
                   (not (panhasegg ?pan))
43
```

```
(not (panhasflour ?pan))
44
              )
45
      )
46
47
      (:action putpaninoven
48
      :parameters (?pan - pan ?oven - oven)
49
      :precondition (and (putpaninoven ?pan ?oven)
50
                         (not (ovenisfull ?oven))
51
                         (not (paninoven ?pan))
52
                    )
53
      :effect (and (ovenisfull ?oven)
54
                   (inoven ?pan ?oven)
                   (paninoven ?pan)
56
              )
57
      )
58
59
      (:action removepanfromoven
60
      :parameters (?pan - pan ?oven - oven)
61
      :precondition (and (removepanfromoven ?pan)
62
                         (inoven ?pan ?oven)
63
                    )
64
      :effect (and (not (ovenisfull ?oven))
65
                   (not (inoven ?pan ?oven))
66
                   (not (paninoven ?pan))
67
              )
68
      )
70
      (:action bakecake
71
      :parameters (?oven - oven ?pan - pan ?new - ingredient)
72
      :precondition (and (bakecake ?new ?oven)
73
                         (ismixed ?pan)
74
                         (inoven ?pan ?oven)
75
```

```
(hypothetical ?new)
76
                     )
77
       :effect (and (not (ismixed ?pan))
78
                    (not (panisclean ?pan))
79
                    (not (hypothetical ?new))
80
                    (iscake ?new)
81
               )
82
      )
83
84
       (:action bakesouffle
85
       :parameters (?oven - oven ?egg - ingredient ?pan - pan ?new - ingredient)
86
       :precondition (and (bakesouffle ?new ?oven)
87
                          (inpan ?egg ?pan)
88
                          (isegg ?egg)
89
                          (not (panhasflour ?pan))
90
                          (inoven ?pan ?oven)
91
                          (hypothetical ?new)
92
                     )
93
       :effect (and (not (isegg ?egg))
94
                    (not (inpan ?egg ?pan))
95
                    (not (panhasegg ?pan))
96
                    (not (panisclean ?pan))
97
                    (not (hypothetical ?new))
98
                    (issouffle ?new)
99
               )
100
      )
       (:action cleanpan
103
       :parameters (?pan - pan ?soap - soap)
       :precondition (and (cleanpan ?pan ?soap)
                          (not (soapconsumed ?soap))
106
                          (not (paninoven ?pan))
107
```

```
108 )
109 :effect (and (panisclean ?pan)
110 (soapconsumed ?soap)
111 )
112 )
```

## A.1.3 Largest Set of Objects in a Problem

```
oven-0 - oven
       oven-1 - oven
       egg-0 - ingredient
       egg-1 - ingredient
       flour-0 - ingredient
5
       flour-1 - ingredient
6
       pan-0 - pan
7
       pan-1 - pan
       new-0 - ingredient
9
       new-1 - ingredient
10
       soap-0 - soap
11
```

# A.2 Blocks

## A.2.1 Predicates

**Action Predicates** 

```
(pickup ?x - block)
(putdown ?x - block)
(stack ?x - block ?y - block)
(unstack ?x - block)
```

```
1 (on ?x - block ?y - block)
2 (ontable ?x - block)
3 (clear ?x - block)
4 (handempty ?x - robot)
5 (handfull ?x - robot)
6 (holding ?x - block)
```

### A.2.2 Operators

```
(:action pick-up
          :parameters (?x - block ?robot - robot)
2
          :precondition (and
              (pickup ?x)
              (clear ?x)
              (ontable ?x)
              (handempty ?robot)
          )
          :effect (and
9
              (not (ontable ?x))
10
              (not (clear ?x))
              (not (handempty ?robot))
12
              (handfull ?robot)
13
              (holding ?x)
14
          )
15
      )
16
17
      (:action put-down
18
          :parameters (?x - block ?robot - robot)
19
          :precondition (and
20
              (putdown ?x)
21
```

```
(holding ?x)
22
               (handfull ?robot)
23
          )
24
          :effect (and
25
               (not (holding ?x))
26
               (clear ?x)
27
               (handempty ?robot)
28
               (not (handfull ?robot))
29
               (ontable ?x))
30
          )
31
32
       (:action stack
33
           :parameters (?x - block ?y - block ?robot - robot)
34
           :precondition (and
35
               (stack ?x ?y)
36
               (holding ?x)
37
               (clear ?y)
38
               (handfull ?robot)
39
          )
40
          :effect (and
41
               (not (holding ?x))
42
               (not (clear ?y))
43
               (clear ?x)
44
               (handempty ?robot)
45
               (not (handfull ?robot))
46
               (on ?x ?y)
47
          )
48
      )
49
50
      (:action unstack
51
           :parameters (?x - block ?y - block ?robot - robot)
52
          :precondition (and
53
```

54	(unstack ?x)
55	(on ?x ?y)
56	(clear ?x)
57	(handempty ?robot)
58	)
59	:effect (and
60	(holding ?x)
61	(clear ?y)
62	(not (clear ?x))
63	(not (handempty ?robot))
64	(handfull ?robot)
65	(not (on ?x ?y))
66	)
67	

# A.2.3 Largest Set of Objects in a Problem

1	d – block
2	b – block
3	a – block
4	c – block
5	e – block
6	f – block
7	robot - robot

# A.3 Doors

## A.3.1 Predicates

Action Predicates

```
(moveto ?loc - location)
(pick ?key - key)
```

#### **State Predicates**

(at ?loc - location)
(unlocked ?room - room)
(locinroom ?loc - location ?room - room)
(keyat ?key - key ?loc - location)
(keyforroom ?key - key ?room - room)

## A.3.2 Operators

```
(:action moveto
2
      :parameters (?sloc - location ?eloc - location ?eroom - room)
3
      :precondition (and (moveto ?eloc)
                        (at ?sloc)
                        (unlocked ?eroom)
                        (locinroom ?eloc ?eroom)
                   )
8
      :effect (and (not (at ?sloc))
9
                   (at ?eloc)
10
             )
    )
12
13
    (:action pick
14
      :parameters (?loc - location ?key - key ?room - room)
      :precondition (and (pick ?key)
16
                        (at ?loc)
17
                        (keyat ?key ?loc)
18
```

```
19 (keyforroom ?key ?room)
20 )
21 :effect (and (not (keyat ?key ?loc))
22 (unlocked ?room)
23 )
24 )
```

## A.3.3 Largest Set of Objects in a Problem

```
key-0 - key
    key-1 - key
2
    key-2 - key
3
    key-3 - key
4
          key-4 - key
5
         key-5 - key
6
    loc-0-0 - location
7
    loc-0-1 - location
8
    loc-0-2 - location
9
    loc-0-3 - location
    loc-0-4 - location
11
    loc-0-5 - location
    loc-0-6 - location
13
    loc-0-7 - location
14
    loc-0-8 - location
15
    loc-0-9 - location
16
    loc-1-0 - location
17
    loc-1-1 - location
18
    loc-1-2 - location
19
    loc-1-3 - location
20
    loc-1-4 - location
21
    loc-1-5 - location
22
    loc-1-6 - location
23
```

24	loc-1-7	-	location
25	loc-1-8	-	location
26	loc-1-9	-	location
27	loc-2-0	-	location
28	loc-2-1	-	location
29	loc-2-2	-	location
30	loc-2-3	-	location
31	loc-2-4	-	location
32	loc-2-5	-	location
33	loc-2-6	-	location
34	loc-2-7	-	location
35	loc-2-8	-	location
36	loc-2-9	-	location
37	loc-3-0	-	location
38	loc-3-1	-	location
39	loc-3-2	-	location
40	loc-3-3	-	location
41	loc-3-4	-	location
42	loc-3-5	-	location
43	loc-3-6	-	location
44	loc-3-7	-	location
45	loc-3-8	-	location
46	loc-3-9	-	location
47	loc-4-0	-	location
48	loc-4-1	-	location
49	loc-4-2	-	location
50	loc-4-3	-	location
51	loc-4-4	-	location
52	loc-4-5	-	location
53	loc-4-6	-	location
54	loc-4-7	-	location
55	loc-4-8	-	location

56	loc-4-9 - location
57	loc-5-0 - location
58	loc-5-1 - location
59	loc-5-2 - location
60	loc-5-3 - location
61	loc-5-4 - location
62	loc-5-5 - location
63	loc-5-6 - location
64	loc-5-7 - location
65	loc-5-8 - location
66	loc-5-9 - location
67	loc-6-0 - location
68	loc-6-1 - location
69	loc-6-2 - location
70	loc-6-3 - location
71	loc-6-4 - location
72	loc-6-5 - location
73	loc-6-6 - location
74	loc-6-7 - location
75	loc-6-8 - location
76	loc-6-9 - location
77	loc-7-0 - location
78	loc-7-1 - location
79	loc-7-2 - location
80	loc-7-3 - location
81	loc-7-4 - location
82	loc-7-5 - location
83	loc-7-6 - location
84	loc-7-7 - location
85	loc-7-8 - location
86	loc-7-9 - location
87	room-0 - room

I.

 88
 room-1
 room

 89
 room-2
 room

 90
 room-3
 room

 91
 room-4
 room

 92
 room-5
 room

 93
 room-6
 room

# A.4 Minecraft

### A.4.1 Predicates

**Action Predicates** 

State Predicates

```
(isgrass ?arg0 - moveable)
(islog ?arg0 - moveable)
(isplanks ?arg0 - moveable)
(at ?arg0 - moveable ?arg1 - static)
(agentat ?arg0 - static)
(inventory ?arg0 - moveable)
(hypothetical ?arg0 - moveable)
(equipped ?arg0 - moveable ?arg1 - agent)
(handsfree ?arg0 - agent)
```

## A.4.2 Operators

```
(:action recall
  :parameters (?var0 - moveable ?var1 - agent)
  :precondition (and
```

```
(recall ?var0)
        (equipped ?var0 ?var1)
5
      )
      :effect (and
7
        (inventory ?var0)
        (not (equipped ?var0 ?var1))
              (handsfree ?var1)
10
      )
11
    )
12
13
    (:action move
14
      :parameters (?var0 - static ?var1 - static)
15
      :precondition (and
16
              (move ?var0)
17
        (agentat ?var1)
18
      )
19
      :effect (and
20
        (agentat ?var0)
21
        (not (agentat ?var1))
22
      )
23
    )
24
25
    (:action craftplank
26
      :parameters (?var0 - moveable ?var1 - agent ?var2 - moveable)
27
      :precondition (and
28
              (craftplank ?var0 ?var2)
29
        (hypothetical ?var0)
30
        (islog ?var2)
31
        (equipped ?var2 ?var1)
      )
33
      :effect (and
34
        (inventory ?var0)
35
```

```
(isplanks ?var0)
36
              (handsfree ?var1)
37
        (not (equipped ?var2 ?var1))
38
        (not (hypothetical ?var0))
39
        (not (islog ?var2))
40
      )
41
    )
42
43
    (:action equip
44
      :parameters (?var0 - moveable ?var1 - agent)
45
      :precondition (and
46
              (equip ?var0)
47
        (inventory ?var0)
48
        (handsfree ?var1)
49
      )
50
      :effect (and
51
        (equipped ?var0 ?var1)
              (not (handsfree ?var1))
53
        (not (inventory ?var0))
54
      )
    )
56
57
    (:action pick
58
      :parameters (?var0 - moveable ?var1 - static)
59
      :precondition (and
60
              (pick ?var0)
61
        (at ?var0 ?var1)
62
        (agentat ?var1)
63
      )
64
      :effect (and
65
        (inventory ?var0)
66
        (not (at ?var0 ?var1))
67
```
)

68

# A.4.3 Largest Set of Objects in a Problem

1	log-0 -	moveable
2	log-1 -	moveable
3	log-2 -	moveable
4	log-3 -	moveable
5	grass-4	- moveable
6	grass-5	- moveable
7	new-0 -	moveable
8	new-1 -	moveable
9	new-2 -	moveable
10	agent -	agent
11	loc-0-0	- static
12	loc-0-1	- static
13	loc-0-2	- static
14	loc-0-3	- static
15	loc-0-4	- static
16	loc-1-0	- static
17	loc-1-1	- static
18	loc-1-2	- static
19	loc-1-3	- static
20	loc-1-4	- static
21	loc-2-0	- static
22	loc-2-1	- static
23	loc-2-2	- static
24	loc-2-3	- static
25	loc-2-4	- static
26	loc-3-0	- static
27	loc-3-1	- static
28	loc-3-2	- static

29	loc-3-3 - static
30	loc-3-4 - static
31	loc-4-0 - static
32	loc-4-1 - static
33	loc-4-2 - static
34	loc-4-3 - static
35	loc-4-4 - static

## A.5 Baking-Large

### A.5.1 Predicates

#### **Action Predicates**

```
(beat-egg-whites ?arg0 - electric_stand_mixer ?arg1 - container ?arg2 -
          \hookrightarrow egg_hypothetical)
      (crack-egg-and-put-in-container ?arg0 - egg_hypothetical ?arg1 - container)
      (fold-stiff-egg-whites-into-mixture ?arg0 - spatula ?arg1 - container ?arg2 -
          \hookrightarrow container ?arg3 - egg_hypothetical ?arg4 - mixture_hypothetical)
      (move-baked-good-in-container-to-different-container ?arg0 - container ?arg1 -
          \hookrightarrow container ?arg2 - dessert_hypothetical)
      (pour-mixture-only ?arg0 - container ?arg1 - container ?arg2 -
          \hookrightarrow mixture_hypothetical)
      (pour-powdery-ingredient-from-container ?arg0 - container ?arg1 - container ?
          \hookrightarrow arg2 - powder_ingredient_hypothetical)
      (pour-powdery-ingredient-from-measuring-cup ?arg0 -
          \rightarrow powder_ingredient_hypothetical ?arg1 - measuring_cup ?arg2 - container)
      (preheat-oven-with-cake-settings ?arg0 - oven)
      (preheat-oven-with-souffle-settings ?arg0 - oven)
      (put-butter-in-container-from-measuring-cup ?arg0 - butter_hypothetical ?arg1 -
10
          \hookrightarrow container)
      (put-pan-in-oven ?arg0 - container ?arg1 - oven)
```

12	(remove-pan-from-oven ?arg0 - container ?arg1 - oven)
13	(separate-raw-yolk-from-egg-whites ?arg0 - egg_hypothetical ?arg1 -
	$\hookrightarrow$ egg_hypothetical ?arg2 - container ?arg3 - container)
14	(set-oven-with-cake-bake-time-and-press-start ?arg0 - oven ?arg1 -
	$\hookrightarrow$ dessert_hypothetical ?arg2 - mixture_hypothetical)
15	(set-oven-with-souffle-bake-time-and-press-start ?arg0 - oven ?arg1 -
	$\hookrightarrow$ dessert_hypothetical ?arg2 - mixture_hypothetical)
16	(transfer-butter-from-pan-or-bowl ?arg0 - container ?arg1 - container ?arg2 -
	$\hookrightarrow$ butter_hypothetical)
17	(transfer-egg-from-pan-or-bowl ?arg0 - container ?arg1 - container ?arg2 -
	$\hookrightarrow$ egg_hypothetical)
18	(use-stand-mixer ?arg0 - electric_stand_mixer ?arg1 - container ?arg2 -
	$\hookrightarrow$ mixture_hypothetical)

## State Predicates

1	(sugar-is-consumed ?arg0 - powder_ingredient_hypothetical)
2	(baking-powder-is-consumed ?arg0 - powder_ingredient_hypothetical)
3	(tablespoons-of-flour-is-consumed ?arg0 - powder_ingredient_hypothetical)
4	(cups-of-flour-is-consumed ?arg0 - powder_ingredient_hypothetical)
5	(butter-is-consumed ?arg0 - butter_hypothetical)
6	(butter-in-container ?arg0 - container ?arg1 - butter_hypothetical)
7	(butter-in-measuring-cup ?arg0 - butter_hypothetical ?arg1 - measuring_cup)
8	(container-in-an-oven ?arg0 - container)
9	(container-in-oven ?arg0 - container ?arg1 - oven)
10	(dessert-in-container ?arg0 - container ?arg1 - dessert_hypothetical)
11	(dessert-is-hypothetical ?arg0 - dessert_hypothetical)
12	(different ?arg0 - spatula ?arg1 - spatula)
13	(egg-in-container ?arg0 - container ?arg1 - egg_hypothetical)
14	(egg-is-hypothetical ?arg0 - egg_hypothetical)
15	(is-baking-powder ?arg0 - powder_ingredient_hypothetical)
16	(is-bowl ?arg0 - container)

17	(is-butter ?arg0 - butter_hypothetical)
18	(is-cake ?arg0 - dessert_hypothetical)
19	(is-cups-of-flour ?arg0 - powder_ingredient_hypothetical)
20	(is-egg ?arg0 - egg_hypothetical)
21	(is-in-shell ?arg0 - egg_hypothetical)
22	(is-mixture ?arg0 - mixture_hypothetical)
23	(is-oven ?arg0 - oven)
24	(is-pan ?arg0 - container)
25	(is-plate ?arg0 - container)
26	(is-raw-egg-whites ?arg0 - egg_hypothetical)
27	(is-raw-egg-yolk ?arg0 - egg_hypothetical)
28	(is-souffle ?arg0 - dessert_hypothetical)
29	(is-sugar ?arg0 - powder_ingredient_hypothetical)
30	(is-tablespoons-of-flour ?arg0 - powder_ingredient_hypothetical)
31	(is-whipped-egg-whites ?arg0 - egg_hypothetical)
32	(is-whole-raw-egg ?arg0 - egg_hypothetical)
33	(mixture-has-baking-powder ?arg0 - mixture_hypothetical)
34	(mixture-has-butter ?arg0 - mixture_hypothetical)
35	(mixture-has-cups-of-flour ?arg0 - mixture_hypothetical)
36	(mixture-has-folded-raw-egg-whites ?arg0 - mixture_hypothetical)
37	(mixture-has-blended-raw-egg-whites ?arg0 - mixture_hypothetical)
38	(mixture-has-raw-egg-yolk ?arg0 - mixture_hypothetical)
39	(mixture-has-sugar ?arg0 - mixture_hypothetical)
40	(mixture-has-tablespoons-of-flour ?arg0 - mixture_hypothetical)
41	(mixture-in-container ?arg0 - container ?arg1 - mixture_hypothetical)
42	(mixture-is-airy ?arg0 - mixture_hypothetical)
43	(mixture-is-hypothetical ?arg0 - mixture_hypothetical)
44	(oven-is-full ?arg0 - oven)
45	(oven-is-heated-for-cake ?arg0 - oven)
46	(oven-is-heated-for-souffle ?arg0 - oven)
47	(pan-is-damaged ?arg0 - container)
48	(pan-is-full ?arg0 - container)

```
(powder-ingredient-in-container ?arg0 - container ?arg1 -
49
          \hookrightarrow powder_ingredient_hypothetical)
      (powder-ingredient-in-measuring-cup ?arg0 - powder_ingredient_hypothetical ?
50
          \hookrightarrow arg1 - measuring_cup)
```

## A.5.2 Operators

1	(:action bake-cake
2	:parameters (?oven - oven ?pan - container ?mixture - mixture_hypothetical ?
	$\hookrightarrow$ cake - dessert_hypothetical)
3	:precondition (and (set-oven-with-cake-bake-time-and-press-start ?oven ?cake ?
	$\hookrightarrow$ mixture)
4	(oven-is-heated-for-cake ?oven)
5	(dessert-is-hypothetical ?cake)
6	<pre>(not (mixture-is-hypothetical ?mixture))</pre>
7	(container-in-oven ?pan ?oven)
8	(not (pan-is-full ?pan))
9	(is-pan ?pan)
10	(mixture-in-container ?pan ?mixture)
11	(mixture-has-blended-raw-egg-whites ?mixture)
12	(mixture-has-raw-egg-yolk ?mixture)
13	(mixture-has-butter ?mixture)
14	(mixture-has-sugar ?mixture)
15	(mixture-has-cups-of-flour ?mixture)
16	(mixture-has-baking-powder ?mixture)
17	(not (mixture-is-airy ?mixture)))
18	:effect (and
19	(not (is-mixture ?mixture))
20	(not (mixture-has-blended-raw-egg-whites ?mixture))
21	(not (mixture-has-raw-egg-yolk ?mixture))
22	(not (mixture-has-butter ?mixture))
23	(not (mixture-has-sugar ?mixture))

24	(not (mixture-has-cups-of-flour ?mixture))
25	(not (mixture-has-baking-powder ?mixture))
26	(not (mixture-in-container ?pan ?mixture))
27	<pre>(not (dessert-is-hypothetical ?cake))</pre>
28	(is-cake ?cake)
29	(pan-is-full ?pan)
30	(dessert-in-container ?pan ?cake))
31	)
32	
33	(:action bake-souffle
34	:parameters (?oven - oven ?pan - container ?mixture - mixture_hypothetical ?
	$\hookrightarrow$ souffle - dessert_hypothetical)
35	:precondition (and (set-oven-with-souffle-bake-time-and-press-start ?oven ?
	$\hookrightarrow$ souffle ?mixture)
36	(oven-is-heated-for-souffle ?oven)
37	(dessert-is-hypothetical ?souffle)
38	(is-mixture ?mixture)
39	(is-pan ?pan)
40	(not (pan-is-full ?pan))
41	(mixture-in-container ?pan ?mixture)
42	(container-in-oven ?pan ?oven)
43	<pre>(not (mixture-is-hypothetical ?mixture))</pre>
44	(mixture-has-folded-raw-egg-whites ?mixture)
45	(mixture-has-raw-egg-yolk ?mixture)
46	(mixture-has-butter ?mixture)
47	(mixture-has-sugar ?mixture)
48	<pre>(not (mixture-has-cups-of-flour ?mixture))</pre>
49	(mixture-is-airy ?mixture))
50	:effect (and
51	(not (is-mixture ?mixture))
52	<pre>(not (mixture-has-folded-raw-egg-whites ?mixture))</pre>
53	(not (mixture-has-raw-egg-yolk ?mixture))

```
(not (mixture-has-butter ?mixture))
54
        (not (mixture-has-sugar ?mixture))
55
        (not (mixture-has-tablespoons-of-flour ?mixture))
56
        (not (mixture-is-airy ?mixture))
        (not (mixture-in-container ?pan ?mixture))
58
        (not (dessert-is-hypothetical ?souffle))
        (is-souffle ?souffle)
60
        (pan-is-full ?pan)
61
        (dessert-in-container ?pan ?souffle))
    )
63
64
    (:action beat-egg-whites
      :parameters (?e - egg_hypothetical ?m - electric_stand_mixer ?c - container)
66
      :precondition (and (beat-egg-whites ?m ?c ?e)
67
        (egg-in-container ?c ?e)
68
        (is-raw-egg-whites ?e)
69
        (is-egg ?e)
70
        (not (egg-is-hypothetical ?e)))
71
      :effect (and
72
        (is-whipped-egg-whites ?e))
73
    )
74
75
    (:action crack-egg-in-container
76
      :parameters (?egg - egg_hypothetical ?c - container)
77
      :precondition (and (crack-egg-and-put-in-container ?egg ?c)
78
        (not (egg-in-container ?c ?egg))
79
        (is-egg ?egg)
80
        (is-in-shell ?egg)
81
        (not (container-in-an-oven ?c)))
82
      :effect (and
83
        (egg-in-container ?c ?egg)
84
        (is-whole-raw-egg ?egg)
85
```

```
(not (is-in-shell ?egg)))
86
    )
87
88
     (:action fold-egg-whites-into-mixture
89
       :parameters (?e - egg_hypothetical ?m - mixture_hypothetical ?from - container
90
          \hookrightarrow ?to - container ?s - spatula)
       :precondition (and (fold-stiff-egg-whites-into-mixture ?s ?from ?to ?e ?m)
91
         (mixture-in-container ?to ?m)
92
         (egg-in-container ?from ?e)
95
         (is-whipped-egg-whites ?e)
94
         (not (egg-is-hypothetical ?e))
95
         (is-mixture ?m)
96
         (not (mixture-is-hypothetical ?m))
97
         (not (container-in-an-oven ?from))
98
         (not (container-in-an-oven ?to)))
90
       :effect (and
100
         (mixture-is-airy ?m)
         (not (is-whipped-egg-whites ?e))
         (not (is-egg ?e))
         (mixture-has-folded-raw-egg-whites ?m)
         (not (egg-in-container ?from ?e)))
105
    )
106
     (:action place-cake-on-plate
108
       :parameters (?from - container ?to - container ?cake - dessert_hypothetical)
109
       :precondition (and (move-baked-good-in-container-to-different-container ?from ?
          \hookrightarrow to ?cake)
         (dessert-in-container ?from ?cake)
         (not (dessert-is-hypothetical ?cake))
         (is-cake ?cake)
         (is-plate ?to)
114
         (not (container-in-an-oven ?to))
```

```
(not (container-in-an-oven ?from)))
116
       :effect (and
         (dessert-in-container ?to ?cake)
118
         (not (pan-is-full ?from))
         (not (dessert-in-container ?from ?cake)))
120
    )
     (:action place-souffle-on-plate
123
       :parameters (?from - container ?to - container ?souffle - dessert_hypothetical)
       :precondition (and (dessert-in-container ?from ?souffle)
125
         (move-baked-good-in-container-to-different-container ?from ?to ?souffle)
126
        (is-souffle ?souffle)
         (not (dessert-is-hypothetical ?souffle))
128
        (is-plate ?to)
129
        (not (container-in-an-oven ?to))
130
         (not (container-in-an-oven ?from)))
131
       :effect (and
         (dessert-in-container ?to ?souffle)
         (not (pan-is-full ?from))
134
         (not (dessert-in-container ?from ?souffle)))
135
    )
136
137
     (:action preheat-for-cake
138
       :parameters (?o - oven)
139
       :precondition (and (preheat-oven-with-cake-settings ?o)
140
         (not (oven-is-heated-for-cake ?o)))
141
       :effect (and
142
         (oven-is-heated-for-cake ?o)
143
         (not (oven-is-heated-for-souffle ?o)))
144
    )
145
146
     (:action preheat-for-souffle
147
```

```
:parameters (?o - oven)
148
       :precondition (and (preheat-oven-with-souffle-settings ?o)
149
         (not (oven-is-heated-for-souffle ?o)))
150
       :effect (and
         (not (oven-is-heated-for-cake ?o))
         (oven-is-heated-for-souffle ?o))
    )
     (:action put-butter-in-bowl
156
       :parameters (?butter - butter_hypothetical ?bowl - container ?cup -
157
          \hookrightarrow measuring_cup)
       :precondition (and (put-butter-in-container-from-measuring-cup ?butter ?bowl)
158
         (is-butter ?butter)
         (not (butter-in-container ?bowl ?butter))
160
         (not (container-in-an-oven ?bowl))
161
         (butter-in-measuring-cup ?butter ?cup))
162
       :effect (and
163
         (butter-in-container ?bowl ?butter)
164
         (not (butter-in-measuring-cup ?butter ?cup)))
165
    )
166
167
     (:action put-butter-in-pan
168
       :parameters (?butter - butter_hypothetical ?pan - container ?cup -
169
          \hookrightarrow measuring_cup)
       :precondition (and (put-butter-in-container-from-measuring-cup ?butter ?pan)
         (not (butter-in-container ?pan ?butter))
17
         (is-butter ?butter)
172
         (is-pan ?pan)
173
         (not (container-in-an-oven ?pan))
174
         (butter-in-measuring-cup ?butter ?cup))
175
       :effect (and
176
         (butter-in-container ?pan ?butter)
```

```
(not (butter-in-measuring-cup ?butter ?cup)))
178
     )
179
180
     (:action put-pan-in-oven
181
       :parameters (?pan - container ?oven - oven)
182
       :precondition (and (put-pan-in-oven ?pan ?oven)
183
         (is-pan ?pan)
184
         (not (container-in-an-oven ?pan))
185
         (not (oven-is-full ?oven)))
186
       :effect (and
187
         (container-in-oven ?pan ?oven)
188
         (container-in-an-oven ?pan)
189
         (oven-is-full ?oven))
190
     )
191
192
     (:action put-powdery-ingredient-in-container
193
       :parameters (?p - powder_ingredient_hypothetical ?c - container ?cup -
194
           \hookrightarrow measuring_cup)
       :precondition (and (pour-powdery-ingredient-from-measuring-cup ?p ?cup ?c)
195
         (not (powder-ingredient-in-container ?c ?p))
196
         (not (container-in-an-oven ?c))
197
         (powder-ingredient-in-measuring-cup ?p ?cup))
198
       :effect (and
199
         (powder-ingredient-in-container ?c ?p)
200
         (not (powder-ingredient-in-measuring-cup ?p ?cup)))
201
     )
202
203
     (:action put-powdery-ingredient-into-container
204
       :parameters (?s - powder_ingredient_hypothetical ?p - container ?cup -
205
           \hookrightarrow measuring_cup)
       :precondition (and (pour-powdery-ingredient-from-measuring-cup ?s ?cup ?p)
206
         (not (powder-ingredient-in-container ?p ?s))
207
```

```
(not (container-in-an-oven ?p))
208
         (powder-ingredient-in-measuring-cup ?s ?cup))
200
       :effect (and
210
         (powder-ingredient-in-container ?p ?s)
211
         (not (powder-ingredient-in-measuring-cup ?s ?cup)))
212
     )
213
     (:action remove-pan-from-oven
215
       :parameters (?pan - container ?oven - oven)
216
       :precondition (and (remove-pan-from-oven ?pan ?oven)
217
         (container-in-oven ?pan ?oven)
218
         (is-pan ?pan))
210
       :effect (and
220
         (not (oven-is-full ?oven))
221
         (not (container-in-oven ?pan ?oven))
222
         (not (container-in-an-oven ?pan))
223
         (not (oven-is-heated-for-cake ?oven))
224
         (not (oven-is-heated-for-souffle ?oven)))
225
     )
226
227
     (:action separate-egg
228
       :parameters (?x - egg_hypothetical ?y - egg_hypothetical ?eggyolkcontainer -
220
           \hookrightarrow container ?eggwhitescontainer - container)
       :precondition (and (separate-raw-yolk-from-egg-whites ?x ?y ?eggyolkcontainer ?
230
           \hookrightarrow eggwhitescontainer)
         (is-egg ?x)
231
         (is-whole-raw-egg ?x)
232
         (not (is-in-shell ?x))
233
         (not (is-egg ?y))
234
         (egg-is-hypothetical ?y)
235
         (egg-in-container ?eggyolkcontainer ?x))
236
       :effect (and
237
```

```
(not (egg-is-hypothetical ?y))
238
         (not (is-whole-raw-egg ?x))
         (is-egg ?y)
240
         (is-raw-egg-yolk ?x)
241
         (is-raw-egg-whites ?y)
242
         (not (is-in-shell ?y))
243
         (not (egg-in-container ?eggyolkcontainer ?y))
244
         (egg-in-container ?eggwhitescontainer ?y))
245
     )
246
247
     (:action transfer-butter
248
       :parameters (?from - container ?to - container ?butter - butter_hypothetical)
240
       :precondition (and (transfer-butter-from-pan-or-bowl ?from ?to ?butter)
250
         (butter-in-container ?from ?butter)
251
         (is-butter ?butter)
252
         (not (container-in-an-oven ?from))
253
         (not (container-in-an-oven ?to)))
254
       :effect (and
255
         (not (butter-in-container ?from ?butter))
256
         (butter-in-container ?to ?butter))
257
     )
258
259
     (:action transfer-egg
260
       :parameters (?from - container ?to - container ?egg - egg_hypothetical)
261
       :precondition (and (transfer-egg-from-pan-or-bowl ?from ?to ?egg)
262
         (egg-in-container ?from ?egg)
263
         (not (egg-is-hypothetical ?egg))
264
         (is-egg ?egg)
265
         (not (container-in-an-oven ?from))
266
         (not (container-in-an-oven ?to)))
267
       :effect (and
268
         (not (egg-in-container ?from ?egg))
269
```

```
(egg-in-container ?to ?egg))
270
     )
271
272
     (:action transfer-mixture
273
       :parameters (?from - container ?to - container ?mixture - mixture_hypothetical)
274
       :precondition (and (pour-mixture-only ?from ?to ?mixture)
         (mixture-in-container ?from ?mixture)
276
         (not (mixture-is-hypothetical ?mixture))
         (is-mixture ?mixture)
         (not (container-in-an-oven ?from))
279
         (not (container-in-an-oven ?to))
280
         (not (pan-is-full ?to)))
281
       :effect (and
282
         (not (mixture-in-container ?from ?mixture))
283
         (mixture-in-container ?to ?mixture))
284
     )
28
286
     (:action transfer-powdery-ingredient
287
       :parameters (?from - container ?to - container ?p -
288
           \hookrightarrow powder_ingredient_hypothetical)
       :precondition (and (pour-powdery-ingredient-from-container ?from ?to ?p)
289
         (powder-ingredient-in-container ?from ?p)
290
         (not (container-in-an-oven ?from))
291
         (not (container-in-an-oven ?to)))
292
       :effect (and
293
         (not (powder-ingredient-in-container ?from ?p))
294
         (powder-ingredient-in-container ?to ?p))
     )
296
297
     (:action use-stand-mixer-in-bowl-for-cake
298
       :parameters (?mixer - electric_stand_mixer ?bowl - container ?mixture -
299
           \rightarrow mixture_hypothetical ?egg0 - egg_hypothetical ?egg1 - egg_hypothetical ?
```

	$\hookrightarrow$ egg2 - egg_hypothetical ?egg3 - egg_hypothetical ?cupsofflour -
	$\hookrightarrow$ powder_ingredient_hypothetical ?tablespoonsofflour -
	$\hookrightarrow$ powder_ingredient_hypothetical ?sugar - powder_ingredient_hypothetical ?
	$\hookrightarrow$ bakingpowder - powder_ingredient_hypothetical ?butter -
	$\hookrightarrow$ butter_hypothetical)
300	:precondition (and (use-stand-mixer ?mixer ?bowl ?mixture)
301	(is-bowl ?bowl)
302	(mixture-is-hypothetical ?mixture)
303	(butter-in-container ?bowl ?butter)
304	(egg-in-container ?bowl ?egg1)
305	(is-baking-powder ?bakingpowder)
306	(is-butter ?butter)
307	(is-cups-of-flour ?cupsofflour)
308	(is-sugar ?sugar)
309	(is-whole-raw-egg ?egg1)
310	(powder-ingredient-in-container ?bowl ?bakingpowder)
311	(powder-ingredient-in-container ?bowl ?cupsofflour)
312	(powder-ingredient-in-container ?bowl ?sugar)
313	(not (egg-in-container ?bowl ?egg0))
314	(not (egg-in-container ?bowl ?egg2))
315	<pre>(not (egg-in-container ?bowl ?egg3)))</pre>
316	:effect (and
317	(sugar-is-consumed ?sugar)
318	(butter-is-consumed ?butter)
319	(cups-of-flour-is-consumed ?cupsofflour)
320	(baking-powder-is-consumed ?bakingpowder)
321	<pre>(not (mixture-is-airy ?mixture))</pre>
322	(mixture-in-container ?bowl ?mixture)
323	(is-mixture ?mixture)
324	<pre>(not (mixture-is-hypothetical ?mixture))</pre>
325	(mixture-has-baking-powder ?mixture)
326	(mixture-has-butter ?mixture)

```
(mixture-has-cups-of-flour ?mixture)
327
         (mixture-has-sugar ?mixture)
328
         (mixture-has-raw-egg-yolk ?mixture)
320
         (mixture-has-blended-raw-egg-whites ?mixture)
330
         (not (butter-in-container ?bowl ?butter))
331
         (not (egg-in-container ?bowl ?egg1))
332
         (not (is-egg ?egg1))
         (not (is-baking-powder ?bakingpowder))
         (not (is-butter ?butter))
335
         (not (is-cups-of-flour ?cupsofflour))
336
         (not (is-sugar ?sugar))
         (not (is-whole-raw-egg ?egg1))
338
         (not (powder-ingredient-in-container ?bowl ?bakingpowder))
339
         (not (powder-ingredient-in-container ?bowl ?cupsofflour))
340
         (not (powder-ingredient-in-container ?bowl ?sugar)))
341
     )
342
343
     (:action use-stand-mixer-in-bowl-for-souffle
344
       :parameters (?mixer - electric_stand_mixer ?bowl - container ?mixture -
345
           \rightarrow mixture_hypothetical ?egg0 - egg_hypothetical ?egg1 - egg_hypothetical ?
           \hookrightarrow egg2 - egg_hypothetical ?egg3 - egg_hypothetical ?cupsofflour -
           \hookrightarrow powder_ingredient_hypothetical ?tablespoonsofflour -
           → powder_ingredient_hypothetical ?sugar - powder_ingredient_hypothetical ?
           \hookrightarrow bakingpowder - powder_ingredient_hypothetical ?butter -
           \hookrightarrow butter_hypothetical)
       :precondition (and (use-stand-mixer ?mixer ?bowl ?mixture)
346
         (is-bowl ?bowl)
347
         (mixture-is-hypothetical ?mixture)
348
         (butter-in-container ?bowl ?butter)
349
         (egg-in-container ?bowl ?egg1)
350
         (is-butter ?butter)
351
         (is-raw-egg-yolk ?egg1)
352
```

353	(is-sugar ?sugar)
354	(is-tablespoons-of-flour ?tablespoonsofflour)
355	(powder-ingredient-in-container ?bowl ?sugar)
356	(powder-ingredient-in-container ?bowl ?tablespoonsofflour)
357	<pre>(not (egg-in-container ?bowl ?egg0))</pre>
358	<pre>(not (egg-in-container ?bowl ?egg2))</pre>
359	<pre>(not (egg-in-container ?bowl ?egg3)))</pre>
360	:effect (and
361	(sugar-is-consumed ?sugar)
362	(butter-is-consumed ?butter)
363	(tablespoons-of-flour-is-consumed ?tablespoonsofflour)
364	<pre>(not (mixture-is-airy ?mixture))</pre>
365	(mixture-in-container ?bowl ?mixture)
366	(is-mixture ?mixture)
367	<pre>(not (mixture-is-hypothetical ?mixture))</pre>
368	(mixture-has-butter ?mixture)
369	(mixture-has-raw-egg-yolk ?mixture)
370	(mixture-has-sugar ?mixture)
371	(mixture-has-tablespoons-of-flour ?mixture)
372	<pre>(not (butter-in-container ?bowl ?butter))</pre>
373	<pre>(not (egg-in-container ?bowl ?egg1))</pre>
374	(not (is-butter ?butter))
375	(not (is-raw-egg-yolk ?egg1))
376	<pre>(not (is-egg ?egg1))</pre>
377	(not (is-sugar ?sugar))
378	<pre>(not (is-tablespoons-of-flour ?tablespoonsofflour))</pre>
379	<pre>(not (powder-ingredient-in-container ?bowl ?sugar))</pre>
380	<pre>(not (powder-ingredient-in-container ?bowl ?tablespoonsofflour)))</pre>
381	)

# A.5.3 Largest Set of Objects in a Problem

1	bowl-0 - container
2	bowl-1 - container
3	butter-0 - butter_hypothetical
4	butter-1 - butter_hypothetical
5	dessert-0 - dessert_hypothetical
6	dessert-1 - dessert_hypothetical
7	egg-0 - egg_hypothetical
8	egg-1 - egg_hypothetical
9	egg-2 - egg_hypothetical
10	egg-3 - egg_hypothetical
11	flour-2 - powder_ingredient_hypothetical
12	flour-3 - powder_ingredient_hypothetical
13	<pre>measuring-cup-0 - measuring_cup</pre>
14	<pre>measuring-cup-1 - measuring_cup</pre>
15	<pre>measuring-cup-2 - measuring_cup</pre>
16	<pre>measuring-cup-3 - measuring_cup</pre>
17	<pre>measuring-cup-4 - measuring_cup</pre>
18	<pre>measuring-cup-5 - measuring_cup</pre>
19	<pre>measuring-cup-6 - measuring_cup</pre>
20	measuring-cup-7 - measuring_cup
21	<pre>measuring-cup-8 - measuring_cup</pre>
22	<pre>measuring-cup-9 - measuring_cup</pre>
23	mixer-0 - electric_stand_mixer
24	mixture-0 - mixture_hypothetical
25	mixture-1 - mixture_hypothetical
26	mixture-2 - mixture_hypothetical
27	mixture-3 - mixture_hypothetical
28	<pre>mixture-4 - mixture_hypothetical</pre>
29	mixture-5 - mixture_hypothetical
30	mixture-6 - mixture_hypothetical

31	mixture-7 - mixture_hypothetical
32	mixture-8 - mixture_hypothetical
33	oven-0 - oven
34	oven-1 - oven
35	pan-0 - container
36	pan-1 - container
37	plate-0 - container
38	plate-1 - container
39	spatula-0 - spatula
40	<pre>sugar-0 - powder_ingredient_hypothetical</pre>
41	<pre>sugar-1 - powder_ingredient_hypothetical</pre>

# Appendix B

# **Prompt** Templates

This appendix presents several prompt templates used in experiments with LLMs throughout this thesis.

## **B.1** Prompt Templates for Generating Operators

In one set of experiments, we prompted the LLM to generate operators to initialize GLIB. We did not observe a significant difference in effectiveness across different prompting techniques, likely due to the simplicity and small scale of the domains used in these experiments (these techniques were not tested on Baking-Large).

#### Fill-in-the-Blank PDDL Domain Template

This template uses the predicates, object types, and domain name as inputs to generate the prompt. Below is an example prompt for the Blocks domain.

```
(holding ?v0 - block)
8
                   (on ?v0 - block ?v1 - block)
9
                   (ontable ?v0 - block)
10
          )
12
          (:action pickup
13
                   :parameters (?v0 - block <TODO>)
14
                   :precondition (and
15
                                   <TODO>
16
                   )
17
                   :effect (and
18
                                   <TODO>
19
                   )
20
          )
21
          (:action putdown
22
                   :parameters (?v0 - block <TODO>)
23
                   :precondition (and
24
                                   <TODO>
25
                   )
26
                   :effect (and
27
                                   <TODO>
28
                   )
29
          )
30
          (:action stack
31
                   :parameters (?v0 - block ?v1 - block <TODO>)
32
                   :precondition (and
33
                                   <TODO>
34
                   )
35
                   :effect (and
36
                                   <TODO>
37
                  )
38
          )
39
```

```
(:action unstack
40
                    :parameters (?v0 - block <TODO>)
41
                    :precondition (and
42
                                      <TODO>
43
                    )
44
                    :effect (and
4.5
                                      <TODO>
46
                    )
47
           )
48
  )
49
```

## B.1.1 Few-Shot Chain-of-Thought Prompt Template

This prompting pipeline, inspired by ADA [41] and adapted for our problem setting, is designed to generate PDDL operators with action predicates. The pipeline consists of four stages:

- 1. **Goal Translation**: Since our goals are conjunctions of predicates, the LLM is prompted using few-shot examples to translate these into natural language descriptions.
- 2. Task Decomposition: Using the natural language goal, the LLM is prompted using few-shot examples to generate a sequence of action predicates required to achieve the goal.
- 3. **Operator Definition**: Based on the action names generated in the previous step, the LLM is prompted using few-shot examples to define operators (including parameters, preconditions, and effects) without associating them with action predicates.
- 4. **Skill Association**: Finally, the LLM is prompted using few-shot examples to associate each operator from the previous step with a specific action predicate defined in the domain.

The following templates illustrate the use of few-shot examples from domains not included in our experiments.

#### **Goal Translation Prompt Template**

Few-shot examples from domains not covered in this thesis are used to prompt the LLM to generate natural language descriptions of goals represented in predicate form. The template is shown below.

```
""";;;; Translate the goal into natural language.
  Q: (and (movie-rewound)
3
      (counter-at-zero)
      (have-chips)
Ę
      (have-dip)
      (have-pop)
7
      (have-cheese)
8
      (have-crackers)
9
      )
11
12 A: The movie is rewound, the counter is at zero, and the agent has chips, dip, pop,
      \hookrightarrow cheese, and crackers.
13
  Q: (and (person-at person0 f5-5f))
14
15
  A: person0 is at location f5-5f.
16
17
  Q: (and (ontable shot15)
18
      (dispenses dispenser1 ingredient1)
19
      (contains shot1 cocktail5)
20
      )
21
22
  A: The shot15 glass is on the table, the dispenser dispenser1 is set up to dispense
23
      \hookrightarrow ingredient1, and the shot glass shot1 contains cocktail5.
24
25 Q: (and (at bear-0 loc-4-2) (holding pawn-1) )
```

```
<sup>26</sup>
A: bear-0 is at location loc-4-2, and the agent is holding pawn-1.
<sup>28</sup>
<sup>29</sup> Q: {our_goal}
<sup>31</sup> """
```

#### Task Decomposition Prompt Template

We prompt for a sequence of action predicates using few-shot examples from domains not used in our experiments. Here is the template.

```
""";;;; Given natural language goals, predict a sequence of PDDL actions.
2
 Q:
4
6 Domain: Casino
7 Goal: The agent has secured prize-1-1 and prize-1-2 from the first prize group,
     \hookrightarrow prize-2-1 and prize-2-2 from the second prize group, and prize-3-1 and prize
     \hookrightarrow -3-2 from the third prize group.
 Objects: loc-3-3:location,loc-5-0:location,loc-1-6:location,loc-7-8:location,loc
     \leftrightarrow -3-1:location,loc-3-9:location,loc-7-6:location,loc-2-0:location,loc-0-0:
     \rightarrow location,loc-5-6:location,prize-2-2:prize2,loc-7-2:location,loc-1-1:location
     \rightarrow ,loc-6-1:location,loc-6-5:location,loc-0-2:location,loc-7-3:location,loc
     \leftrightarrow -3-0:location,loc-5-5:location,loc-6-8:location,loc-4-3:location,loc-3-7:
     \leftrightarrow location,loc-4-0:location,prize-1-1:prize1,loc-2-2:location,loc-2-1:location
     \rightarrow ,loc-2-5:location,loc-0-8:location,loc-5-9:location,loc-1-8:location,loc
     ← -7-1:location,prize-3-1:prize3,loc-2-4:location,loc-2-8:location,loc-1-9:
     ← location,loc-6-9:location,loc-5-2:location,loc-0-6:location,prize-2-1:prize2
     \leftrightarrow ,loc-5-8:location,loc-2-3:location,loc-4-6:location,loc-5-3:location,loc
     → -2-9:location,loc-3-2:location,loc-4-8:location,loc-4-2:location,loc-6-6:
     \leftrightarrow location, loc-0-7: location, loc-0-1: location, prize-1-2: prize1, loc-6-0: location
```

```
\leftrightarrow ,loc-4-1:location,loc-4-7:location,loc-6-2:location,loc-1-4:location,loc
      \leftrightarrow -2-6:location,loc-3-4:location,loc-6-7:location,loc-0-4:location,loc-0-3:
      \hookrightarrow location, loc-3-6:location, loc-3-8:location, loc-2-7:location, loc-7-0:location
      ↔ ,loc-7-5:location,loc-1-2:location,loc-7-9:location,prize-3-2:prize3,loc
      \leftrightarrow -5-7:location,loc-1-3:location,loc-4-9:location,loc-4-5:location,loc-1-7:
      \hookrightarrow location, loc-6-4: location, loc-7-4: location, loc-1-5: location, loc-0-9: location
      \leftrightarrow ,loc-5-1:location,loc-0-5:location,loc-6-3:location,loc-5-4:location,loc
      \leftrightarrow -4-4:location,loc-7-7:location,loc-1-0:location,loc-3-5:location
9 State: iscasino(loc-5-4), at(loc-0-0)
  A:
11
12
  (moveto loc-0-0 loc-5-4), (getprize3 prize-3-2 loc-5-4), (getprize3 prize-3-1 loc
13
      → -5-4),(getprize2 prize-2-2 loc-5-4),(getprize2 prize-2-1 loc-5-4),(getprize1
      \hookrightarrow prize-1-2 loc-5-4), (getprize1 prize-1-1 loc-5-4)
14
  Q:
15
16
17
  Domain: Elevator
  Goal: Service has been provided to person p0.
18
19 Objects: f1:floor,p0:passenger,f0:floor
20 State: lift-at(f0), origin(p0,f1), above(f0,f1), destin(p0,f0)
21
  A:
22
23
  (up f0 f1), (board f1 p0), (down f1 f0), (depart f0 p0)
24
25
  Q:
26
27
28 Domain: {domain_name}
29 {goal}
30 {state}
```



#### **Operator Definition Prompt Template**

We prompt the LLM to define operators without associating them with action predicates, using few-shot examples from domains not included in our experiments. The template is provided below.

```
f""";;;; You are a software engineer who will be writing planning operators in the
      \hookrightarrow PDDL planning language. These operators are based on the following PDDL
      \hookrightarrow domain definition.
  ### The predicates in Sokoban are:
3
  (move-dir ?v0 - location ?v1 - location ?v2 - direction)
5
    (is-nongoal ?v0 - location)
    (clear ?v0 - location)
7
    (is-stone ?v0 - thing)
8
    (at ?v0 - thing ?v1 - location)
9
    (is-player ?v0 - thing)
    (at-goal ?v0 - thing)
11
    (move ?v0 - direction)
12
    (is-goal ?v0 - location)
13
14
  Q: Propose a PDDL operator called "move".
15
16
  A: (:action move
17
      :parameters (?p - thing ?from - location ?to - location ?dir - direction)
18
      :precondition (and (move ?dir)
19
        (is-player ?p)
20
        (at ?p ?from)
21
```

```
(clear ?to)
22
        (move-dir ?from ?to ?dir))
23
      :effect (and
24
        (not (at ?p ?from))
25
        (not (clear ?to))
26
        (at ?p ?to)
27
        (clear ?from))
28
    )
29
30
  Q: Propose an operator called "push-to-goal".
31
  A: (:action push-to-goal
32
      :parameters (?p - thing ?s - thing ?ppos - location ?from - location ?to -
33
          \hookrightarrow location ?dir - direction)
      :precondition (and (move ?dir)
34
        (is-player ?p)
35
        (is-stone ?s)
36
        (at ?p ?ppos)
37
        (at ?s ?from)
38
        (clear ?to)
39
        (move-dir ?ppos ?from ?dir)
40
        (move-dir ?from ?to ?dir)
41
        (is-goal ?to))
42
      :effect (and
43
        (not (at ?p ?ppos))
44
        (not (at ?s ?from))
45
        (not (clear ?to))
46
        (at ?p ?from)
47
        (at ?s ?to)
48
        (clear ?ppos)
49
        (at-goal ?s))
50
    )
51
```

```
Q: Propose an operator called "push-to-nongoal".
53
     (:action push-to-nongoal
54 A:
      :parameters (?p - thing ?s - thing ?ppos - location ?from - location ?to -
55
          \hookrightarrow location ?dir - direction)
      :precondition (and (move ?dir)
56
        (is-player ?p)
57
        (is-stone ?s)
58
        (at ?p ?ppos)
59
        (at ?s ?from)
60
        (clear ?to)
61
        (move-dir ?ppos ?from ?dir)
62
        (move-dir ?from ?to ?dir)
63
        (is-nongoal ?to))
64
      :effect (and
65
        (not (at ?p ?ppos))
66
        (not (at ?s ?from))
67
        (not (clear ?to))
68
        (at ?p ?from)
69
        (at ?s ?to)
70
        (clear ?ppos)
71
        (not (at-goal ?s)))
72
    )
73
74
75
76 ### The predicates in {domain_name} are:
77
  {predicates}
78
79
80 Q: Propose an operator called {op_name}.
81
82 A:
  0.0.0
83
```

#### Skill Association Prompt Template

We prompt the LLM to associate each generated operator with an action predicate defined in the domain, using few-shot examples from domains not included in our experiments. The template is shown below:

```
""";;;; Given the list of skills and a PDDL operator, pick the skill that is needed
      \hookrightarrow to execute the PDDL operator.
3 Q:
5 Domain: Sokoban
6 Skills: move(?v0 - direction), throw(?v0 - ball), walk(?v3 - loc), pick(?v1 -
      \hookrightarrow object)
7 Operator:
  (:action push-to-goal
8
      :parameters (?p - thing ?s - thing ?ppos - location ?from - location ?to -
          \hookrightarrow location ?dir - direction)
      :precondition (and (move ?dir)
10
        (is-player ?p)
11
        (is-stone ?s)
12
        (at ?p ?ppos)
13
        (at ?s ?from)
14
        (clear ?to)
15
        (move-dir ?ppos ?from ?dir)
16
        (move-dir ?from ?to ?dir)
17
        (is-goal ?to))
18
      :effect (and
19
        (not (at ?p ?ppos))
20
        (not (at ?s ?from))
21
        (not (clear ?to))
22
```

```
(at ?p ?from)
23
        (at ?s ?to)
24
        (clear ?ppos)
25
        (at-goal ?s))
26
    )
27
28
  A: move
29
30
  Q:
31
32
33 Domain: Spanner
34 Skills: (walk ?v0 - location ?v1 - location ?v2 - man), (pickup_spanner ?v0 -
      \hookrightarrow location ?v1 - spanner ?v2 - man),(rotate ?v0 - location ?v1 - spanner ?v2 -
      \hookrightarrow man ?v3 - nut)
35 Operator:
  (:action tighten_nut
36
           :parameters (?1 - location ?s - spanner ?m - man ?n - nut)
37
          :precondition (and (at ?m ?l)
38
                  (at ?n ?1)
39
            (carrying ?m ?s)
40
           (useable ?s)
41
           (loose ?n))
42
          :effect (and (not (loose ?n))(not (useable ?s)) (tightened ?n)))
43
44 )
45
46 A: rotate
47
48
49 Q:
50
51 Domain: {domain_name}
52 Skills: {skills_list}
```

```
53 Operator:
54 {operator_string}
55 
56 A:
57 """
```

## **B.2** Predicate Descriptions for Baking-Large

To prompt GPT-4 for open-loop plans, we used handcrafted natural language descriptions of predicates and goals. This section provides the descriptions used in our experiments.

### **B.2.1** Grounded Literal Descriptions

Below is the mapping of each predicate name to its corresponding natural language description. Brackets indicate placeholders for the object names that serve as arguments for the ground literal of the predicate.

```
{
              "pan-is-full": "The pan {} is full and can't contain anymore things.",
              "cups-of-flour-is-consumed": "{} is used up. ",
              "baking-powder-is-consumed": "{} is used up. ",
              "mixture-has-blended-raw-egg-whites": "{} has raw egg whites that were
                  \hookrightarrow smoothly and uniformly mixed in. ",
              "mixture-has-folded-raw-egg-whites": "{} has folded raw egg whites with
                  \hookrightarrow pockets of air in them. ",
              "sugar-is-consumed": "{} is used up. ",
              "butter-is-consumed": "{} is used up.",
              "mixture-is-hypothetical": "{} is a hypothetical object that doesn't
                  \hookrightarrow exist yet. ",
              "is-whole-raw-egg": "{} is out of its shell and is a whole raw egg, with
                  \hookrightarrow the egg whites and yolk not separated. ",
              "powder-ingredient-in-measuring-cup": "{} is in the measuring cup {}. ",
11
```

12	"butter-in-measuring-cup": "{} is in the measuring cup {}. ",
13	"dessert-is-hypothetical": "{} is a hypothetical dessert that has not
	$\hookrightarrow$ been made yet. ",
14	"egg-is-hypothetical": "{} references a hypothetical egg (partial egg or
	$\hookrightarrow$ whole) that has not been made yet. ",
15	"is-baking-powder": "{} is chemically and physically identifiable as
	$\hookrightarrow$ baking powder. ",
16	"is-raw-egg-yolk": "{} is a raw egg yolk out of the shell. ",
17	"is-raw-egg-whites": "{} is raw egg whites out of the shell. ",
18	"is-whipped-egg-whites": "{} is whipped egg whites. ",
19	"is-in-shell": "The egg {} is in its uncracked shell. ",
20	"is-egg": "The egg {} is chemically and physically still an egg, and
	$\hookrightarrow$ separable from other objects. ",
21	"mixture-is-airy": "The mixture {} is light and airy, and has air
	$\hookrightarrow$ pockets. ",
22	"pan-is-damaged": "{} is damaged. ",
23	"is-whole-egg": "The egg {} is raw and whole, containing both the yolk
	$\hookrightarrow$ and egg whites. ",
24	"is-tablespoons-of-flour": "{} is a couple of tablespoons of flour. ",
25	"is-cups-of-flour": "{} is a couple of cups of flour. ",
26	"is-sugar": "{} is chemically and physically still separate from other
	$\hookrightarrow$ objects and identifiable as sugar. ",
27	"is-butter": "{} is chemically and physically still separate from other
	$\hookrightarrow$ objects and identifiable as butter. ",
28	"oven-is-full": "{} is at capacity. ",
29	"is-pan": "The container {} is a pan. ",
30	"is-bowl": "The container {} is a bowl. ",
31	"is-mixture": "{} is not hypothetical and exists. ",
32	"is-plate": "The container {} is a plate. ",
33	"container-is-full": "The container {} is full and cannot hold any more
	$\hookrightarrow$ objects. ",
34	"powder-ingredient-in-container": "The container {} has {} in it. ",

35	"butter-in-container": "The container {} has {} in it. ",
36	"egg-in-container": "The container {} has {} in it. ",
37	"dessert-in-container": "The container {} has {} in it. ",
38	"mixture-in-container": "The container {} has {} in it. ",
39	"container-in-oven": "The container {} is in the oven {}. ",
40	"container-in-an-oven": "The container {} is in an oven. ",
41	"mixture-has-raw-egg-yolk": "The mixture {} contains raw egg yolk. ",
42	"mixture-has-cups-of-flour": "The mixture {} contains cups of flour. ",
43	<pre>"mixture-has-tablespoons-of-flour":"The mixture {} contains tablespoons</pre>
	$\hookrightarrow$ of flour. ",
44	"mixture-has-raw-egg-whites": "The mixture {} contains raw egg whites. "
	$\hookrightarrow$ ,
45	"mixture-has-sugar": "The mixture {} contains sugar. ",
46	"mixture-has-butter": "The mixture {} contains butter. ",
47	"mixture-has-baking-powder": "The mixture {} contains baking powder. ",
48	"is-cake": "{} is not hypothetical, and is a real, physical cake. ",
49	"is-souffle": "{} is not hypothetical, and is a light and airy souffle.
	$\hookrightarrow$ ",
50	"oven-is-heated-for-souffle": "The oven {} is preheated for a souffle. "
	$\hookrightarrow$ ,
51	"oven-is-heated-for-cake": "The oven {} is preheated for a cake. ",
52	"pour-powdery-ingredient-from-measuring-cup": "Pour the entire {} in {}
	$\hookrightarrow$ into {}. ",
53	"pour-mixture-only": "Pour the entire mixture {} from {} into {}. ",
54	"pour-powdery-ingredient-from-container": "Pour all of the {} from {}
	$\hookrightarrow$ into {}. ",
55	"transfer-butter-from-pan-or-bowl": "Move all of the butter {} from
	$\hookrightarrow$ container {} into container {}. ",
56	"transfer-egg-from-pan-or-bowl": "Move the egg object {} from container
	$\hookrightarrow$ {} into container {}. ",
57	"move-baked-good-in-container-to-different-container": "Move the dessert
	$\hookrightarrow$ from {} into {}. ",

58	"crack-egg-and-put-in-container": "Crack the egg {} into {} and discard
	$\hookrightarrow$ the eggshell. ",
59	"put-butter-in-container-from-measuring-cup": "Put the butter {} in the
	$\hookrightarrow$ measuring cup into {}. ",
60	"put-pan-in-oven": "Put the {} on the kitchen counter in the {}. ",
61	"preheat-oven-with-cake-settings": "Close the oven door if it's open,
	$\hookrightarrow$ and preheat the oven {} at 350 degrees Fahrenheit, the
	$\hookrightarrow$ temperature at which to bake cakes. ",
62	"preheat-oven-with-souffle-settings": "Close the oven door if it's open,
	$\hookrightarrow$ and preheat the oven {} at 375 degrees Fahrenheit, the
	$\hookrightarrow$ temperature at which to bake souffles. ",
63	"use-stand-mixer": "Mix the ingredients in {} with the mixer {} to
	$\hookrightarrow$ create a new mixture {} with all the ingredients fused together.
	$\hookrightarrow$ ",
64	<pre>number "remove-pan-from-oven": "Remove the pan {} from the oven it is in and place</pre>
	$\hookrightarrow$ it on the counter to cool, leaving the oven door open. ",
65	"set-oven-with-cake-bake-time-and-press-start": "Set the amount of time
	$\hookrightarrow$ to bake a cake and press 'start' on {} to begin baking, not
	$\hookrightarrow$ opening the oven again until done baking. ",
66	"set-oven-with-souffle-bake-time-and-press-start": "Set the amount of
	$\hookrightarrow$ time to bake a souffle and press 'start' on {} to begin baking,
	$\hookrightarrow$ not opening the oven again until done baking. ",
67	"separate-raw-yolk-from-egg-whites": "Separate the raw egg {} that is
	$\hookrightarrow$ out of its shell into an egg yolk, now called {} and in the
	$\hookrightarrow$ container {}, and egg whites called {} and in the container {}. "
	$\hookrightarrow$ ,
68	"beat-egg-whites": "Beat the egg {} with the mixer {} in the {}. ",
69	"fold-stiff-egg-whites-into-mixture": "Fold the egg whites {} in {} into
	$\hookrightarrow$ the contents of {}, using the {}. "
70	}

## B.2.2 Action Predicate Descriptions

Below is the mapping of each action predicate name to its corresponding description.

1	{
2	"pour-powdery-ingredient-from-measuring-cup": "Pour all of a powdery
	$\hookrightarrow$ ingredient in a measuring cup into a container.",
3	"pour-mixture-only": "Pour the entire mixture that is inside a container
	$\hookrightarrow$ into another container.",
4	"pour-powdery-ingredient-from-container": "Pour all of the powdery
	$\hookrightarrow$ ingredient inside a container into another container.",
5	"transfer-butter-from-pan-or-bowl": "Move all of the butter from a
	$\hookrightarrow$ container into another container.",
6	"transfer-egg-from-pan-or-bowl": "Move the egg object from a container into
	$\hookrightarrow$ another container.",
7	"move-baked-good-in-container-to-different-container": "Move a dessert from
	$\hookrightarrow$ a container into another container.",
8	"crack-egg-and-put-in-container": "Crack a raw egg into a container and
	$\hookrightarrow$ discard its eggshell.",
9	"put-butter-in-container-from-measuring-cup": "Put all of the butter in the
	$\hookrightarrow$ measuring cup into the container.",
10	"put-pan-in-oven": "Open the oven door, put the pan in the oven, and close
	$\hookrightarrow$ the oven door.",
11	"preheat-oven-with-cake-settings": "Close the oven door if it's open, and
	$\hookrightarrow$ preheat the oven at 350 degrees Fahrenheit, the temperature at which
	$\hookrightarrow$ to bake cakes.",
12	"preheat-oven-with-souffle-settings": "Close the oven door if it's open,
	$\hookrightarrow$ and preheat the oven at 375 degrees Fahrenheit, the temperature at
	$\hookrightarrow$ which to bake souffles.",
13	"use-stand-mixer": "Mix the ingredients in the container to create a smooth
	$\hookrightarrow$ mixture, so that the ingredients in the container are no longer
	$\hookrightarrow$ separable.",
14	"remove-pan-from-oven": "Open the oven door, remove a pan from the oven it
	$\hookrightarrow$ is in and place it on the counter to cool, leaving the oven door open
----	---
	$\hookrightarrow$ .",
15	"set-oven-with-cake-bake-time-and-press-start": "Set the amount of time to
	$\hookrightarrow$ bake a cake and press 'start' on the oven to begin baking, not
	$\hookrightarrow$ opening the oven again until done baking.",
16	"set-oven-with-souffle-bake-time-and-press-start": "Set the amount of time
	$\hookrightarrow$ to bake a souffle and press 'start' on the oven to begin baking, not
	$\hookrightarrow$ opening the oven again until done baking.",
17	"separate-raw-yolk-from-egg-whites": "Separate a raw egg that is out of its
	$\hookrightarrow$ shell into an egg yolk and egg whites and place them in containers."
	$\hookrightarrow$ ,
18	"beat-egg-whites": "Beat the egg whites in the container with the electric
	$\hookrightarrow$ mixer until stiff peaks form.",
19	"fold-stiff-egg-whites-into-mixture": "Fold the egg whites in the container
	$\hookrightarrow$ into the mixture in the container, using the spatula."
20	}

# B.2.3 Lifted Action Literal Variable Descriptions

We provide hand-written descriptions of each variable in the lifted action literal in Baking-Large. Below is a map from the action name to a list of descriptions for each of the variables.

```
"the container to pour the mixture into",
                  "the mixture to be poured between the containers"
             ],
              "pour-powdery-ingredient-from-container": [
12
                  "the container that has the powdery ingredient right now",
                 "the container to pour the powdery ingredient into",
                 "the powdery ingredient to be poured between the containers"
             ],
16
              "transfer-butter-from-pan-or-bowl": [
                  "the container that the butter is in right now",
18
                 "the container to put the butter in",
                 "the butter that will be moved"
20
             ],
21
              "transfer-egg-from-pan-or-bowl": [
22
                 "the container that the egg is in right now",
23
                  "the container to put the egg in",
24
                  "the egg that will be moved"
25
             ],
26
              "move-baked-good-in-container-to-different-container": [
                  "the container that the dessert is in right now",
28
                 "the container to put the dessert in",
29
                 "the dessert that will be moved"
30
             ],
31
              "crack-egg-and-put-in-container": [
32
                 "the egg that is to be cracked",
33
                 "the container that will hold the whole raw egg after cracking it
34
                     \hookrightarrow out of its shell"
             ],
35
              "put-butter-in-container-from-measuring-cup": [
36
                 "the butter that is in the measuring cup currently",
37
                  "the container that the butter will be in after moving it from the
38
                     \hookrightarrow measuring cup"
```

```
],
39
              "put-pan-in-oven": [
40
                  "the pan that is to be moved",
41
                  "the oven that the container will be in after moving"
42
             ],
43
              "preheat-oven-with-cake-settings": [
44
                  "the oven to preheat"
45
             ],
46
              "preheat-oven-with-souffle-settings": [
47
                  "the oven to preheat"
48
             ],
49
              "use-stand-mixer": [
50
                  "the mixer to use",
                  "the container containing ingredients to mix",
                  "the name of the new mixture that didn't exist before"
             ],
54
              "remove-pan-from-oven": [
                  "the container to remove from the oven",
56
                  "the oven that the container is in"
             ],
58
              "set-oven-with-cake-bake-time-and-press-start": [
                  "the oven to start baking in",
60
                  "the name of the dessert object that will become a physical cake",
61
                  "the cake mixture that we are baking"
62
             ],
63
              "set-oven-with-souffle-bake-time-and-press-start": [
                  "the oven to start baking in",
65
                  "the name of the dessert object that will become a physical souffle"
66
                     \hookrightarrow ,
                  "the souffle mixture that we are baking"
67
             ],
68
              "separate-raw-yolk-from-egg-whites": [
```

```
"the name of the whole raw egg to do the separation on, whose name
70
                      \hookrightarrow will represent the egg yolk after separation",
                  "the hypothetical egg object that will represent the egg whites
71
                      \hookrightarrow after separation",
                  "the name of the container that will contain the raw egg yolk after
72
                      \hookrightarrow separation",
                  "the name of the container that will contain the raw egg whites
73
                      \hookrightarrow after separation"
              ],
74
              "beat-egg-whites": [
75
                  "the electric stand mixer to use to beat the egg whites",
76
                  "the container holding the egg whites",
77
                  "the egg whites"
78
              ],
79
              "fold-stiff-egg-whites-into-mixture": [
80
                  "the spatula to use for folding the egg whites",
81
                  "the container containing the stiff egg whites",
82
                  "the container containing the mixture",
83
                  "the stiff, beaten egg whites",
84
                  "the mixture to fold the egg whites into"
85
              ]
86
  }
87
```

## **B.2.4** Goal Descriptions

We provide hand-written natural language descriptions for goals. Below is the mapping of each training problem to its corresponding goal description:

# B.3 Prompt Templates for Generating Plan Sketches Open-Loop in Baking-Large

To generate plan sketches in Baking-Large, we initiate a new conversation with the LLM using four user prompts:

- 1. **Domain Introduction**: The first prompt introduces the domain and asks the LLM to role-play as the agent.
- 2. Initial State Introduction: The second prompt describes the initial state of the agent.
- 3. Goal Introduction: The third prompt specifies the goal for the agent.
- 4. Action Sequence Request: The fourth prompt asks the LLM to generate the sequence of actions needed to achieve the goal.

The response from GPT-4 to each prompt is appended to the conversation context.

## Introduction Prompt

f"""

```
You are a household robot in a kitchen. You are in front of the kitchen counter,

→ where there are some prepared ingredients.
```

Since you are baking desserts, first determine what are the differences between a  $\hookrightarrow$  cake and sweet, light, and airy souffle. Please rationalize what are the  $\hookrightarrow$  essential ingredients and their amounts to make those desserts and use only  $\hookrightarrow$  those.

### **Initial State Introduction Prompt Template**

0.0.0

The initial state description is a formatted string created by combining the predicate descriptions from Section B.2. This description is used to populate the following template, forming our prompt.

```
f"""
<sup>2</sup> In the kitchen, there different kinds of objects that you can interact with. The
      \hookrightarrow different kind of objects that you see are categorized into the following:
  container
  measuring cup
5
F
  dessert
 powder
7
  butter
8
9 mixture
10 egg
  oven
12 spatula
13 electric stand mixer
14
```

```
15 Right now, you see the some of these ingredients and items on the counter. You also
      \hookrightarrow see some appliances in the kitchen. The following things are true at this
      \hookrightarrow moment:
16
  {initial_state_description}
17
18
  As a reminder, in the kitchen, the pans, measuring cups, and bowls are on the
19
      \hookrightarrow counter, and the oven(s) is (are) behind the counter. If you are baking
      \hookrightarrow desserts, please rationalize what are the essential ingredients and their
      \hookrightarrow amounts to make those desserts and use only those. Once an ingredient is
      \hookrightarrow used once, it can't be reused.
20
21 You should have all of the ingredients that you need on the counter prepared for
      \hookrightarrow you. I'll let you know what desserts you will make shortly.
  .....
22
```

#### **Goal Introduction Prompt Template**

We fill in this prompt template with a goal description from Section B.2.4 to form our prompt.

### Action Sequence Request Prompt Template

We use the handwritten action predicate descriptions from Section B.2.2 to construct a string containing all the action descriptions. This string is then used to populate the below prompt template, forming the input for generating the plan sketch.

# **B.4** Prompt Templates for Grounding Plans

Using the sequence of action predicate names provided by the LLM, we generated the grounded plan using the following prompt templates.

To ground the actions sequentially according to the action names (plan sketch) from the LLM, we prompted step-by-step, appending each response to the ongoing conversation. Grounding begins with the first action using this prompt:

0.0.0

 $\hookrightarrow$  action\_description\_with\_nonspecific\_articles}. We need to identify the names  $\hookrightarrow$  of the specific objects involved in this action. Here are more details  $\hookrightarrow$  about how the objects involved need to relate to the action:

#### """ + '\n'.join(variable\_description\_list)

In this prompt, the following components are used:

- actions\_done\_string: Contains the ground literal descriptions (from Section B.2) of actions earlier in the plan that the LLM has already grounded.
- actions list: The list of parsed ground actions that have been accumulating.
- action\_description\_with\_nonspecific\_articles: The action description from Section B.2.2.
- variable\_description\_list: The list of variable descriptions for the action, retrieved from Section B.2.3.

We then prompt the LLM to assign each variable in the lifted action literal to a specific object in the state:

In this prompt:

- action\_description\_with\_nonspecific\_articles: This is the same action description used in the previous prompt.
- **variable\_description**: The description of the specific variable being grounded, corresponding to the current iteration over the *variable\_description\_list* from the previous prompt.

• objects list: The list of object names observed in the current state.

We parse the LLM's response and add the grounded action literal to our plan. This process is repeated for all actions in the plan sketch provided by the LLM. Once all actions are grounded, the complete grounded plan is returned for execution to gather demonstrations.

# References

- A. Ahmetoglu, M. Y. Seker, J. Piater, E. Oztop, and E. Ugur. "Deepsym: Deep symbol generation and rule learning from unsupervised continuous robot interaction for planning". In: arXiv preprint arXiv:2012.02532 (2020).
- S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup. "A survey of exploration methods in reinforcement learning". In: arXiv preprint arXiv:2109.00157 (2021). URL: https://arxiv.org/pdf/2109.00157.pdf.
- [3] A. Arora, H. Fiorino, D. Pellier, M. Métivier, and S. Pesty. "A review of learning planning action models". In: *The Knowledge Engineering Review* (2018).
- [4] M. Asai and A. Fukunaga. "Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary". In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. Ed. by S. A. McIlraith and K. Q. Weinberger. AAAI Press, 2018, pp. 6094–6101. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16302.
- R. I. Brafman and M. Tennenholtz. "R-max a general polynomial time algorithm for near-optimal reinforcement learning". In: J. Mach. Learn. Res. 3.null (2003). ISSN: 1532-4435. DOI: 10.1162/153244303765208377. URL: https://doi.org/10.1162/ 153244303765208377.
- [6] R. Chitnis, T. Silver, J. B. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. "GLIB: Exploration via Goal-Literal Babbling for Lifted Operator Learning". In: *CoRR* abs/2001.08299 (2020). arXiv: 2001.08299. URL: https://arxiv.org/abs/2001.08299.

- [7] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. "Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning". In: AAAI CLeaR Workshop. 2022.
- [8] C. Colas, P. Fournier, M. Chetouani, O. Sigaud, and P.-Y. Oudeyer. "Curious: intrinsically motivated modular multi-goal reinforcement learning". In: *International Conference on Machine Learning (ICML)*. 2019. URL: https://proceedings.mlr.press/v97/ colas19a/colas19a.pdf.
- K. Driessens and S. Džeroski. "Integrating Guidance into Relational Reinforcement Learning". In: Mach. Learn. 57.3 (Dec. 2004), pp. 271–304. ISSN: 0885-6125. DOI: 10. 1023/B:MACH.0000039779.47329.3a. URL: https://doi.org/10.1023/B:MACH. 0000039779.47329.3a.
- [10] Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas. Guiding Pretraining in Reinforcement Learning with Large Language Models. 2023. arXiv: 2302.06692 [cs.LG]. URL: https://arxiv.org/abs/2302.06692.
- [11] M. Helmert. "The Fast Downward Planning System". In: CoRR abs/1109.6051 (2011).
   arXiv: 1109.6051. URL: http://arxiv.org/abs/1109.6051.
- [12] S. James, B. Rosman, and G. Konidaris. "Autonomous learning of object-centric abstractions for high-level planning". In: Proceedings of the The Tenth International Conference on Learning Representations. 2022.
- [13] L. Kaelbling, H. Pasula, and L. Zettlemoyer. "Learning Symbolic Models of Stochastic Domains". In: Journal of Artificial Intelligence Research 29 (Oct. 2011). DOI: 10.1613/ jair.2113.
- [14] S. Kambhampati, K. Valmeekam, L. Guan, K. Stechly, M. Verma, S. Bhambri, L. Saldyt, and A. Murthy. "LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks". In: arXiv preprint. 2024. URL: https://arxiv.org/pdf/2402.01817.
- [15] M. Kearns and S. Singh. "Near-optimal reinforcement learning in polynomial time". In: *Machine learning* 49 (2002), pp. 209–232.

- [16] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez. "From skills to symbols: Learning symbolic representations for abstract high-level planning". In: *Journal of Artificial Intelligence Research* (2018). URL: https://jair.org/index.php/jair/article/view/ 11175/26380.
- [17] N. Kumar, W. McClinton, R. Chitnis, T. Silver, T. Lozano-Pérez, and L. P. Kaelbling. "Learning Efficient Abstract Planning Models that Choose What to Predict". In: Conference on Robot Learning (CoRL). 2023. URL: https://openreview.net/pdf? id=\_gZLyRGGuo.
- [18] N. Kumar, T. Silver, W. McClinton, L. Zhao, S. Proulx, T. Lozano-Pérez, L. P. Kaelbling, and J. Barry. *Practice Makes Perfect: Planning to Learn Skill Parameter Policies*. 2024. arXiv: 2402.15025 [cs.R0]. URL: https://arxiv.org/abs/2402.15025.
- P. Ladosz, L. Weng, M. Kim, and H. Oh. "Exploration in deep reinforcement learning: A survey". In: *Information Fusion* 85 (2022), pp. 1–22. ISSN: 1566-2535. DOI: https: //doi.org/10.1016/j.inffus.2022.03.003. URL: https://www.sciencedirect.com/science/ article/pii/S1566253522000288.
- [20] T. Lang, M. Toussaint, and K. Kersting. "Exploration in relational domains for modelbased reinforcement learning". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 3725–3768.
- [21] A. Li, N. Kumar, T. Lozano-Pérez, and L. Kaelbling. Learning to Bridge the Gap: Efficient Novelty Recovery with Planning and Reinforcement Learning. 2024. arXiv: 2409.19226 [cs.RO]. URL: https://arxiv.org/abs/2409.19226.
- [22] R. Ma, J. Luijkx, Z. Ajanovic, and J. Kober. "ExploRLLM: Guiding Exploration in Reinforcement Learning with Large Language Models". In: arXiv preprint arXiv:2403.09583 (2024).
- [23] D. Martínez, G. Alenyà, and C. Torras. "Relational reinforcement learning with guided demonstrations". In: Artificial Intelligence 247 (2017). Special Issue on AI and Robotics, pp. 295–312. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2015.02.006. URL: https://www.sciencedirect.com/science/article/pii/S0004370215000284.

- [24] R. Mendonca, O. Rybkin, K. Daniilidis, D. Hafner, and D. Pathak. Discovering and Achieving Goals via World Models. 2021.
- [25] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. "Overcoming exploration in reinforcement learning with demonstrations". In: 2018 IEEE international conference on robotics and automation (ICRA). IEEE. 2018, pp. 6292–6299.
- [26] J. H. A. Ng and R. P. Petrick. "Incremental Learning of Planning Actions in Model-Based Reinforcement Learning." In: *IJCAI*. 2019.
- [27] A. Nie, Y. Su, B. Chang, J. N. Lee, E. H. Chi, Q. V. Le, and M. Chen. EVOLvE: Evaluating and Optimizing LLMs For Exploration. 2024. arXiv: 2410.06238 [cs.LG].
   URL: https://arxiv.org/abs/2410.06238.
- [28] OpenAI et al. GPT-4 Technical Report. 2024. arXiv: 2303.08774 [cs.CL]. URL: https: //arxiv.org/abs/2303.08774.
- [29] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. "Curiosity-driven exploration by self-supervised prediction". In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.
- [30] K. Pertsch, Y. Lee, Y. Wu, and J. J. Lim. Demonstration-Guided Reinforcement Learning with Learned Skills. 2021. arXiv: 2107.10253 [cs.LG]. URL: https://arxiv.org/abs/ 2107.10253.
- [31] C. Rodrigues, P. Gérard, C. Rouveirol, and H. Soldano. "Incremental learning of relational action rules". In: 2010 Ninth International Conference on Machine Learning and Applications. IEEE. 2010.
- [32] L. D. R. Sašo Džeroski and K. Driessens. "Relational Reinforcement Learning". In: Machine Learning (2001).
- [33] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak. "Planning to Explore via Self-Supervised World Models". In: *CoRR* abs/2005.05960 (2020). arXiv: 2005.05960. URL: https://arxiv.org/abs/2005.05960.
- [34] T. Silver and R. Chitnis. PDDLGym: Gym Environments from PDDL Problems. 2020.
   arXiv: 2002.06432 [cs.AI]. URL: https://arxiv.org/abs/2002.06432.

- [35] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum. "Predicate invention for bilevel planning". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2023. URL: https://ojs.aaai.org/ index.php/AAAI/article/view/26429/26201.
- [36] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. "Learning Symbolic Operators for Task and Motion Planning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021. URL: https://arxiv.org/ pdf/2103.00589.pdf.
- [37] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling. "PDDL Planning with Pretrained Large Language Models". In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022. URL: https://openreview.net/forum?id=1QMMUB4zfl.
- [38] R. S. Sutton. "Dyna, an integrated architecture for learning, planning, and reacting".
   In: SIGART Bull. 2.4 (1991). ISSN: 0163-5719. DOI: 10.1145/122344.122377. URL: https://doi.org/10.1145/122344.122377.
- [39] K. Valmeekam, A. Olmo, S. Sreedharan, and S. Kambhampati. "Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change)". In: *NeurIPS 2022 Foundation Models for Decision Making Workshop*. 2022. URL: https: //openreview.net/pdf?id=wUU-7XTL5XO.
- [40] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards". In: arXiv preprint arXiv:1707.08817 (2017).
- [41] L. Wong, J. Mao, P. Sharma, Z. S. Siegel, J. Feng, N. Korneev, J. B. Tenenbaum, and J. Andreas. *Learning adaptive planning representations with natural language guidance*. 2023. arXiv: 2312.08566 [cs.AI]. URL: https://arxiv.org/abs/2312.08566.