From Real World to Logic and Back: Learning Generalizable Relational Concepts For Long Horizon Robot Planning

Naman Shah[1,2], **Jayesh Nagpal**[1], **and Siddharth Srivastava**[1]
[1]School of Computing and Augmented Intelligence, Arizona State University, USA
[2]Department of Computer Science, Brown University, USA

Abstract:
Robots still lag behind humans in their ability to generalize from limited experience, particularly when transferring learned behaviors to long-horizon tasks in unseen environments. We present the first method that enables robots to autonomously invent symbolic, relational concepts directly from a small number of raw, unsegmented, and unannotated demonstrations. From these, the robot learns logic-based world models that support zero-shot generalization to tasks of far greater complexity than those in training. Our framework achieves performance on par with hand-engineered symbolic models, while scaling to execution horizons far beyond training and handling up to 18× more objects than seen during learning. The results demonstrate a framework for autonomously acquiring transferable symbolic abstractions from raw robot experience, contributing toward the development of interpretable, scalable, and generalizable robot planning systems. Project website and code: https://aair-lab.github.io/r2l-lamp.

Keywords: Learning symbolic abstractions, symbolic world model learning, learning for task and motion planning, learning for planning

1 Introduction

The ability to learn from simple examples or demonstrations, and to generalize and transfer that knowledge to solve more complex problems (a hallmark of learning in humans) is a challenging robot learning problem. For instance, all the concepts necessary for clearing a table cluttered with cups are present in one robot trajectory that picks up and places a cup (with no annotations or segmentation). It should therefore be possible to clear a table scattered with cups, given a handful of demonstrations for picking and placing a single cup. Yet, this problem remains a difficult generalization task in robotics — in large part due to the lack of methods for autonomously inventing *generalizable, abstract concepts and world models* that can be transferred to more complex settings. Recent advances in robot learning address complementary problems of learning tasks with short horizons (e.g., closing a door) [1] and of learning to imitate demonstrations in scenarios with minimal differences from training tasks (e.g., tying shoelaces or chopping fruits) [2, 3, 4, 5, 6] .

Core contribution This paper presents the first approach for learning to invent abstract logic-based concepts and world models from raw demonstrations in the form of of kinematic trajectories of the robot without any human annotation, segmenting, or guidance about primitive controllers, actions, or concepts. Fig. 1 shows an overview of our approach. Our input is a small set of raw training trajectories over kinematic states from simple tasks. The output is a set of auto-invented logical concepts with concrete semantic definitions, a set of high-level actions with auto-generated pose-generators for refinement, and a logic-based world model. No human annotation is required in the entire process. Extensive empirical analysis in a range of mobile manipulation settings in

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

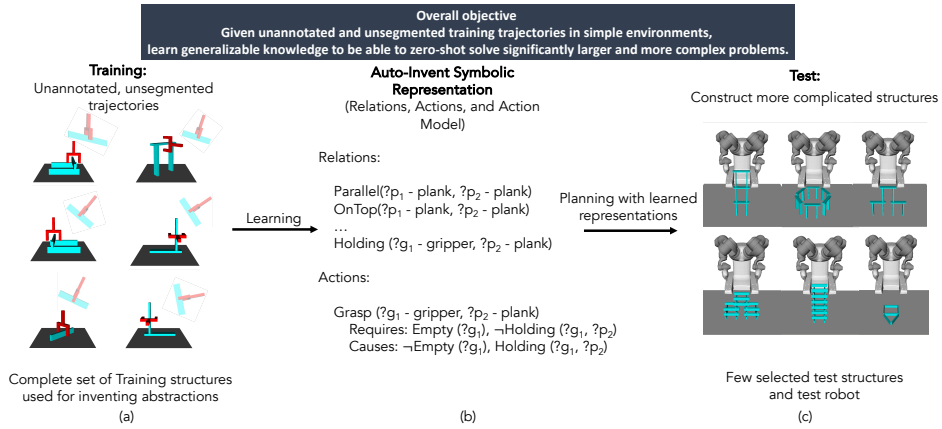


Figure 1: Example of transfer achieved in this paper. (a) shows the complete set of training tasks for one of our experiment domains; (b) shows the automatically invented symbolic concepts and world models by our approach; (c) shows some of the test tasks zero-shot solved using the learned model despite significantly greater numbers and complexity. Interpretable relation names are provided for clarity, not learned by the method.

simulation and in the real world show that the learned concepts enable the robot to solve tasks it has never encountered before, with up to 18× in the number of objects that need to be manipulated, novel goal configurations and significantly larger horizons.

Related work Typically, logic-based concepts are hand-crafted by human experts [7, 8]. This requires extensive domain engineering efforts limiting the applicability of autonomous robots to tasks envisaged prior to deployment. Other related work shows that unary predicates and actions can be learned for motion planning [9, 10] but not for mobile manipulation. Foundation models have been used for composing high-level robot skills provided as Python APIs [11], for translating natural language instructions to logic-based formulas [12], and for planning [13, 14]. These methods rely on expert-provided knowledge about high-level robot actions and the procedures for executing these actions. Recent work indicates that logical concepts are learnable given a-priori knowledge of high-level robot skills or actions [15], and that more complex concepts can be learned from an input set of concepts [16, 17]. However, the problem of learning logical concepts and actions without prior knowledge of either kind has remained understudied at best. We show for the first time that it is possible to solve this problem, and to learn symbolic world models that support complex reasoning for solving significantly larger tasks than the ones that are encountered in training demonstrations. Detailed comparison with other related work is presented in Appendix E .

2 Problem Setting

We represent the set of possible configurations as $\mathcal{X}$, where each state $x \in \mathcal{X}$ specifies 6-dimensional poses (positions and orientations) of objects in the environment, links of the robot, and the set of objects attached to the robot. The set of *primitive actions* ($\mathcal{A}$) represents actions that induce minor changes in the robot pose that are small enough for a native robot controller to execute. Such actions are typically executed in under a second and affect pose changes of at most a few millimeters. However, there are uncountably many infinite primitive actions, many of which transform the configuration space (such as closing the gripper around an object to obtain a stable grasp); tasks of interest in this paper comprise of composing tens of thousands of such actions. This makes naïve planning via optimization over all possible sequences of primitive actions infeasible. Formally, this planning problem can be expressed as an optimization of a utility function over the space of all possible policies as follows:

$$\arg\max_{\pi} \left\{ J\left(\pi, x_0, H\right) \text{ such that } \pi : \mathcal{X} \to \mathcal{A};\ J \text{ is a utility function} \right\}$$

Here, we use a satisficing formulation for J that assigns a positive utility to policies meeting a goal condition $\mathcal{G}$ within H time steps. However, the same method is valid for optimization models where J accounts for action costs. Solving this optimization problem in a deterministic setting would yield a motion plan i.e., a sequence of primitive actions starting from x_0 and reaching a state that fulfills the goal condition $\mathcal{G}$.

Abstract world models Since solving the planning problem at the level of detail of primitive actions is intractable even for simple tasks, the conventional approach is to use hand-crafted abstract logical concepts and create abstract world models for solving multiple tasks in a domain. These concepts include state abstractions in terms of relationships such as "On(obj, table)" and temporal abstractions or conceptual, high-level actions such as "pickup(obj)" and "place(obj, loc)". Thus world models consist of three components: a *logical concept vocabulary* $\mathcal{V}$ with semantics (functions determining when each predicate is true), a set of *high-level actions* $\bar{\mathcal{A}}$, and *action interpreters*, or methods for translating high-level actions into a robot's primitive actions. Each high-level action in $\bar{\mathcal{A}}$ is defined using a precondition (a logical formula over $\mathcal{V}$ describing when the action can be executed) and an effect (a logical formula over $\mathcal{V}$ describing the effects of the action). Each high-level action needs action interpreters that "refine" it into motion plans that can be executed by the robot. Such world models represent knowledge about the world while abstracting away the specifics of robot and object poses, which may change with changes in goals and problem situations [18]. This yields significant generalizability, as the conceptual relations and actions are task agnostic.

The use of abstract concepts changes the formal planning problem to an optimization over abstract policies defined as follows:

$$\arg\max_{[\pi]} \{J(\pi, [x_0], H) \text{ such that } [x_0] = \alpha(x_0) \text{ and } [\pi] : \alpha(\mathcal{X}) \to \beta(\mathcal{A}) \text{ and } \alpha, \beta \text{ satisfy } \kappa\}$$

Here, α and β are hand-coded state and action abstractions respectively that satisfy refinability constraints κ often crafted by expert roboticists based on their experience with the expected tasks and domains of deployment. This is the task and motion planning (TMP) setting [19, 8, 20, 7] (described more in App. A). However, this formulation introduces dependency on hand-crafted knowledge, and it can result in incorrect solutions because it is difficult to design α and β that satisfy refinability.

Critical regions Our approach builds on the notion of *critical regions* [21, 9], which generalizes and unifies the concepts of hubs and bottlenecks. Earlier work defines critical regions in a goal-agnostic manner, however, in this work we consider goal-conditioned critical regions. Intuitively, as the name suggests, goal-conditioned critical regions learn critical regions for a specific training problem. We learn goal-conditioned critical regions for each training task and combine them in order to compute the set of critical regions. Given a robot with a configuration space $\mathcal{X}$, goal-conditioned regions are defined as follows.

Definition 1 *Given a set of solutions for a robot planning problem T, the measure of criticality of a Lebesgue-measurable open set $\rho \subseteq \mathcal{X}$, $\mu(\rho)$, is defined as $\lim_{s_n \to^+ \rho} \frac{f(\rho)}{v(s_n)}$ where $f(r)$ is a fraction of observed motion plans solving the task T that pass through s_n, $v(s_n)$ is the measure of s_n under a reference density (usually uniform), and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing ρ ($\rho \subseteq s_n$, $\forall\, n$).*

3 Our Approach

The objective of this work is to enable robots to create well-founded conceptual world models by inventing abstract concepts. It turns out that by including the abstraction as a parameter in the optimization objective, the agent can learn to perform better as it no longer depends on human-crafted abstractions. Formally, this changes the problem to:

$$\arg\max_{\pi,\alpha,\beta} \{J(\pi, x_0, H) \text{ such that } \pi : \alpha(\mathcal{X}) \to \beta(\mathcal{A}) \text{ and } \alpha, \beta \text{ satisfy } \kappa\}$$

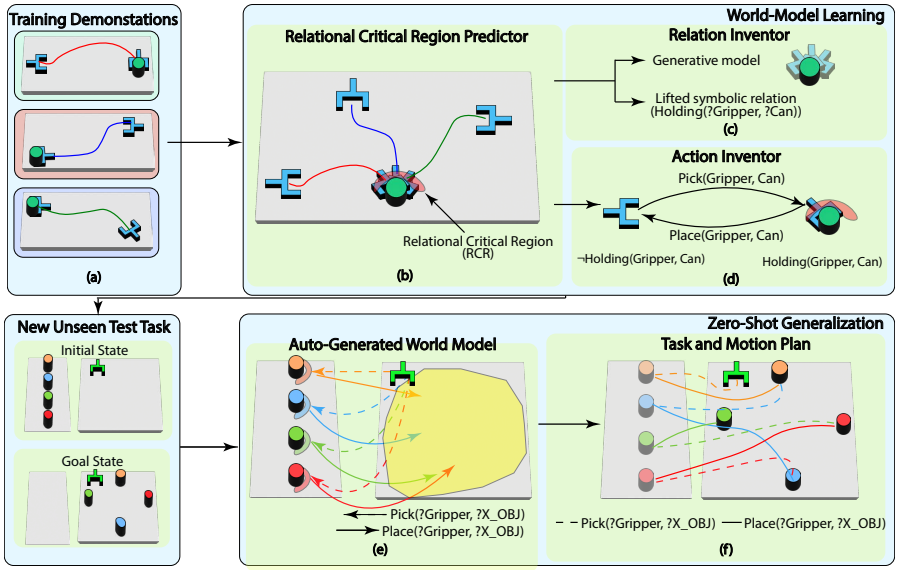


Figure 2: Overview of LAMP. From unlabeled, unsegmented demonstrations (a), LAMP learns relational critical region predictors (b), each defining relations between object pairs (c). These relations form abstract states, with learned actions as transitions (d), together constituting a symbolic world model learned from scratch. For new tasks with more objects or obstructions, LAMP predicts new relations to build the abstract state space and goal, then uses the model to synthesize behaviors that achieve it (f).

where the abstractions become a part of the parameters to be optimized.

Our approach -- *Learning Abstract Models for Planning (**LAMP**)* -- automatically learns these abstractions from kinematic demonstrations and represents them as world models in the planning domain definition language (PDDL [18]). LAMP consists of the following steps (App. B): (i) learn to predict a general, relational form of critical regions called *relational critical regions* that characterize salient regions in the relative state spaces among objects (Sec. 3.1), (ii) invent a symbolic relational vocabulary based on predicted relational critical regions (Sec. 3.2), and (iii) invent high-level actions and learn symbolic world models (Sec. 3.3). The resulting knowledge, i.e. a symbolic world model, is then used to zero-shot solve new tasks. We now present these components in detail.

3.1 Learning Proto-Relations

We postulate that high-level robot actions currently crafted by experts are effectively transitions to and from salient regions within the environment and that such regions can be automatically discerned. E.g., Fig. 2 illustrates a task where the gripper needs to pick up a can. The area from which the can can be grasped constitutes a salient region. Classically hand-crafted "Pick" and "Place" actions transition the gripper into and out of this salient region, respectively. Thus, if we automatically identify such salient regions, high-level actions can also be invented autonomously, subsequently enabling autonomous formulations of generalizable world models for long-horizon problems.

Relational critical regions We formalize the notion of salient regions as *relational critical regions (RCRs)*. Critical regions (CRs, Def. 1) help identify these regions in the robot's configuration space [22]. However, for long-horizon planning involving multiple objects, salient regions occur not in the robot's absolute configuration space but in the space of relative poses among objects. E.g., in Fig 2(a) grasping poses for the can constitute a relational critical region (shaded red in Fig. 2(b)), regardless of the can's location. Intuitively, a relational critical region for object x w.r.t. object y is a region in the set of poses of x relative to y that has a high density of solutions for a given distribution of tasks. We thus generalize critical regions to define *relational critical regions* as follows.

Definition 2 *Let T be a robot planning problem and $\mathcal{D}_T$ be a set of solution trajectories for the planning problem T. Let $o_1, o_2 \in \mathcal{O}$ be a pair of objects, and let $\mathcal{X}_{o_2}^{o_1}$ define the set of relative poses for object o_2 in the frame of o_1. The measure of the criticality of a Lebesgue-measurable open set $\rho \subseteq \mathcal{X}_{o_2}^{o_1}$, $\mu(\rho)$, is defined as $\lim_{s_n \to^+ \rho} \frac{f(\rho)}{v(s_n)}$ where $f(\rho)$ is the fraction of observed solution trajectories solving for the planning problem T that contains a relative pose $P_{o_2}^{o_1}$ such that, $P_{o_2}^{o_1} \in \rho$ $v(s_n)$ is the measure of s_n under a reference density (usually uniform); and $\to^+$ denotes the limit from above along any sequence $\{s_n\}$ of sets containing ρ ($\rho \subseteq s_n$, $\forall n$). A region $\rho \subseteq \mathcal{X}_{o_2}^{o_1}$ is a* relational critical region *(RCR) iff $\mu(\rho)$ is greater than a predefined threshold θ.*

LAMP takes demonstrations of simple tasks in the form of kinematic-state trajectories and converts them into trajectories of relative poses between object pairs. These trajectories are clustered into pairwise occupancy matrices between object-type pairs (τ_i, τ_j), and Def. 2 is applied to identify clusters with high criticality. LAMP then applies the `label`[1] function from the OpenCV package to extract connected components from each occupancy matrix. Let Γ denote the set of all connected components, and $\Gamma_{ij} \subset \Gamma$ the subset associated with object-type pair (τ_i, τ_j).

From each connected component $\gamma \in \Gamma$, LAMP uniformly samples poses and fits a Gaussian mixture model (GMM) using `GaussianMixture`[2] from scikit-learn, parameterized by $(\mu_\gamma, \Sigma_\gamma)$. The number of Gaussian components is chosen according to the number of connected components within γ. Each resulting mixture model defines a relational critical region (RCR) ψ_γ. A relative pose $p_{o_1}^{o_2}$ is classified into ψ_γ (i.e., $\psi_\gamma(p_{o_1}^{o_2}) = 1$) iff its likelihood under $(\mu_\gamma, \Sigma_\gamma)$ exceeds a threshold ϵ. For brevity, we omit the subscript γ when clear from context.

Once trained, these mixture models act as generative predictors, enabling zero-shot inference of relational critical regions in novel environments, given the object configurations and occupancy matrix of the environment.

We now discuss our approach for learning symbolic relational vocabulary from the predicted RCRs.

3.2 Inventing Semantically Well-Founded Concepts for Logic

Relational critical regions represent salient regions in the environment. However, they are insufficient for generalizable long-horizon reasoning; high-level reasoning requires abstract actions, and a relational vocabulary for expressing the pre- and post-conditions of these actions. Therefore, for each relational critical region predicted by the learned multivariate Gaussian predictors between a pair of objects, LAMP's *Relation Inventor* (Alg. 2, App. C) creates a unique binary relation between the corresponding object types. E.g., the RCR shown in Fig. 2(a) constitutes the extent of a newly invented relation (a concept equivalent to `Holding(Gripper, Can)`), and it is true when the gripper is in the shaded red region around the can.

Formally, let $\mathcal{O}_{\tau_i}$ and $\mathcal{O}_{\tau_j}$ be the set of objects of type τ_i and τ_j, respectively, and let Ψ_{ij} be a set of RCR predictors between $\mathcal{O}_{\tau_i}$ and $\mathcal{O}_{\tau_j}$. The Relation Inventor defines a unique binary relation $R_{ij}^k : \mathcal{O}_{\tau_i} \times \mathcal{O}_{\tau_j} \to \{\top, \bot\}$ for each relation region predictor $\psi^k \in \Psi_{ij}$ such that $R_{ij}^k(o_i, o_j) = \top$ iff for the relative pose $p_{o_i}^{o_j}$, $\psi^k(p_{o_i}^{o_j}) = 1$, where $p_{o_i}^{o_j}$ is the pose of the object o_i relative to the o_j.

Next, the Relation Inventor (RI) defines two additional sets of Boolean relations. First, given the sets of objects $\mathcal{O}_{\tau_i}$ and $\mathcal{O}_{\tau_j}$, it defines a relation $R'_{ij} : \mathcal{O}_{\tau_i} \times \mathcal{O}_{\tau_j} \to \{\top, \bot\}$ such that $R'_{ij}(o_i, o_j) \iff \forall k \left[\neg R_{ij}^k(o_i, o_j)\right]$. Second, it defines a relation for each relational critical region predictor indicating whether multiple objects can occupy that RCR or not. Intuitively, this models the free volume in the predicted region. E.g., the RCR for gripper w.r.t. can (Fig. 2(b), red) can be only occupied by a single can but the RCR for the placed cans w.r.t. table (Fig. 2(e), yellow) can be occupied by multiple cans. Formally, given the sets of objects $\mathcal{O}_{\tau_i}$ and $\mathcal{O}_{\tau_j}$, relational regions predictors Ψ_{ij}, the RI defines a Boolean relation for each relational region predictor $\psi^k \in \Psi_{ij}$, $R_{ij}^{free_k} : \mathcal{O}_{\tau_i} \to \{\top, \bot\}$

[1] https://github.com/opencv/opencv-python
[2] https://scikit-learn.org/1.5/modules/mixture.html

such that it is true iff for $o_i \in \mathcal{O}_{\tau_i}$ and $o_j \in \mathcal{O}_{\tau_j}$, $\rho_{free}(\psi^k, o_i) > \rho(o_j)$. Here, $\rho_{free}(\psi^k)$ is the free volume of the predicted region ψ^k and $\rho(o_j)$, the volume of the object o_j.

Given a new task T and its set of objects $\mathcal{O}_T$, LAMP uses the automatically learned relational region predictors to predict the RCRs for objects and generate the relational vocabulary $\mathcal{V}_T$ for the new task.

We now discuss our approach for autonomously inventing symbolic actions and world models.

3.3 Learning High-Level Actions and Logical World Models from Raw Data

The last step in the overall LAMP algorithm is to synthesize generalizable and transferrable actions, models, and action interpreters. Recall that we hypothesized that high-level actions are transitions to and from RCRs. E.g., the transitions in and out from the red RCR in Fig. 2(a) induce high-level actions `Pickup(Gripper, Object)` and `Place(Gripper, Object)` respectively. Therefore, LAMP's *High-Level Action Inventor (HLAI)* (App. D) use the predicted relational critical regions in training tasks to invent high-level actions. However, using the exhaustive enumeration of the set of predicted regions may lead to a large number of actions, most of which may be infeasible to realize. Instead, HLAI creates high-level actions corresponding to transitions between abstract states in the input training demonstrations, ensuring that all actions are executable by the robot.

More precisely, HLAI uses the invented relational vocabulary $\mathcal{V}$ to first convert each state trajectory $\langle x_0, \dots x_n \rangle$ of a demonstration to abstract states $\langle s'_0, \dots, s'_n \rangle$, and then lifts it to lifted states $\langle s_0, \dots, s_n \rangle$ by replacing object identities with placeholder variables of the original object types. Given the relational vocabulary $\mathcal{V}$ invented by RI (Sec. 3.2), each abstract state $s'_i = \{R^k(o_i, o_j) | \forall o_i, o_j \in \mathcal{O}, \forall R^k \in \mathcal{V}, x \models R^k(o_i, o_j)\}$.

Next, for each lifted transition $C_{ij} = s_i \rightarrow s_j$, HLAI computes sets of added and deleted relations such that $C^+_{ij} = s_j \setminus s_i$ and $C^-_{ij} = s_i \setminus s_j$ and identifies the set of all transitions from the training demonstrations that induce the same $\langle C^+_{ij}, C^-_{ij} \rangle$. Each of these sets induces a high-level action $\bar{a}_{ij}$. It repeats this process for every identified set and build the set of high-level actions $\bar{\mathcal{A}}$. HLAI then uses associative action model learning to learn symbolic models for these actions (App. D).

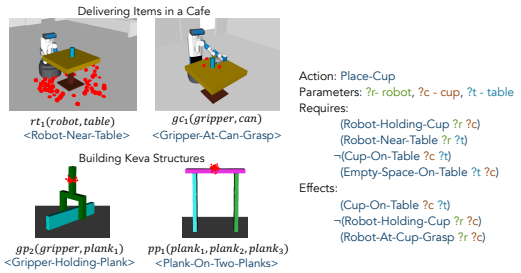


Figure 3: Examples of relations and an invented action with their critical regions. (a) Each image shows a binary predicate with its semantic interpretation; red dots indicate sampled poses within the relational critical region. (b) An auto-invented action is shown with parameters, preconditions, and effects. Relation names in blue reflect author-provided interpretations.

Each action invented in this manner can have spurious preconditions corresponding to static relations that do not change when the action is applied, but hold true in states where the action is executed. Therefore, the AI removes relations from the learned precondition that (i) are not parameterized by any of the objects that are changed by the action and (ii) that are not changed at any point in any of the demonstrations. This removes spurious preconditions with respect to the observed data.

To execute learned symbolic actions, robots require interpreters that convert abstract actions into sequences of primitive actions. LAMP autonomously constructs these interpreters using learned multivariate Gaussian predictors, whose generative properties enable sampling feasible configurations. Formally, for a grounded symbolic relation R', let $\Gamma_{R'} = \{x \in \mathcal{X} \mid R'(x) = 1\}$ denote its interpretation region. Then, the interpreter for an abstract action $\bar{a}'$ with effects $R'^1 \wedge \dots \wedge R'^k$ is defined as $\Gamma_{\bar{a}'} = \Gamma_{R'^1} \cap \dots \cap \Gamma_{R'^k}$. We build on STAMP [7] for task and motion planning using learned representations. We mitigate inaccuracies in learning using predicate abstraction [23] and a top-k planner [24]. App. F discusses this in detail.

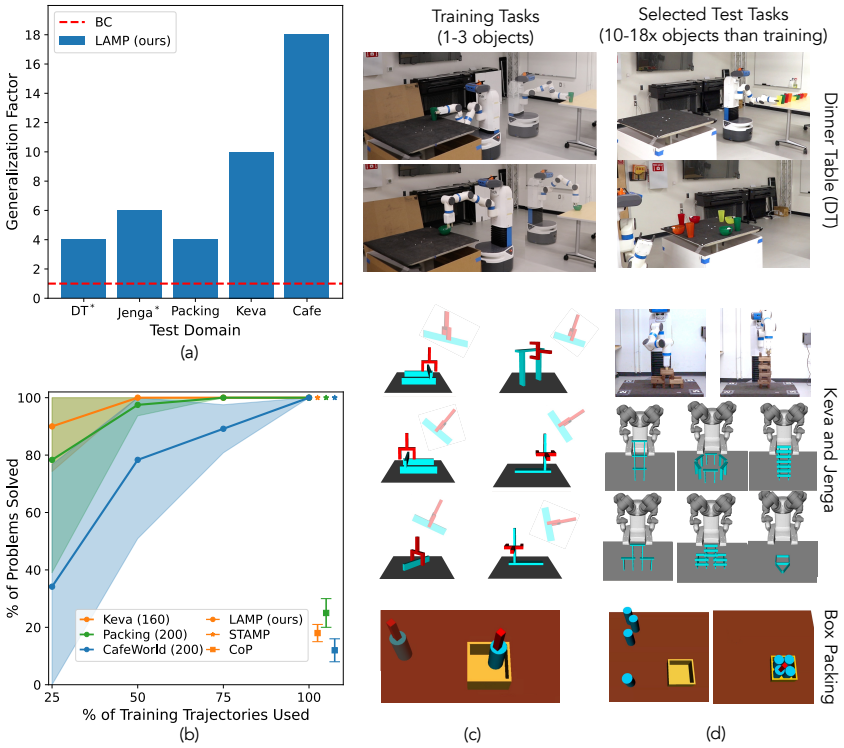


Figure 4: Empirical evaluation of LAMP. (a) Generalization achieved by our approach, with the x-axis denoting domain and the y-axis the generalization factor. The red dotted line marks the 1× generalization zone typical of imitation learning or behavior cloning. (b) Robustness comparison with Code-as-Policies and STAMP, where the x-axis shows the number of training demonstrations and the y-axis the proportion of solved test tasks; shaded regions indicate standard deviation over 10 runs. (c) Training tasks used to learn symbolic world models. (d) Test tasks used to evaluate generalization.

4 Empirical Evaluation

We evaluated LAMP across five realistic settings in simulation and the real world: (i) packing cans into a small box (**Box Packing**, 200 demos, one can); (ii) assembling free-standing structures with Keva planks (**Keva**, 160 demos, up to three planks); (iii) stacking Jenga planks (**Jenga**, 160 demos, up to three planks) using YuMi and Fetch robots; (iv) autonomous food delivery in a café (**Café**, 200 demos, one item); and (v) setting a dining table with cups and bowls (**Dinner Table**, 200 demos, one object). Across all domains, LAMP trained on at most 200 demonstrations of simpler tasks–of which only half were successful.

We show that the robot can invent concepts that generalize to larger, more complex tasks unseen during training—in simulation (Café, Keva, Box Packing) and in the real world (Dinner Table, Jenga). Figure 3 illustrates examples of these invented relations and a high-level action.

Goal specification Our implementation supports a variety of goal specification methods: (i) In tasks where goal poses are not anchored to a world-frame (e.g., Keva, Jenga) the system is given a goal structure that it automatically transcribes in terms of learned high-level predicates; (ii) For tasks where goal poses are broader regions (e.g., Café, Box Packing) the goal is specified manually using invented predicates; (iii) For tasks where the goal requires specific world-frame poses (e.g., Dinner Table) the goal is defined using `AtGoal(x)` predicates which are defined to be true iff object `x` is in its goal location.

Learning beyond imitation: generalization factors We define the *generalization factor* as the ratio of the maximum number of objects in test tasks to the maximum number of objects in training tasks. Since the number of relevant objects characterizes the size of the state space $\mathcal{X}$ that the robot must plan over, this is an effective proxy for the complexity of a task; it is directly correlated with the planning horizon, i.e., the number of decision-making steps that the robot needs to consider. We used a maximum of three objects in demonstrations across all test problems.

Fig. 4(a) shows the generalization factors for LAMP (blue bar) compared to the zone of generalization for contemporary approaches that use expert demonstrations for imitation (red line), computed using 12 distinct test tasks for each domain with varying numbers of objects or problem settings. Contemporary imitation learning approaches use expert demonstrations to learn to imitate and therefore only perform tasks that were trained on, leading to a generalization factor of 1. On the other hand, LAMP achieves a generalization factor of upto 18. In the café domain, LAMP was able to solve problems with 18 (8) objects in simulation (real-worlxd) based on demonstrations with 1 (simulated) or 2 (real-world) objects, yielding a generalization factor of $18\times$ ($4\times$) in simulated (real-world) tasks. Similarly, for Keva and Jenga domains, LAMP invented its world models from demonstrations with three planks and successfully built structures with 30 Keva planks in simulated settings (generalization factor of $10\times$) using the YuMi robot and 18 Jenga planks in real-world settings (generalization factor of $6\times$) using the Fetch robot. The maximum possible generalization factor for box packing was $4\times$ as the box can only accommodate four cans, and LAMP achieved this generalization factor.

Zero-shot transfer: LAMP vs. symbolic and foundation model baselines To evaluate transferability, we test LAMP on new tasks with no prior knowledge–using only demonstrations from simpler tasks with fewer objects (Fig. 4(b)–and compare its performance to robots with hand-crafted models and with foundation model–based reasoning [25, 26, 11, 13]. We used prior work on stochastic task and motion planning (STAMP) [7] as a baseline approach for the former and Code as Policies (CoP) [11] for the latter approach. LAMP solved 100% of the unseen test tasks in our evaluation reaching similar performance of STAMP, which used expert-provided abstractions. CoP solved only the simpler tasks, with under 35% overall success. We remark that although popular, these baselines solve fundamentally simpler problems as they require manually crafted inputs for the concepts our agent needs to invent on its own: STAMP requires complete world models, while CoP requires Python APIs for actions, interpreters, detailed goal descriptions with hand-crafted predicates, and sample solution code. Additionally, CoP was given 25 attempts per problem with a single successful execution required for the problem considered solved (pass @25).

Sample efficiency and robustness Sample efficiency is a critical factor in robot learning. Modern approaches typically require tens of thousands of demonstrations, which can be difficult to obtain due to the substantially higher cost of obtaining demonstrations in robotics as compared to learning in other domains such as text and images. We observed that enabling the robot to learn generalizable concepts substantially reduces the burden of facilitating demonstrations. Figure 4(b) shows how this approach scales as the number of demonstrations is reduced further, illustrating that LAMP can effectively learn effective world models using as few as 40 goal-achieving demonstrations.

5 Conclusion

This paper presents the first approach for learning symbolic concepts and world models directly from kinematic demonstration trajectories, enabling generalization across robots and unseen problem settings. Extensive evaluation in simulated and real-world settings shows that the learned abstractions are both efficient and interpretable. Future work will leverage these autonomously learned abstractions to make robot re-tasking accessible to non-experts. We also aim to extend our approach to handle stochastic environments with stronger theoretical guarantees.

6 Limitations

Despite the substantial gains in generalization and sample efficiency, further work is needed on problems not addressed in this work. First, the current formulation assumes near-perfect kinematic state estimation during training and deployment. Learning good state estimators is an active and orthogonal direction of research (e.g., using point cloud data [27]). We handle these limitations as follows: (i) By using training data collected in a simulator with perfect state estimation, and (ii) by using a motion capture system at test time. However, reliably bridging this gap will require tighter integration with robust state-estimation pipelines or explicit treatment of perceptual uncertainty inside the abstraction-learning loop.

Second, we restrict the scope of this paper to deterministic world models. However, most real-world settings are stochastic and require stochastic world models for safe and reliable planning [7]. This foundational work presents a step in the direction of learning more general world models for stochastic, and eventually, partially observable settings.

Third, in this paper, we limit our experiments of generalization to novel goals that use the same known object types. This includes scenarios with additional instances of these objects and with new positions and orientations. Future work will extend our approach to generalize learned concepts to entirely new object categories and novel geometries.

Finally, our approach is currently restricted to learning actions where dynamics do not play a significant role and the quasi-static assumption of motion planning is valid. Tasks requiring force-controlled contact, or closed-loop reactive policies lie outside the scope of the current work and constitute promising directions for research.

Acknowledgments

We thank the CoRL reviewers for their useful and constructive feedback. We thank Pulkit Verma for his help with the preliminary version of the presented work. We also thank Prof. George Konidaris with his helpful suggestions and discussions. The work was partially funded by NSF under the grants IIS 2451108 and IIS 1942856, and by ONR under the grant N00014-23-1-2416.

References

[1] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[2] Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *Conference on Robot Learning (CoRL)*, 2024.

[3] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K.-H. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. Ryoo, G. Salazar, P. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. RT-1: Robotics transformer for real-world control at scale. *arXiv:2212.06817*, 2022.

[4] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, S. K. S. Ghasemipour, C. Finn, and A. Wahid. Aloha unleashed: A simple recipe for robot dexterity. In *Proc. CoRL*, 2024.

[5] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[6] J. Barreiros, A. Beaulieu, A. Bhat, R. Cory, E. Cousineau, H. Dai, C.-H. Fang, K. Hashimoto, M. Z. Irshad, M. Itkina, et al. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025.

[7] N. Shah, D. K. Vasudevan, K. Kumar, P. Kamojjhala, and S. Srivastava. Anytime integrated task and motion policies for stochastic environments. In *Proc. ICRA*, 2020.

[8] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous systems*, 4:265–293, 2021.

[9] N. Shah and S. Srivastava. Using deep learning to bootstrap abstractions for hierarchical robot planning. In *Proc. AAMAS*, 2022.

[10] N. Shah and S. Srivastava. Hierarchical planning and learning for robots in stochastic settings using zero-shot option invention. In *Proc. AAAI*, 2024.

[11] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *Proc. ICRA*, 2023.

[12] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv:2304.11477*, 2023.

[13] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. PaLM-E: An Embodied Multimodal Language Model. In *Proc. ICML*, 2023.

[14] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as I can, not as I say: Grounding language in robotic affordances. In *Proc. CoRL*, 2023.

[15] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *JAIR*, 61:215–289, 2018.

[16] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *Proc. CoRL*, 2022.

[17] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. P. Kaelbling, and J. Tenenbaum. Predicate invention for bilevel planning. In *Proc. AAAI*, 2023.

[18] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. S. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[19] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research*, 28(1):104–126, 2009.

[20] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. ICRA*, 2014.

[21] D. Molina, K. Kumar, and S. Srivastava. Learn and link: Learning critical regions for efficient planning. In *Proc. ICRA*, 2020.

[22] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006. ISBN 0521862051.

[23] S. Graf and H. Saidi. Construction of abstract state graphs with pvs. In *Proc. CAV*, 1997.

[24] M. Katz, S. Sohrabi, O. Udrea, and D. Winterer. A novel iterative approach to top-k planning. In *Proc. ICAPS*, 2018.

[25] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. SayPlan: Grounding large language models using 3D scene graphs for scalable robot task planning. In *Proc. CoRL*, 2023.

[26] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia. Language to rewards for robotic skill synthesis. In *Proc. CoRL*, 2023.

[27] B. Wen, W. Yang, J. Kautz, and S. Birchfield. Foundationpose: Unified 6d pose estimation and tracking of novel objects. In *Proc. CVPR*, 2024.

[28] X. Wang. Learning planning operators by observation and practice. In *Proc. AIPS*, 1994.

[29] R. Stern and B. Juba. Efficient, safe, and probably approximately complete learning of action models. In *Proc. IJCAI*, 2017.

[30] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. An incremental constraint-based framework for task and motion planning. *IJRR*, 37(10):1134–1151, 2018.

[31] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proc. ICAPS*, 2020.

[32] U. A. Mishra, S. Xue, Y. Chen, and D. Xu. Generative skill chaining: Long-horizon skill planning with diffusion models. In *Proc. CoRL*, 2023.

[33] S. Cheng, C. Garrett, A. Mandlekar, and D. Xu. NOD-TAMP: Multi-step manipulation planning with neural object descriptors. In *CoRL 2023 LEAP Workshop*, 2023.

[34] X. Fang, C. R. Garrett, C. Eppner, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox. DiMSam: Diffusion models as samplers for task and motion planning under partial observability. In *CoRL 2023 LEAP Workshop*, 2023.

[35] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez. Constructing symbolic representations for high-level planning. In *Proc. AAAI*, 2014.

[36] E. Ugur and J. Piater. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *Proc. ICRA*, 2015.

[37] G. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez. Symbol acquisition for probabilistic high-level planning. In *Proc. IJCAI*, 2015.

[38] G. Andersen and G. Konidaris. Active exploration for learning symbolic representations. In *Proc. NeurIPS*, 2017.

[39] B. Bonet and H. Geffner. Learning first-order symbolic representations for planning from the structure of the state space. In *Proc. ECAI*, 2019.

[40] S. James, B. Rosman, and G. Konidaris. Learning portable representations for high-level planning. In *Proc. ICML*, 2020.

[41] A. Ahmetoglu, M. Y. Seker, J. Piater, E. Oztop, and E. Ugur. DeepSym: Deep symbol generation and rule learning for planning from unsupervised robot interaction. *JAIR*, 75:709–745, 2022.

[42] M. Asai, H. Kajino, A. Fukunaga, and C. Muise. Classical planning in deep latent space. *JAIR*, 74:1599–1686, 2022.

[43] J. Liang and A. Boularias. Learning category-level manipulation tasks from point clouds with dynamic graph cnns. In *Proc. ICRA*, 2023.

[44] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted MAX-SAT. *AIJ*, 171(2-3):107–143, 2007.

[45] S. Cresswell, T. McCluskey, and M. West. Acquisition of object-centred domain models from planning examples. In *Proc. ICAPS*, 2009.

[46] H. H. Zhuo and S. Kambhampati. Action-model acquisition from noisy plan traces. In *Proc. IJCAI*, 2013.

[47] D. Aineto, S. J. Celorrio, and E. Onaindia. Learning action models with minimal observability. *AIJ*, 275:104–137, 2019.

[48] P. Verma, S. R. Marpally, and S. Srivastava. Asking the right questions: Learning interpretable action models through query answering. In *Proc. AAAI*, 2021.

[49] M. Čertický. Real-Time Action Model Learning with Online Algorithm 3SG. *Applied AI*, 28 (7):690–711, Aug. 2014.

[50] L. Lamanna, A. Saetti, L. Serafini, A. Gerevini, and P. Traverso. Online Learning of Action Models for PDDL Planning. In *Proc. IJCAI*, 2021.

[51] D. Bryce, J. Benton, and M. W. Boldt. Maintaining evolving domain models. In *Proc. IJCAI*, 2016.

[52] R. K. Nayyar, P. Verma, and S. Srivastava. Differential assessment of black-box AI agents. In *Proc. AAAI*, 2022.

[53] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. Learning symbolic operators for task and motion planning. In *Proc. IROS*, 2021.

[54] P. Verma, S. R. Marpally, and S. Srivastava. Discovering user-interpretable capabilities of black-box planning agents. In *Proc. KR*, 2022.

[55] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *Proc. IROS*, 2022.

[56] 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 *Proc. CoRL*, 2023.

[57] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Proc. NeurIPS*, 1988.

[58] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[59] A. Goyal, J. Xu, Y. Guo, V. Blukis, Y.-W. Chao, and D. Fox. RVT: Robotic view transformer for 3D object manipulation. In *Proc. CoRL*, 2023.

[60] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proc. CoRL*, 2023.

[61] Q. Vuong, S. Levine, H. R. Walke, K. Pertsch, A. Singh, R. Doshi, C. Xu, J. Luo, L. Tan, D. Shah, C. Finn, M. Du, M. J. Kim, A. Khazatsky, J. H. Yang, T. Z. Zhao, K. Goldberg, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. In *CoRL 2023 TGR Workshop*, 2023.

[62] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proc. ICML*, 2022.

[63] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2Motion: From natural language instructions to feasible plans. *Autonomous Robots*, Nov 2023.

[64] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proc. CoRL*, 2023.

[65] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati. On the planning abilities of large language models–A critical investigation. In *Proc. NeurIPS*, 2023.

Appendix

A Formal Framework

We consider a setting where the environment comprises objects and robots. Each object's state is represented by a 6D pose. A robot, considered a distinct object, is structured as a kinematic chain of links and joints, and its state is $\langle P_{\text{base}}, \Theta \rangle$, where P_{base} indicates the 6D pose of the initial link, and Θ corresponds to the values for each joint. In an environment with objects $\mathcal{O} = \langle o_1, \dots o_n, r_1, \dots r_m \rangle$, the state space is denoted by $\mathcal{X} = \mathcal{X}_{r_i} \times \mathcal{X}_{o_j}$ for every robot $r_i \in \mathcal{O}$ and object $o_j \in \mathcal{O}$. A collision function c categorizes the state space $\mathcal{X}$ into two subsets: $\mathcal{X}_{free}$ (states without collisions) and $\mathcal{X}_{obs}$ (colliding states).

Primitive robot actions allow robots to alter their state, including configuration and base link pose, enabling movement and object manipulation within the environment. A primitive action a is characterized by a deterministic function $a : x \mapsto x'$. Given environment states $x \in \mathcal{X}$ and $x' \in \mathcal{X}$, taking action a in state x leads to state x'. We describe a robot planning problem as follows:

Definition 3 *A **robot planning domain** is characterized as a tuple $\langle \mathcal{O}, \mathcal{T}, \mathcal{X}, \mathcal{A}, x_i, \mathcal{X}_g \rangle$ where $\mathcal{O}$ represents a collection of objects, $\mathcal{T}$ is a set of object types, $\mathcal{X}$ denotes a state space, and $\mathcal{A}$ is an uncountably infinite set of deterministic native actions. The initial state of the environment, $x_i \in \mathcal{X}_{\text{free}}$ is an initial state of the environment, and $\mathcal{X}_g \subseteq \mathcal{X}$ is a set of goal states. Solving a planning problem involves finding a sequence of native actions $a_0, \dots, a_n$ such that $a_n(\dots(a_0(x_i))) \in \mathcal{X}_g$.*

Our approach extensively uses *relative poses*. Every object in the environment also defines a frame of reference. A relative pose defines the pose of an object in the reference frame of another object. Basis transformations from linear algebra can be used to compute relative transformations of objects w.r.t to other objects in the environment. We refer to the pose of an object o_1 relative to an object o_2 as $P_{o_1}^{o_2}$. Let $\tilde{\mathcal{X}}_{o_1}^{o_2}$ define a relative state-space for the pair of objects o_1 and o_2, i.e., the set of all poses of the object o_1 in the relative frame of the object o_2, and $\tilde{\mathcal{X}}$ define the set of relative state spaces such that $\tilde{\mathcal{X}} = \{\tilde{\mathcal{X}}_{o_j}^{o_i} | o_i, o_j \in \mathcal{O} \wedge o_i \neq o_j\}$. Lastly, we define a transformation function $\xi : \mathcal{X} \to \tilde{\mathcal{X}}$ that computes the relative state for each absolute state of the environment.

Symbolic World Models We treat symbolic world models as first-order logic frameworks and use PDDL [18] to express these models. The PDDL domain encompasses two key components: $\langle \mathcal{V}, \bar{\mathcal{A}} \rangle$. Here, $\mathcal{V}$ represents a collection of symbolic relationships, while $\bar{\mathcal{A}}$ comprises high-level actions performed by robots. The relations $R \in \mathcal{V}$, defined by typed parameters, establish how objects of one type relate to one another type. These relations $R \in \mathcal{V}$ can be instantiated with specific objects, termed R when uninstantiated and R' when instantiated. Instantiated relations R' serve as Boolean classifiers, meaning they are true in a low-level state x (indicated by $R'_x = 1$) if the relation is valid for the objects in question within the state x. The abstraction function $\alpha : x \mapsto s$ evaluates all instantiated relations in a low-level state $x \in \mathcal{X}$, resulting in an abstract grounded state $s' \in 2^{\mathcal{V}'}$ (or $s \in \alpha(\mathcal{X})$). This symbolic grounded state s' comprises the relations that hold true in low-level state x, whereas the symbolic lifted state is denoted by s ($s \in 2^{\mathcal{V}}$).

$\bar{\mathcal{A}}$ outlines symbolic lifted actions using lifted relations $\mathcal{V}$. Each action $\bar{a} \in \bar{\mathcal{A}}$ has typed symbolic parameters. $\bar{a}$ is defined as a tuple $\langle \mathit{pre}_{\bar{a}}, \mathit{eff}_{\bar{a}} \rangle$, where $\mathit{pre}_{\bar{a}}$ is a conjunctive formula of parameterized relations $\mathcal{V}$. The action's effect eff_a is a tuple $\mathit{eff}_{\bar{a}} = \langle \mathit{add}_{\bar{a}}, \mathit{del}_{\bar{a}} \rangle$ adding relations $\mathit{add}_{\bar{a}}$ and removing relations $\mathit{del}_{\bar{a}}$ from the state once the action $\bar{a}$ is executed. Actions $\bar{a}$ can be grounded to specific objects, resulting in grounded actions $\bar{a}'$, generating grounded precondition $\mathit{pre}_{\bar{a}'}$ and effect $\mathit{eff}_{\bar{a}'}$. A grounded action $\bar{a}'$ is applicable in a state s only if $\mathit{pre}_{\bar{a}'} \models s$. Every deterministic grounded action $\bar{a}' \in \bar{\mathcal{A}}'$ maps each symbolic state s_i to a new state s_j.

The set $\bar{\mathcal{A}}$ delineates symbolic lifted actions utilizing the lifted relations $\mathcal{V}$. Each action $\bar{a} \in \bar{\mathcal{A}}$ has typed symbolic parameters and is defined using a tuple $\langle \mathit{pre}_{\bar{a}}, \mathit{eff}_{\bar{a}} \rangle$. Here, $\mathit{pre}_{\bar{a}}$ represents a precondition of the action $\bar{a}$ as a conjunction of parameterized relations $\mathcal{V}$. The effect of the action, eff_a, is detailed as $\mathit{eff}_{\bar{a}} = \langle \mathit{add}_{\bar{a}}, \mathit{del}_{\bar{a}} \rangle$, which involves adding $\mathit{add}_{\bar{a}}$ and removing $\mathit{del}_{\bar{a}}$ relations from the state upon execution of action $\bar{a}$. These actions $\bar{a}$ can be instantiated with specific objects

to obtain grounded actions $\bar{a}'$, thus producing a grounded precondition $pre_{\bar{a}'}$ and effect $eff_{\bar{a}'}$. An instantiated action $\bar{a}'$ applies in a state s' only if the precondition $pre_{\bar{a}'} \models s'$. Each deterministic grounded action $\bar{a}' \in \bar{\mathcal{A}}'$ defines a deterministic function $a' : s'_i \mapsto s'_j$ that transitions a symbolic state s'_i to another state s'_j.

Symbolic plans cannot be executed by a robot. It needs to be converted to a sequence of primitive actions that a robot can execute. Task and motion planning approaches use abstract symbolic models along with *pose generators* for computing a sequence of primitive actions for planning problems. A pose generator defines an inverse abstraction function. Let γ_p be a pose generator for a lifted symbolic predicate $p \in \mathcal{P}$. For a grounded predicate p', a pose generator $\gamma_{p'} = \{x | x \in \mathcal{X} \wedge p'_x = 1\}$. A pose generator for a grounded state s' is defined as $\bigcap_{\forall p' \in s'} \gamma_{p'}$.

B Overview of the LAMP

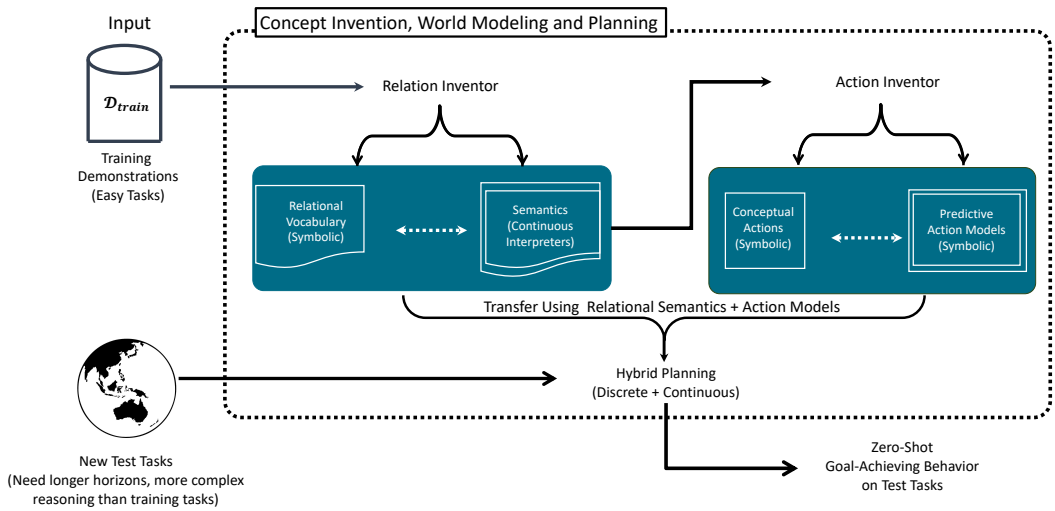


Figure 5: Overview of the proposed algorithm. The approach comprises of two core components: (i) "Relation Inventor" that uses training demonstrations and generates novel relational vocabularies for robots, and (ii) "Action Inventor" that invents high-level symbolic actions. Together, they enable zero-shot goal-achieving behavior by integrating relational semantics, predictive action models, and hybrid planning.

Algorithm 1: LAMP: Learning Abstract Model for Planning

Input: A set of demonstrations $\mathcal{D}_{train}$ for training tasks T_{train}, a set of objects $\mathcal{O}$, a set of types of objects $\mathcal{T}$, test tasks T_{test}

Output: Symbolic world model $\mathcal{M}$

```
/* Use Alg. 2 to invent relations */
```

$\mathcal{V} \leftarrow$ Relation_Inventor($\mathcal{D}_{train}$);

```
/* Use Alg. 3 to invent actions */
```

$\bar{\mathcal{A}} \leftarrow$ Action_Inventor($\mathcal{D}_{train}, \mathcal{V}$);

$\mathcal{M} = \langle \mathcal{V}, \bar{\mathcal{A}} \rangle$;

```
/* Solve new unseen task with task and motion planning (App. F) */
```

$\Pi_{\mathcal{D}_{\text{test}}} \leftarrow$ task_and_motion_planning($T_{test}, \mathcal{M}$);

return $\mathcal{M}$;

C Relation Inventor

Algorithm 2: Relation Inventor

Input: Training Demonstrations $\mathcal{D}_{train}$, set of objects $\mathcal{O}$, set of object types $\mathcal{T}$
Output: Set of relations $\mathcal{V}$

```
/* Prepare data                                                    */
```
$\tilde{\mathcal{D}}_{train} \leftarrow \xi(\mathcal{D}_{train}, \mathcal{O})$
```
/* Learn relational critical region predictor                      */
```
$\Psi \leftarrow$ learn_relational_critical_regions_predictor($\tilde{\mathcal{D}}_{train}$)
```
/* Identify binary relations                                       */
```
$\mathcal{V}_{bin} \leftarrow$ invent_binary_relations(Ψ)
```
/* Identify additional relations                                   */
```
$\mathcal{V}_{add} \leftarrow$ invent_additional_relations($\mathcal{V}_{bin}, \Psi$)
$\mathcal{V} \leftarrow \mathcal{V}_{bin} \cup \mathcal{V}_{add}$
return $\mathcal{V}$

D Action Inventor

Algorithm 3: Inventing Symbolic Actions

Input: Set of demonstrations $\mathcal{D}_{train}$, learned relations $\mathcal{V}$
Output: Set of lifted actions $\bar{\mathcal{A}}$

$\bar{\mathcal{D}}'_{train} \leftarrow$ get_abstract_demonstrations($\mathcal{D}_{train}$,$\mathcal{V}$);
$\bar{\mathcal{D}}_{train} \leftarrow$ lift_demonstrations($\bar{\mathcal{D}}'_{train}$);
changed_predicates $\leftarrow$ [];
foreach $d^k \in \bar{\mathcal{D}}$ **do**
  foreach *consecutive state* $s_i, s_j \in d^k$ **do**
    ${}^{+}C^k_{ij} \leftarrow s_j \setminus s_i$; ${}^{-}C^k_{ij} \leftarrow s_i \setminus s_j$;
    $C^k_{ij} \leftarrow \langle {}^{+}C^k_{ij}, {}^{-}C^k_{ij} \rangle$;
    changed_predicates.add(C^k_{ij});
$\mathcal{C} \leftarrow$ create_clusters($\bar{\mathcal{D}}_{train}$, changed_predicates);
$\bar{\mathcal{A}} \leftarrow$ [];
foreach $(S_i \rightarrow S_j), C \in \mathcal{C}$ **do**
  eff $\leftarrow \langle$*add* = ${}^{+}C$, *del* = ${}^{-}C \rangle$;
  pre $\leftarrow \cap_{s \in S_i} s$;
  pre $\leftarrow$prune_precondition(*pre*);
  param $\leftarrow$ extract_params($S_i \rightarrow S_j$);
  $\bar{\mathcal{A}}$.add(create_action(*param*, *pre*, *eff*));
return $\bar{\mathcal{A}}$

After identifying a group of high-level actions denoted by $\bar{\mathcal{A}}$ we employ associative learning along with the training demonstrations $\mathcal{D}_{train}$ to develop a symbolic representation for each action $\bar{a} \in \bar{\mathcal{A}}$. The symbolic model of an action is described through its symbolic effects, symbolic preconditions, and parameters.

Learning effects As noted earlier, in our setting, effect of an action $\bar{a}$ is represented as $\mathit{eff}_{\bar{a}} = \langle \mathit{add}_{\bar{a}}, \mathit{del}_{\bar{a}} \rangle$. Each cluster $c_i \in \mathcal{C}$ is generated by clustering transitions the sets of changed relations. These changed relations correspond to added and removed relations as an effect of executing the action induced by the cluster. Therefore, for an action $\bar{a}_i$ induced by the cluster c_i with a set of changed relations $C_i = \langle {}^{+}C_i, {}^{-}C_i \rangle$, $\mathit{add}_{\bar{a}_i} = {}^{+}C_i$ and $\mathit{del}_{\bar{a}_i} = {}^{-}C_i$.

Learning preconditions To learn the precondition of an action, we take the intersection of all states where the action is applicable. Given a set of relations, this approach generates a maximal

precondition that is conservative yet sound [28, 29]. We learn the precondition of an action $\bar{a} \in \bar{\mathcal{A}}$ corresponding to a cluster $c = \langle S_i \rightarrow S_j, C_{ij} \rangle$ $pre_{\bar{a}} = \cap_{s \in S_i} s$.

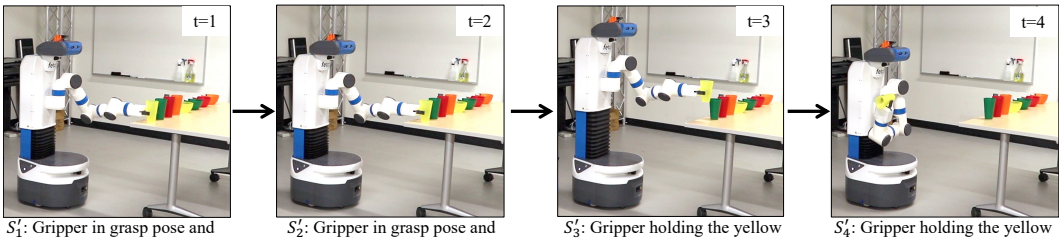
Each action can have spurious preconditions corresponding to static relations that do not change when the action is applied, but are still true in all the pre-states. Therefore, we remove relations from the learned precondition that (i) are not parameterized by any of the objects that are changed by the action and (ii) are not changed at any point in any of the demonstrations. This removes any predicate from the precondition that is spurious with respect to the data.

Learning parameters Once the precondition and effect of an action are learned, the final step is to learn the parameters of the action that can be replaced with objects in order to ground the action. In this step, the relations in precondition and effect are processed in order. These relations are processed in alphanumeric order and each of their parameters is added to the action's parameter list, if not already added. This process leads to an ordered list of parameters of the action, which can be grounded with compatible objects.

D.1 Example of Action Model Learning

Let the set of predicates invented in Sec. 3.2 be the following:

- `(table-can0 ?table ?can)`: Can is not on the table.
- `(table-can1 ?table ?can)`: Can is on the table.
- `(can-gripper1 ?can ?gripper)`: Gripper is at grasp pose (not holding/grasping yet).
- `(can-gripper2 ?can ?gripper)`: Gripper has grasped the object.
- `(base-gripper0 ?base ?gripper)`: Robot's base link and robot's gripper link does not have any relation.
- `(base-gripper1 ?base ?gripper)`: Robot's arm is tucked so there is a specific relative pose between the robot's base link and the robot's gripper link.
- `(base-table1 ?base ?table)`: Robot's base link is located in a way such that the robot's arm can reach objects on the table.

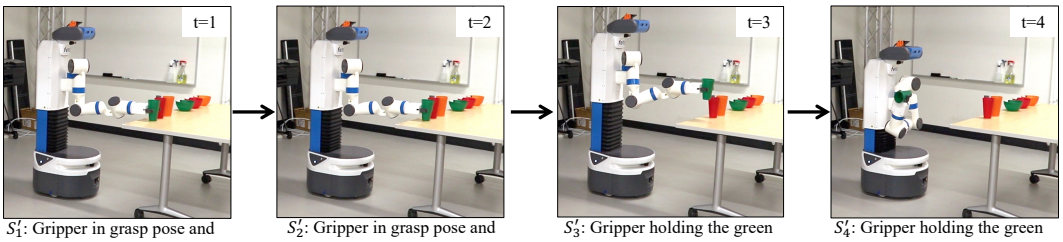


S'_1: Gripper in grasp pose and not holding the yellow cup. S'_2: Gripper in grasp pose and holding the yellow cup. S'_3: Gripper holding the yellow cup, which is not on table. S'_4: Gripper holding the yellow cup. Robot's arm is tucked.

Figure 6: Trajectory $T_1 = \langle S'_1, S'_2, S'_3, S'_4 \rangle$ corresponding to the process of picking up a yellow cup from the table. The state description below each image explains that image in English. These state descriptions are added here for ease of understanding only.

Now, consider the two trajectories T_1 and T_2 as shown in Fig. 6 and Fig. 7, respectively. Here T_1 corresponds to the Fetch robot picking a yellow cup, and T_2 corresponds to the robot picking up a green cup (kept at a different location on the table compared to that of the yellow cup). Here these two trajectories are expressed in terms of grounded objects. These are converted to a lifted form using line 2 of Alg. 3 in terms of the predicates shown earlier. For T_1 and T_2 both, the lifted states will be:

- S_1 : {`(table-can1 ?table ?can)`, `(can-gripper1 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.



S_1': Gripper in grasp pose and not holding the green cup. S_2': Gripper in grasp pose and holding the green cup. S_3': Gripper holding the green cup, which is not on table. S_4': Gripper holding the green cup. Robot's arm is tucked.

Figure 7: Trajectory $T_2 = \langle S_1', S_2', S_3', S_4' \rangle$ corresponding to the process of picking up a green cup from the table. The state description below each image explains that image in English. These state descriptions are added here for ease of understanding only.

- S_2 : {`(table-can1 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.
- S_3 : {`(table-can0 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.
- S_4 : {`(table-can0 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, `(base-gripper1 ?base ?gripper)`, `(base-table1 ?base ?table)`}.

Note that we only show partial states here for brevity. The actual states will also have predicates like `(table-can1 ?table ?can2)`, `(table-can1 ?table ?can3)`, `(table-can1 ?table ?can4)`, `(table-can1 ?table ?bowl1)`, `(table-can1 ?table ?bowl2)`, `(table-can1 ?table ?bowl3)`, etc. corresponding to other objects kept on the table.

Learning effects Alg. 3 creates the following three clusters (lines 4-9) based on these states.

- $C_{12} = \langle {}^{+}C_{12} =$ {`(can-gripper2 ?can ?gripper)`}, ${}^{-}C_{12} =$ {`(can-gripper1 ?can ?gripper)`}$\rangle$.
- $C_{23} = \langle {}^{+}C_{23} =$ {`(table-can0 ?table ?can)`}, ${}^{-}C_{23} =$ {`(table-can1 ?table ?can)`}$\rangle$.
- $C_{34} = \langle {}^{+}C_{34} =$ {`(base-gripper1 ?base ?gripper)`}, ${}^{-}C_{34} =$ {`(base-gripper0 ?base ?gripper)`}$\rangle$.

Learning preconditions Learning preconditions involve taking intersection of states in which all the actions in the same cluster were executed. Here S_1 to S_3 mentioned below will remain the same for the three clusters. E.g., precondition of C_{12} = {`(table-can1 ?table ?can)`, `(can-gripper1 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}. Alg. 3 will prune out `(base-table1 ?base ?table)` from the precondition as (i) it is unchanged between S_1 and S_2, and (ii) none of its parameters (`?base` and `?table`) are part of any other predicate that is changed. Using this, the precondition for each action will be:

- $pre(C_{12})$ = {`(table-can1 ?table ?can)`, `(can-gripper1 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`}.
- $pre(C_{23})$ = {`(table-can1 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.
- $pre(C_{34})$ = {`(table-can0 ?table ?can)`, `(can-gripper2 ?can ?gripper)`, `(base-gripper0 ?base ?gripper)`, `(base-table1 ?base ?table)`}.

Learning parameters Learning parameters from an action's precondition and effect is straightforward. All the unique parameters in predicates in the precondition and effect are added to the parameter list of an action representing a cluster. Using this notion, the parameters for the three clusters will be the following:

- $param(C_{12})$ =(?table ?can ?gripper ?base).
- $param(C_{23})$ =(?table ?can ?gripper ?base).
- $param(C_{34})$ =(?table ?can ?gripper ?base).

E Related Work

The presented approach directly relates to various concepts in task and motion planning, model learning, and abstraction learning. However, to the best of our knowledge, this is the first work that automatically invents generalizable symbolic predicates and high-level actions simultaneously using a set of low-level trajectories.

Task and motion planning Task and motion planning approaches [20, 30, 31, 7] develop approaches for autonomously solving long-horizon robot planning problems. These approaches are complementary to the presented approach as they focus on using provided abstractions for efficiently solving the robot planning problems. Shah and Srivastava [9, 10] learn state and action abstractions for long-horizon motion planning problems. Orthogonal research [32, 33, 34] learn implicit abstractions (action interpreters or abstract actions) for TAMP in the form of generative models. However, these approaches do not learn generalizable relational representations as well as complex high-level relations and actions which is the focus of our work.

Learning symbolic abstractions Several approaches invent symbolic vocabularies given a set of high-level actions (or skills) [35, 36, 37, 38, 15, 39, 40]. Ahmetoglu et al. [41], Asai et al. [42], Liang and Boularias [43] learn symbolic predicates in the form of latent spaces of deep neural networks and use them for high-level symbolic planning. However, these approaches assume high-level actions to be provided as input. On the other hand, the approach presented in this paper automatically learns high-level actions along with symbolic predicates.

Numerous approaches [44, 45, 46, 47, 48] have focused on learning preconditions and effects for high-level actions, i.e., action model. A few approaches [49, 50] have also focused on continually learning action models while collecting experience in the environment. Bryce et al. [51] and Nayyar et al. [52] focus on updating a known model using inconsistent observations. However, these approaches require a set of symbolic predicates and/or high-level action signatures as input whereas our approach automatically invents these predicates and actions. Several approaches [53, 54, 55, 16, 56, 17] have been able to automatically invent high-level actions that are induced by state abstraction akin to the presented approach. However, unlike our approach, these approaches do not automatically learn symbolic predicates and/or low-level samplers and require them as input.

Behavior Cloning for Robotics Behavior cloning (BC) has been widely explored in robotics as a method for learning control policies from expert demonstrations. Early work by Pomerleau [57] introduced the concept of imitation learning through supervised learning, where a model maps sensor inputs to control actions. Recent years have seen renewed interest in behavior cloning approaches due to the rise in applicability of deep neural networks [58, 3, 2, 4, 5]. These appraoches have shown remarkable advancements in dexterous manipulation capabilities (e.g., tying shoelaces and chopping fruits) and ability to manipulate deformable objects (e.g., folding laundry). However, these approaches are only limited to tackling problems in their training demonstrations as well as often require huge amounts of training demonstrations (in the order of 1000s of demonstrations per task), reducing the applicability of such approaches in real-world settings where the distribution of tasks is not known a priori.

LLMs for robot planning Recent years have also seen significantly increased interest in using foundational models such as LLM (large language model), VLM (visual language model), and transformers for robot planning and control owing to their success in other fields such as NLP, text generation, and vision. Several approaches [3, 59, 60, 61] use transformer architecture for learning

reactive policies for short-horizon robot control problems. Problems tackled by these approaches are analogous to individual actions learned by our approach.

Several directions of research explore the use of LLMs for utilizing LLMs as high-level planners to generate sequences comprising of high-level, expert crafted actions [26, 11, 62, 25, 63, 64, 14]. These methods make progress on the problem of near-natural language communication with robots and are complementary to the proposed work. However, there is a strong evidence against the soundness of LLMs as planners. Valmeekam et al. [65] show that LLMs are only $\sim 36\%$ accurate as planners even in simple block stacking settings not involving more than 5 object.

F Task and Motion Planning with Learned World Models

Task and motion planning [20, 31, 7] combines symbolic world representations and action interpreters to develop an interleaved planning approach. This approach seeks a valid sequence of high-level actions each accompanied by legitimate primitive action refinement. We apply a deterministic form of STAMP [7] for task and motion planning relying on learned world representations. STAMP utilizes the acquired relational vocabulary along with high-level robotic actions to generate a high-level action sequence, subsequently refining these actions using learned interpreters into executable primitive actions.

One of the major challenges in TAMP with automatically learned world models is the overly pessimistic or conservative nature of these models, often leading the planner to a failed search even in the presence of potential plans. To address this, we incorporate predicate abstractions [23] to systematically create a relaxed high-level planning problem that STAMP can work to refine. However, the use of predicate abstractions may result in inaccurate world models, thus permitting high-level plans lacking viable refinements. To counteract this, we enhance the STAMP high-level planner to a top-k [24] version, generating multiple high-level plans. We then progressively refine these k plans until we achieve a plan supporting valid primitive action refinements for each high-level robot activity.

G Code and Data

Code and data are available on the project webpage at https://aair-lab.github.io/r2l-lamp.

H Learned Word Models

H.1 Delivering Items in a Cafe

```
(define (domain CafeWorld)
(:requirements :strips :typing :equality :conditional-effects
↪  :existential-preconditions :universal-preconditions)
(:types
        freight
        can
        gripper
        surface
)

(:constants
)

(:predicates
        (freight_surface_0 ?x - freight ?y - surface)
        (freight_surface_1 ?x - freight ?y - surface)
        (gripper_can_0 ?x - gripper ?y - can)
        (gripper_can_1 ?x - gripper ?y - can)
        (gripper_can_2 ?x - gripper ?y - can)
```

```
        (freight_gripper_0 ?x - freight ?y - gripper)
        (freight_gripper_1 ?x - freight ?y - gripper)
        (can_surface_0 ?x - can ?y - surface)
        (can_surface_1 ?x - can ?y - surface)
        (freight_can_0 ?x - freight ?y - can)
        (freight_can_1 ?x - freight ?y - can)
        (clear3_gripper_can_1 ?x - gripper)
        (clear3_freight_can_1 ?x - freight)
        (clear3_freight_gripper_1 ?x - freight)
        (clear3_freight_surface_1 ?x - freight)
        (clear3_can_surface_1 ?x - can)
        (clear3_gripper_can_2 ?x - gripper)
)

(:action a1
:parameters ( ?can_p1 - can  ?freight_p1 - freight  ?surface_extra_p1 -
↪  surface  ?gripper_p1 - gripper )
:precondition (and
        (can_surface_1 ?can_p1 ?surface_extra_p1)
        (freight_can_0 ?freight_p1 ?can_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
        (gripper_can_2 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_1 ?gripper_p1)
)
:effect (and
         (gripper_can_1 ?gripper_p1 ?can_p1)
        (not (gripper_can_0 ?gripper_p1 ?can_p1))
        (not (gripper_can_2 ?gripper_p1 ?can_p1))
        (clear3_gripper_can_2 ?gripper_p1)
        (not (clear3_gripper_can_1 ?gripper_p1))
 )
)

(:action a2
:parameters ( ?surface_extra_p4 - surface  ?can_p1 - can  ?gripper_p1 -
↪  gripper  ?surface_extra_p1 - surface  ?freight_p1 - freight )
:precondition (and
        (not (= ?surface_extra_p4 ?surface_extra_p1))
        (can_surface_0 ?can_p1 ?surface_extra_p4)
        (freight_can_0 ?freight_p1 ?can_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (freight_surface_0 ?freight_p1 ?surface_extra_p4)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
        (gripper_can_2 ?gripper_p1 ?can_p1)
        (clear3_freight_can_1 ?freight_p1)
        (clear3_freight_gripper_1 ?freight_p1)
)
:effect (and
         (freight_can_1 ?freight_p1 ?can_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (not (freight_can_0 ?freight_p1 ?can_p1))
        (not (freight_gripper_0 ?freight_p1 ?gripper_p1))
        (not (clear3_freight_can_1 ?freight_p1))
        (not (clear3_freight_gripper_1 ?freight_p1))
 )
)

(:action a3
```

```
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper  ?surface_extra_p1 -
↪  surface  ?freight_p1 - freight )
:precondition (and
        (can_surface_1 ?can_p1 ?surface_extra_p1)
        (freight_can_0 ?freight_p1 ?can_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
        (gripper_can_1 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_2 ?gripper_p1)
)
:effect (and
         (gripper_can_0 ?gripper_p1 ?can_p1)
        (not (gripper_can_1 ?gripper_p1 ?can_p1))
        (not (gripper_can_2 ?gripper_p1 ?can_p1))
        (clear3_gripper_can_1 ?gripper_p1)
 )
)

(:action a4
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper  ?surface_extra_p1 -
↪  surface  ?freight_p1 - freight )
:precondition (and
        (freight_can_1 ?freight_p1 ?can_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
        (gripper_can_2 ?gripper_p1 ?can_p1)
)
:effect (and
         (freight_can_0 ?freight_p1 ?can_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (not (freight_can_1 ?freight_p1 ?can_p1))
        (not (freight_gripper_1 ?freight_p1 ?gripper_p1))
        (clear3_freight_can_1 ?freight_p1)
        (clear3_freight_gripper_1 ?freight_p1)
 )
)

(:action a5
:parameters ( ?gripper_extra_p1 - gripper  ?can_p1 - can  ?freight_extra_p1
↪  - freight  ?surface_p1 - surface )
:precondition (and
        (can_surface_0 ?can_p1 ?surface_p1)
        (freight_can_0 ?freight_extra_p1 ?can_p1)
        (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
        (freight_surface_1 ?freight_extra_p1 ?surface_p1)
        (gripper_can_2 ?gripper_extra_p1 ?can_p1)
        (clear3_can_surface_1 ?can_p1)
)
:effect (and
         (can_surface_1 ?can_p1 ?surface_p1)
        (not (can_surface_0 ?can_p1 ?surface_p1))
        (not (clear3_can_surface_1 ?can_p1))
 )
)

(:action a6
:parameters ( ?gripper_p1 - gripper  ?surface_extra_p1 - surface
↪  ?freight_p1 - freight )
:precondition (and
```

```
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
)
:effect (and
         (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (not (freight_gripper_1 ?freight_p1 ?gripper_p1))
        (clear3_freight_gripper_1 ?freight_p1)
 )
)

(:action a7
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper  ?surface_extra_p1 -
↪  surface  ?freight_p1 - freight )
:precondition (and
        (can_surface_1 ?can_p1 ?surface_extra_p1)
        (freight_can_0 ?freight_p1 ?can_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
        (gripper_can_1 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_2 ?gripper_p1)
)
:effect (and
         (gripper_can_2 ?gripper_p1 ?can_p1)
        (not (gripper_can_0 ?gripper_p1 ?can_p1))
        (not (gripper_can_1 ?gripper_p1 ?can_p1))
        (clear3_gripper_can_1 ?gripper_p1)
        (not (clear3_gripper_can_2 ?gripper_p1))
 )
)

(:action a8
:parameters ( ?gripper_p1 - gripper  ?surface_extra_p1 - surface
↪  ?freight_p1 - freight )
:precondition (and
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
        (clear3_freight_gripper_1 ?freight_p1)
)
:effect (and
         (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (not (freight_gripper_0 ?freight_p1 ?gripper_p1))
        (not (clear3_freight_gripper_1 ?freight_p1))
 )
)

(:action a9
:parameters ( ?can_p1 - can  ?freight_p1 - freight  ?surface_extra_p1 -
↪  surface  ?gripper_p1 - gripper )
:precondition (and
        (can_surface_1 ?can_p1 ?surface_extra_p1)
        (freight_can_0 ?freight_p1 ?can_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_extra_p1)
        (gripper_can_0 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_1 ?gripper_p1)
        (clear3_gripper_can_2 ?gripper_p1)
)
:effect (and
         (gripper_can_1 ?gripper_p1 ?can_p1)
```

```
        (not (gripper_can_0 ?gripper_p1 ?can_p1))
        (not (gripper_can_2 ?gripper_p1 ?can_p1))
        (not (clear3_gripper_can_1 ?gripper_p1))
 )
)

(:action a10
:parameters ( ?gripper_p1 - gripper  ?surface_p1 - surface  ?freight_p1 -
↪  freight )
:precondition (and
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (freight_surface_0 ?freight_p1 ?surface_p1)
        (clear3_freight_surface_1 ?freight_p1)
)
:effect (and
         (freight_surface_1 ?freight_p1 ?surface_p1)
        (not (freight_surface_0 ?freight_p1 ?surface_p1))
        (not (clear3_freight_surface_1 ?freight_p1))
 )
)

(:action a11
:parameters ( ?gripper_extra_p1 - gripper  ?can_p1 - can  ?freight_extra_p1
↪  - freight  ?surface_p1 - surface )
:precondition (and
        (can_surface_1 ?can_p1 ?surface_p1)
        (freight_can_0 ?freight_extra_p1 ?can_p1)
        (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
        (freight_surface_1 ?freight_extra_p1 ?surface_p1)
        (gripper_can_2 ?gripper_extra_p1 ?can_p1)
)
:effect (and
         (can_surface_0 ?can_p1 ?surface_p1)
        (not (can_surface_1 ?can_p1 ?surface_p1))
        (clear3_can_surface_1 ?can_p1)
 )
)

(:action a12
:parameters ( ?gripper_p1 - gripper  ?surface_p1 - surface  ?freight_p1 -
↪  freight )
:precondition (and
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (freight_surface_1 ?freight_p1 ?surface_p1)
)
:effect (and
         (freight_surface_0 ?freight_p1 ?surface_p1)
        (not (freight_surface_1 ?freight_p1 ?surface_p1))
        (clear3_freight_surface_1 ?freight_p1)
 )
))
```

H.2 Setting Up a Dinner Table

```
(define (domain DinnerTable)
(:requirements :strips :typing :equality :conditional-effects
↪  :existential-preconditions :universal-preconditions)
(:types
        bowltargetLoc
```

```
        gripper
        glass
        bowl
        glasstargetLoc
        glassinitLoc
        freight
        bowlinitLoc
)

(:constants
)

(:predicates
        (glass_glasstargetLoc_0 ?x - glass ?y - glasstargetLoc)
        (glass_glasstargetLoc_1 ?x - glass ?y - glasstargetLoc)
        (freight_bowl_0 ?x - freight ?y - bowl)
        (freight_bowl_1 ?x - freight ?y - bowl)
        (freight_bowlinitLoc_0 ?x - freight ?y - bowlinitLoc)
        (freight_bowlinitLoc_1 ?x - freight ?y - bowlinitLoc)
        (bowl_bowlinitLoc_0 ?x - bowl ?y - bowlinitLoc)
        (bowl_bowlinitLoc_1 ?x - bowl ?y - bowlinitLoc)
        (gripper_bowl_0 ?x - gripper ?y - bowl)
        (gripper_bowl_1 ?x - gripper ?y - bowl)
        (gripper_bowl_2 ?x - gripper ?y - bowl)
        (freight_bowltargetLoc_0 ?x - freight ?y - bowltargetLoc)
        (freight_bowltargetLoc_1 ?x - freight ?y - bowltargetLoc)
        (freight_glass_0 ?x - freight ?y - glass)
        (freight_glass_1 ?x - freight ?y - glass)
        (bowl_bowltargetLoc_0 ?x - bowl ?y - bowltargetLoc)
        (bowl_bowltargetLoc_1 ?x - bowl ?y - bowltargetLoc)
        (glass_glassinitLoc_0 ?x - glass ?y - glassinitLoc)
        (glass_glassinitLoc_1 ?x - glass ?y - glassinitLoc)
        (freight_gripper_0 ?x - freight ?y - gripper)
        (freight_gripper_1 ?x - freight ?y - gripper)
        (freight_glassinitLoc_0 ?x - freight ?y - glassinitLoc)
        (freight_glassinitLoc_1 ?x - freight ?y - glassinitLoc)
        (freight_glasstargetLoc_0 ?x - freight ?y - glasstargetLoc)
        (freight_glasstargetLoc_1 ?x - freight ?y - glasstargetLoc)
        (gripper_glass_0 ?x - gripper ?y - glass)
        (gripper_glass_1 ?x - gripper ?y - glass)
        (gripper_glass_2 ?x - gripper ?y - glass)
        (clear3_freight_bowltargetLoc_1 ?x - freight)
        (clear3_gripper_glass_1 ?x - gripper)
        (clear3_gripper_bowl_2 ?x - gripper)
        (clear3_freight_bowlinitLoc_1 ?x - freight)
        (clear3_freight_bowl_1 ?x - freight)
        (clear3_freight_glass_1 ?x - freight)
        (clear3_gripper_glass_2 ?x - gripper)
        (clear3_bowl_bowlinitLoc_1 ?x - bowl)
        (clear3_gripper_bowl_1 ?x - gripper)
        (clear3_bowl_bowltargetLoc_1 ?x - bowl)
        (clear3_freight_glassinitLoc_1 ?x - freight)
        (clear3_glass_glasstargetLoc_1 ?x - glass)
        (clear3_freight_gripper_1 ?x - freight)
        (clear3_freight_glasstargetLoc_1 ?x - freight)
        (clear3_glass_glassinitLoc_1 ?x - glass)
)

(:action a1
```

```
:parameters ( ?bowl_extra_p1 - bowl  ?freight_p1 - freight  ?bowlinitLoc_p1
↪  - bowlinitLoc  ?gripper_p1 - gripper )
:precondition (and
        (freight_bowl_1 ?freight_p1 ?bowl_extra_p1)
        (freight_bowlinitLoc_1 ?freight_p1 ?bowlinitLoc_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (gripper_bowl_2 ?gripper_p1 ?bowl_extra_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
)
:effect (and
         (freight_bowlinitLoc_0 ?freight_p1 ?bowlinitLoc_p1)
        (not (freight_bowlinitLoc_1 ?freight_p1 ?bowlinitLoc_p1))
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
 )
)

(:action a2
:parameters ( ?bowltargetLoc_extra_p1 - bowltargetLoc  ?bowl_p1 - bowl
↪  ?gripper_p1 - gripper  ?freight_p1 - freight )
:precondition (and
        (freight_bowl_1 ?freight_p1 ?bowl_p1)
        (freight_bowltargetLoc_1 ?freight_p1 ?bowltargetLoc_extra_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (gripper_bowl_2 ?gripper_p1 ?bowl_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (freight_bowl_0 ?freight_p1 ?bowl_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (not (freight_bowl_1 ?freight_p1 ?bowl_p1))
        (not (freight_gripper_1 ?freight_p1 ?gripper_p1))
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_gripper_1 ?freight_p1)
 )
)

(:action a3
:parameters ( ?glassinitLoc_extra_p1 - glassinitLoc  ?gripper_p1 - gripper
↪  ?glass_p1 - glass  ?freight_p1 - freight )
:precondition (and
        (freight_glass_0 ?freight_p1 ?glass_p1)
        (freight_glassinitLoc_1 ?freight_p1 ?glassinitLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (glass_glassinitLoc_0 ?glass_p1 ?glassinitLoc_extra_p1)
        (gripper_glass_2 ?gripper_p1 ?glass_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
```

```
        (clear3_freight_gripper_1 ?freight_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
)
:effect (and
         (freight_glass_1 ?freight_p1 ?glass_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (not (freight_glass_0 ?freight_p1 ?glass_p1))
        (not (freight_gripper_0 ?freight_p1 ?gripper_p1))
        (not (clear3_freight_glass_1 ?freight_p1))
        (not (clear3_freight_gripper_1 ?freight_p1))
 )
)

(:action a4
:parameters ( ?glassinitLoc_extra_p1 - glassinitLoc  ?gripper_p1 - gripper
↪  ?glass_p1 - glass  ?freight_p1 - freight )
:precondition (and
        (freight_glass_0 ?freight_p1 ?glass_p1)
        (freight_glassinitLoc_1 ?freight_p1 ?glassinitLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (glass_glassinitLoc_1 ?glass_p1 ?glassinitLoc_extra_p1)
        (gripper_glass_1 ?gripper_p1 ?glass_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (gripper_glass_2 ?gripper_p1 ?glass_p1)
        (not (gripper_glass_0 ?gripper_p1 ?glass_p1))
        (not (gripper_glass_1 ?gripper_p1 ?glass_p1))
        (clear3_gripper_glass_1 ?gripper_p1)
        (not (clear3_gripper_glass_2 ?gripper_p1))
 )
)

(:action a5
:parameters ( ?bowl_p1 - bowl  ?gripper_extra_p1 - gripper
↪  ?freight_extra_p1 - freight  ?bowlinitLoc_p1 - bowlinitLoc )
:precondition (and
        (bowl_bowlinitLoc_1 ?bowl_p1 ?bowlinitLoc_p1)
        (freight_bowl_0 ?freight_extra_p1 ?bowl_p1)
        (freight_bowlinitLoc_1 ?freight_extra_p1 ?bowlinitLoc_p1)
        (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
        (gripper_bowl_2 ?gripper_extra_p1 ?bowl_p1)
)
:effect (and
         (bowl_bowlinitLoc_0 ?bowl_p1 ?bowlinitLoc_p1)
        (not (bowl_bowlinitLoc_1 ?bowl_p1 ?bowlinitLoc_p1))
        (clear3_bowl_bowlinitLoc_1 ?bowl_p1)
 )
)

(:action a6
:parameters ( ?bowltargetLoc_p1 - bowltargetLoc  ?gripper_p1 - gripper
↪  ?bowlinitLoc_extra_p1 - bowlinitLoc  ?freight_p1 - freight )
:precondition (and
        (freight_bowlinitLoc_0 ?freight_p1 ?bowlinitLoc_extra_p1)
```

```
        (freight_bowltargetLoc_1 ?freight_p1 ?bowltargetLoc_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
)
:effect (and
         (freight_bowltargetLoc_0 ?freight_p1 ?bowltargetLoc_p1)
        (not (freight_bowltargetLoc_1 ?freight_p1 ?bowltargetLoc_p1))
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
 )
)

(:action a7
:parameters ( ?bowl_p1 - bowl  ?gripper_extra_p1 - gripper
↪  ?bowlinitLoc_extra_p1 - bowlinitLoc  ?freight_extra_p1 - freight
↪  ?bowltargetLoc_p1 - bowltargetLoc )
:precondition (and
        (bowl_bowlinitLoc_0 ?bowl_p1 ?bowlinitLoc_extra_p1)
        (bowl_bowltargetLoc_0 ?bowl_p1 ?bowltargetLoc_p1)
        (freight_bowl_0 ?freight_extra_p1 ?bowl_p1)
        (freight_bowlinitLoc_0 ?freight_extra_p1 ?bowlinitLoc_extra_p1)
        (freight_bowltargetLoc_1 ?freight_extra_p1 ?bowltargetLoc_p1)
        (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
        (gripper_bowl_2 ?gripper_extra_p1 ?bowl_p1)
        (clear3_bowl_bowltargetLoc_1 ?bowl_p1)
)
:effect (and
         (bowl_bowltargetLoc_1 ?bowl_p1 ?bowltargetLoc_p1)
        (not (bowl_bowltargetLoc_0 ?bowl_p1 ?bowltargetLoc_p1))
        (not (clear3_bowl_bowltargetLoc_1 ?bowl_p1))
 )
)

(:action a8
:parameters ( ?glass_p1 - glass  ?gripper_p1 - gripper
↪  ?glasstargetLoc_extra_p1 - glasstargetLoc  ?freight_p1 - freight )
:precondition (and
        (freight_glass_0 ?freight_p1 ?glass_p1)
        (freight_glasstargetLoc_1 ?freight_p1 ?glasstargetLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (glass_glasstargetLoc_1 ?glass_p1 ?glasstargetLoc_extra_p1)
        (gripper_glass_2 ?gripper_p1 ?glass_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
)
:effect (and
         (gripper_glass_1 ?gripper_p1 ?glass_p1)
        (not (gripper_glass_0 ?gripper_p1 ?glass_p1))
        (not (gripper_glass_2 ?gripper_p1 ?glass_p1))
        (clear3_gripper_glass_2 ?gripper_p1)
        (not (clear3_gripper_glass_1 ?gripper_p1))
 )
)

(:action a9
```

```
:parameters ( ?bowltargetLoc_extra_p1 - bowltargetLoc  ?bowl_p1 - bowl
↪  ?gripper_p1 - gripper  ?freight_p1 - freight )
:precondition (and
        (bowl_bowltargetLoc_1 ?bowl_p1 ?bowltargetLoc_extra_p1)
        (freight_bowl_0 ?freight_p1 ?bowl_p1)
        (freight_bowltargetLoc_1 ?freight_p1 ?bowltargetLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (gripper_bowl_1 ?gripper_p1 ?bowl_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (gripper_bowl_0 ?gripper_p1 ?bowl_p1)
        (not (gripper_bowl_1 ?gripper_p1 ?bowl_p1))
        (not (gripper_bowl_2 ?gripper_p1 ?bowl_p1))
        (clear3_gripper_bowl_1 ?gripper_p1)
 )
)

(:action a10
:parameters ( ?bowltargetLoc_extra_p1 - bowltargetLoc  ?bowl_p1 - bowl
↪  ?gripper_p1 - gripper  ?freight_p1 - freight )
:precondition (and
        (bowl_bowltargetLoc_1 ?bowl_p1 ?bowltargetLoc_extra_p1)
        (freight_bowl_0 ?freight_p1 ?bowl_p1)
        (freight_bowltargetLoc_1 ?freight_p1 ?bowltargetLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (gripper_bowl_2 ?gripper_p1 ?bowl_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (gripper_bowl_1 ?gripper_p1 ?bowl_p1)
        (not (gripper_bowl_0 ?gripper_p1 ?bowl_p1))
        (not (gripper_bowl_2 ?gripper_p1 ?bowl_p1))
        (clear3_gripper_bowl_2 ?gripper_p1)
        (not (clear3_gripper_bowl_1 ?gripper_p1))
 )
)

(:action a11
:parameters ( ?freight_extra_p1 - freight  ?glass_p1 - glass
↪  ?gripper_extra_p1 - gripper  ?glasstargetLoc_p1 - glasstargetLoc )
:precondition (and
        (freight_glass_0 ?freight_extra_p1 ?glass_p1)
        (freight_glasstargetLoc_1 ?freight_extra_p1 ?glasstargetLoc_p1)
        (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
        (glass_glasstargetLoc_0 ?glass_p1 ?glasstargetLoc_p1)
        (gripper_glass_2 ?gripper_extra_p1 ?glass_p1)
        (clear3_glass_glasstargetLoc_1 ?glass_p1)
)
:effect (and
         (glass_glasstargetLoc_1 ?glass_p1 ?glasstargetLoc_p1)
        (not (glass_glasstargetLoc_0 ?glass_p1 ?glasstargetLoc_p1))
        (not (clear3_glass_glasstargetLoc_1 ?glass_p1))
 )
)
```

```
(:action a12
:parameters ( ?bowltargetLoc_p1 - bowltargetLoc  ?bowl_extra_p1 - bowl
↪  ?gripper_p1 - gripper  ?bowlinitLoc_extra_p1 - bowlinitLoc  ?freight_p1
↪  - freight )
:precondition (and
        (bowl_bowlinitLoc_0 ?bowl_extra_p1 ?bowlinitLoc_extra_p1)
        (freight_bowl_1 ?freight_p1 ?bowl_extra_p1)
        (freight_bowlinitLoc_0 ?freight_p1 ?bowlinitLoc_extra_p1)
        (freight_bowltargetLoc_0 ?freight_p1 ?bowltargetLoc_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (gripper_bowl_2 ?gripper_p1 ?bowl_extra_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
)
:effect (and
         (freight_bowltargetLoc_1 ?freight_p1 ?bowltargetLoc_p1)
        (not (freight_bowltargetLoc_0 ?freight_p1 ?bowltargetLoc_p1))
        (not (clear3_freight_bowltargetLoc_1 ?freight_p1))
 )
)

(:action a13
:parameters ( ?glassinitLoc_p1 - glassinitLoc  ?gripper_p1 - gripper
↪  ?freight_p1 - freight )
:precondition (and
        (freight_glassinitLoc_0 ?freight_p1 ?glassinitLoc_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
)
:effect (and
         (freight_glassinitLoc_1 ?freight_p1 ?glassinitLoc_p1)
        (not (freight_glassinitLoc_0 ?freight_p1 ?glassinitLoc_p1))
        (not (clear3_freight_glassinitLoc_1 ?freight_p1))
 )
)

(:action a14
:parameters ( ?gripper_p1 - gripper  ?bowl_p1 - bowl  ?bowlinitLoc_extra_p1
↪  - bowlinitLoc  ?freight_p1 - freight )
:precondition (and
        (bowl_bowlinitLoc_1 ?bowl_p1 ?bowlinitLoc_extra_p1)
        (freight_bowl_0 ?freight_p1 ?bowl_p1)
        (freight_bowlinitLoc_1 ?freight_p1 ?bowlinitLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (gripper_bowl_1 ?gripper_p1 ?bowl_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
```

```
         (gripper_bowl_2 ?gripper_p1 ?bowl_p1)
        (not (gripper_bowl_0 ?gripper_p1 ?bowl_p1))
        (not (gripper_bowl_1 ?gripper_p1 ?bowl_p1))
        (clear3_gripper_bowl_1 ?gripper_p1)
        (not (clear3_gripper_bowl_2 ?gripper_p1))
 )
)

(:action a15
:parameters ( ?gripper_p1 - gripper  ?bowl_p1 - bowl  ?bowlinitLoc_extra_p1
↪  - bowlinitLoc  ?freight_p1 - freight )
:precondition (and
        (bowl_bowlinitLoc_1 ?bowl_p1 ?bowlinitLoc_extra_p1)
        (freight_bowl_0 ?freight_p1 ?bowl_p1)
        (freight_bowlinitLoc_1 ?freight_p1 ?bowlinitLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (gripper_bowl_0 ?gripper_p1 ?bowl_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (gripper_bowl_1 ?gripper_p1 ?bowl_p1)
        (not (gripper_bowl_0 ?gripper_p1 ?bowl_p1))
        (not (gripper_bowl_2 ?gripper_p1 ?bowl_p1))
        (not (clear3_gripper_bowl_1 ?gripper_p1))
 )
)

(:action a16
:parameters ( ?glass_p1 - glass  ?gripper_p1 - gripper
↪  ?glasstargetLoc_extra_p1 - glasstargetLoc  ?freight_p1 - freight )
:precondition (and
        (freight_glass_1 ?freight_p1 ?glass_p1)
        (freight_glasstargetLoc_1 ?freight_p1 ?glasstargetLoc_extra_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (gripper_glass_2 ?gripper_p1 ?glass_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
)
:effect (and
         (freight_glass_0 ?freight_p1 ?glass_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (not (freight_glass_1 ?freight_p1 ?glass_p1))
        (not (freight_gripper_1 ?freight_p1 ?gripper_p1))
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_gripper_1 ?freight_p1)
 )
)

(:action a17
```

```
:parameters ( ?glasstargetLoc_extra_p1 - glasstargetLoc  ?glass_p1 - glass
↪  ?freight_p1 - freight  ?glassinitLoc_extra_p1 - glassinitLoc
↪  ?gripper_p1 - gripper )
:precondition (and
        (freight_glass_0 ?freight_p1 ?glass_p1)
        (freight_glassinitLoc_0 ?freight_p1 ?glassinitLoc_extra_p1)
        (freight_glasstargetLoc_1 ?freight_p1 ?glasstargetLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (glass_glassinitLoc_0 ?glass_p1 ?glassinitLoc_extra_p1)
        (glass_glasstargetLoc_1 ?glass_p1 ?glasstargetLoc_extra_p1)
        (gripper_glass_1 ?gripper_p1 ?glass_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (gripper_glass_0 ?gripper_p1 ?glass_p1)
        (not (gripper_glass_1 ?gripper_p1 ?glass_p1))
        (not (gripper_glass_2 ?gripper_p1 ?glass_p1))
        (clear3_gripper_glass_1 ?gripper_p1)
 )
)

(:action a18
:parameters ( ?glasstargetLoc_p1 - glasstargetLoc  ?gripper_p1 - gripper
↪  ?glassinitLoc_extra_p1 - glassinitLoc  ?glass_extra_p1 - glass
↪  ?freight_p1 - freight )
:precondition (and
        (freight_glass_1 ?freight_p1 ?glass_extra_p1)
        (freight_glassinitLoc_0 ?freight_p1 ?glassinitLoc_extra_p1)
        (freight_glasstargetLoc_0 ?freight_p1 ?glasstargetLoc_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (gripper_glass_2 ?gripper_p1 ?glass_extra_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
)
:effect (and
         (freight_glasstargetLoc_1 ?freight_p1 ?glasstargetLoc_p1)
        (not (freight_glasstargetLoc_0 ?freight_p1 ?glasstargetLoc_p1))
        (not (clear3_freight_glasstargetLoc_1 ?freight_p1))
 )
)

(:action a19
:parameters ( ?gripper_p1 - gripper  ?freight_p1 - freight )
:precondition (and
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_gripper_1 ?freight_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
```

```
)
:effect (and
         (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (not (freight_gripper_0 ?freight_p1 ?gripper_p1))
        (not (clear3_freight_gripper_1 ?freight_p1))
 )
)

(:action a20
:parameters ( ?glassinitLoc_extra_p1 - glassinitLoc  ?gripper_p1 - gripper
↪  ?glasstargetLoc_p1 - glasstargetLoc  ?freight_p1 - freight )
:precondition (and
        (freight_glassinitLoc_0 ?freight_p1 ?glassinitLoc_extra_p1)
        (freight_glasstargetLoc_1 ?freight_p1 ?glasstargetLoc_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
)
:effect (and
         (freight_glasstargetLoc_0 ?freight_p1 ?glasstargetLoc_p1)
        (not (freight_glasstargetLoc_1 ?freight_p1 ?glasstargetLoc_p1))
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
 )
)

(:action a21
:parameters ( ?glassinitLoc_p1 - glassinitLoc  ?glass_p1 - glass
↪  ?freight_extra_p1 - freight  ?gripper_extra_p1 - gripper )
:precondition (and
        (freight_glass_0 ?freight_extra_p1 ?glass_p1)
        (freight_glassinitLoc_1 ?freight_extra_p1 ?glassinitLoc_p1)
        (freight_gripper_0 ?freight_extra_p1 ?gripper_extra_p1)
        (glass_glassinitLoc_1 ?glass_p1 ?glassinitLoc_p1)
        (gripper_glass_2 ?gripper_extra_p1 ?glass_p1)
)
:effect (and
         (glass_glassinitLoc_0 ?glass_p1 ?glassinitLoc_p1)
        (not (glass_glassinitLoc_1 ?glass_p1 ?glassinitLoc_p1))
        (clear3_glass_glassinitLoc_1 ?glass_p1)
 )
)

(:action a22
:parameters ( ?gripper_p1 - gripper  ?bowl_p1 - bowl  ?bowlinitLoc_extra_p1
↪  - bowlinitLoc  ?freight_p1 - freight )
:precondition (and
        (bowl_bowlinitLoc_0 ?bowl_p1 ?bowlinitLoc_extra_p1)
        (freight_bowl_0 ?freight_p1 ?bowl_p1)
        (freight_bowlinitLoc_1 ?freight_p1 ?bowlinitLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (gripper_bowl_2 ?gripper_p1 ?bowl_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
```

```
        (clear3_freight_gripper_1 ?freight_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (freight_bowl_1 ?freight_p1 ?bowl_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (not (freight_bowl_0 ?freight_p1 ?bowl_p1))
        (not (freight_gripper_0 ?freight_p1 ?gripper_p1))
        (not (clear3_freight_bowl_1 ?freight_p1))
        (not (clear3_freight_gripper_1 ?freight_p1))
 )
)

(:action a23
:parameters ( ?gripper_p1 - gripper  ?freight_p1 - freight )
:precondition (and
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (not (freight_gripper_1 ?freight_p1 ?gripper_p1))
        (clear3_freight_gripper_1 ?freight_p1)
 )
)

(:action a24
:parameters ( ?glassinitLoc_p1 - glassinitLoc  ?gripper_p1 - gripper
↪  ?glass_extra_p1 - glass  ?freight_p1 - freight )
:precondition (and
        (freight_glass_1 ?freight_p1 ?glass_extra_p1)
        (freight_glassinitLoc_1 ?freight_p1 ?glassinitLoc_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (gripper_glass_2 ?gripper_p1 ?glass_extra_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
)
:effect (and
         (freight_glassinitLoc_0 ?freight_p1 ?glassinitLoc_p1)
        (not (freight_glassinitLoc_1 ?freight_p1 ?glassinitLoc_p1))
        (clear3_freight_glassinitLoc_1 ?freight_p1)
 )
)

(:action a25
:parameters ( ?glassinitLoc_extra_p1 - glassinitLoc  ?gripper_p1 - gripper
↪  ?glass_p1 - glass  ?freight_p1 - freight )
:precondition (and
```

```
        (freight_glass_0 ?freight_p1 ?glass_p1)
        (freight_glassinitLoc_1 ?freight_p1 ?glassinitLoc_extra_p1)
        (freight_gripper_0 ?freight_p1 ?gripper_p1)
        (glass_glassinitLoc_1 ?glass_p1 ?glassinitLoc_extra_p1)
        (gripper_glass_0 ?gripper_p1 ?glass_p1)
        (clear3_gripper_bowl_1 ?gripper_p1)
        (clear3_gripper_bowl_2 ?gripper_p1)
        (clear3_gripper_glass_1 ?gripper_p1)
        (clear3_gripper_glass_2 ?gripper_p1)
)
:effect (and
         (gripper_glass_1 ?gripper_p1 ?glass_p1)
        (not (gripper_glass_0 ?gripper_p1 ?glass_p1))
        (not (gripper_glass_2 ?gripper_p1 ?glass_p1))
        (not (clear3_gripper_glass_1 ?gripper_p1))
 )
)

(:action a26
:parameters ( ?gripper_p1 - gripper  ?bowlinitLoc_p1 - bowlinitLoc
↪  ?glasstargetLoc_extra_p1 - glasstargetLoc  ?freight_p1 - freight )
:precondition (and
        (freight_bowlinitLoc_0 ?freight_p1 ?bowlinitLoc_p1)
        (freight_glasstargetLoc_0 ?freight_p1 ?glasstargetLoc_extra_p1)
        (freight_gripper_1 ?freight_p1 ?gripper_p1)
        (clear3_freight_bowl_1 ?freight_p1)
        (clear3_freight_bowlinitLoc_1 ?freight_p1)
        (clear3_freight_bowltargetLoc_1 ?freight_p1)
        (clear3_freight_glass_1 ?freight_p1)
        (clear3_freight_glassinitLoc_1 ?freight_p1)
        (clear3_freight_glasstargetLoc_1 ?freight_p1)
)
:effect (and
         (freight_bowlinitLoc_1 ?freight_p1 ?bowlinitLoc_p1)
        (not (freight_bowlinitLoc_0 ?freight_p1 ?bowlinitLoc_p1))
        (not (clear3_freight_bowlinitLoc_1 ?freight_p1))
 )
))
```

H.3 Building Structures with Keva Planks

```
(define (domain Keva)
(:requirements :strips :typing :equality :conditional-effects
    :existential-preconditions :universal-preconditions)
(:types
            loc
            plank
            gripper
)

(:constants
    loc_plankTarget_Const - loc
 )

(:predicates
            (gripper_plank_0 ?x - gripper ?y - plank)
            (gripper_plank_1 ?x - gripper ?y - plank)
            (gripper_plank_2 ?x - gripper ?y - plank)
            (plank_plank_0 ?x - plank ?y - plank)
```

```
            (plank_plank_1 ?x - plank ?y - plank)
            (plank_plank_2 ?x - plank ?y - plank)
            (plank_plank_3 ?x - plank ?y - plank)
            (plank_plank_4 ?x - plank ?y - plank)
            (plank_plank_5 ?x - plank ?y - plank)
            (plank_plank_6 ?x - plank ?y - plank)
            (plank_plank_7 ?x - plank ?y - plank)
            (plank_plank_8 ?x - plank ?y - plank)
            (plank_plank_9 ?x - plank ?y - plank)
            (loc_plank_0 ?x - loc ?y - plank)
            (loc_plank_1 ?x - loc ?y - plank)
            (clear3_plank_plank_2 ?x - plank)
            (clear3_gripper_plank_2 ?x - gripper)
            (clear3_plank_plank_6 ?x - plank)
            (clear3_plank_plank_8 ?x - plank)
            (clear3_plank_plank_5 ?x - plank)
            (clear3_plank_plank_7 ?x - plank)
            (clear3_plank_plank_9 ?x - plank)
            (clear3_plank_plank_4 ?x - plank)
            (clear3_gripper_plank_1 ?x - gripper)
            (clear3_plank_plank_1 ?x - plank)
            (clear3_plank_plank_3 ?x - plank)
)

(:action a1
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p2 - plank
        ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p2)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (clear3_plank_plank_4 ?plank_p1)
                (clear3_plank_plank_4 ?plank_p2)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (plank_plank_4 ?plank_p2 ?plank_p1)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p1))
                (not (plank_plank_0 ?plank_p2 ?plank_p1))
                (not (plank_plank_1 ?plank_p2 ?plank_p1))
                (not (plank_plank_2 ?plank_p2 ?plank_p1))
                (not (plank_plank_3 ?plank_p2 ?plank_p1))
                (not (plank_plank_5 ?plank_p2 ?plank_p1))
                (not (plank_plank_6 ?plank_p2 ?plank_p1))
                (not (plank_plank_7 ?plank_p2 ?plank_p1))
                (not (plank_plank_8 ?plank_p2 ?plank_p1))
                (not (plank_plank_9 ?plank_p2 ?plank_p1))
                (not (clear3_plank_plank_4 ?plank_p2))
    )
)

(:action a2
```

```
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p3 - plank  ?plank_p2
    ↪  - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p3)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_4 ?plank_p3 ?plank_p1)
                (clear3_plank_plank_1 ?plank_p1)
                (clear3_plank_plank_1 ?plank_p2)
                (clear3_plank_plank_1 ?plank_p3)
                (clear3_plank_plank_3 ?plank_p1)
                (clear3_plank_plank_3 ?plank_p2)
                (clear3_plank_plank_3 ?plank_p3)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_1 ?plank_p1 ?plank_p2)
                (plank_plank_3 ?plank_p3 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_1 ?plank_p3 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p1 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (plank_plank_9 ?plank_p3 ?plank_p2))
                (not (clear3_plank_plank_1 ?plank_p1))
                (not (clear3_plank_plank_3 ?plank_p3))
    )
)

(:action a3
    :parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
```

```
    :precondition (and
                (gripper_plank_2 ?gripper_p1 ?plank_p1)
                (clear3_gripper_plank_1 ?gripper_p1)
    )
    :effect (and
                 (gripper_plank_1 ?gripper_p1 ?plank_p1)
                (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
                (not (gripper_plank_2 ?gripper_p1 ?plank_p1))
                (clear3_gripper_plank_2 ?gripper_p1)
                (not (clear3_gripper_plank_1 ?gripper_p1))
    )
)

(:action a4
    :parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
    :precondition (and
                (gripper_plank_0 ?gripper_p1 ?plank_p1)
                (clear3_gripper_plank_1 ?gripper_p1)
                (clear3_gripper_plank_2 ?gripper_p1)
    )
    :effect (and
                 (gripper_plank_2 ?gripper_p1 ?plank_p1)
                (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
                (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
                (not (clear3_gripper_plank_2 ?gripper_p1))
    )
)

(:action a5
    :parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
    :precondition (and
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (gripper_plank_2 ?gripper_p1 ?plank_p1)
                (clear3_gripper_plank_1 ?gripper_p1)
    )
    :effect (and
                 (gripper_plank_0 ?gripper_p1 ?plank_p1)
                (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
                (not (gripper_plank_2 ?gripper_p1 ?plank_p1))
                (clear3_gripper_plank_2 ?gripper_p1)
    )
)

(:action a6
    :parameters ( ?plank_p1 - plank  ?gripper_p1 - gripper )
    :precondition (and
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (gripper_plank_1 ?gripper_p1 ?plank_p1)
                (clear3_gripper_plank_2 ?gripper_p1)
    )
    :effect (and
                 (gripper_plank_2 ?gripper_p1 ?plank_p1)
                (not (gripper_plank_0 ?gripper_p1 ?plank_p1))
                (not (gripper_plank_1 ?gripper_p1 ?plank_p1))
                (clear3_gripper_plank_1 ?gripper_p1)
                (not (clear3_gripper_plank_2 ?gripper_p1))
    )
)
```

```
(:action a7
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p3 - plank  ?plank_p2
    ↪  - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p3)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_4 ?plank_p3 ?plank_p1)
                (clear3_plank_plank_8 ?plank_p1)
                (clear3_plank_plank_8 ?plank_p2)
                (clear3_plank_plank_8 ?plank_p3)
                (clear3_plank_plank_9 ?plank_p1)
                (clear3_plank_plank_9 ?plank_p2)
                (clear3_plank_plank_9 ?plank_p3)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_8 ?plank_p1 ?plank_p2)
                (plank_plank_9 ?plank_p3 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_1 ?plank_p3 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p1 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p1 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (clear3_plank_plank_8 ?plank_p1))
                (not (clear3_plank_plank_9 ?plank_p3))
    )
)

(:action a8
```

```
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p3 - plank  ?plank_p2
    ↪  - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p3)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_4 ?plank_p3 ?plank_p1)
                (clear3_plank_plank_2 ?plank_p1)
                (clear3_plank_plank_2 ?plank_p2)
                (clear3_plank_plank_2 ?plank_p3)
                (clear3_plank_plank_6 ?plank_p1)
                (clear3_plank_plank_6 ?plank_p2)
                (clear3_plank_plank_6 ?plank_p3)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_2 ?plank_p3 ?plank_p2)
                (plank_plank_6 ?plank_p1 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_1 ?plank_p3 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_3 ?plank_p1 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (plank_plank_9 ?plank_p3 ?plank_p2))
                (not (clear3_plank_plank_2 ?plank_p3))
                (not (clear3_plank_plank_6 ?plank_p1))
    )
)

(:action a9
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p2 - plank
```

```
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p2)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (clear3_plank_plank_5 ?plank_p1)
                (clear3_plank_plank_5 ?plank_p2)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (plank_plank_5 ?plank_p2 ?plank_p1)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p1))
                (not (plank_plank_0 ?plank_p2 ?plank_p1))
                (not (plank_plank_1 ?plank_p2 ?plank_p1))
                (not (plank_plank_2 ?plank_p2 ?plank_p1))
                (not (plank_plank_3 ?plank_p2 ?plank_p1))
                (not (plank_plank_4 ?plank_p2 ?plank_p1))
                (not (plank_plank_6 ?plank_p2 ?plank_p1))
                (not (plank_plank_7 ?plank_p2 ?plank_p1))
                (not (plank_plank_8 ?plank_p2 ?plank_p1))
                (not (plank_plank_9 ?plank_p2 ?plank_p1))
                (not (clear3_plank_plank_5 ?plank_p2))
    )
)

(:action a10
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p2 - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p2)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (clear3_plank_plank_7 ?plank_p1)
                (clear3_plank_plank_7 ?plank_p2)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (plank_plank_7 ?plank_p2 ?plank_p1)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p1))
                (not (plank_plank_0 ?plank_p2 ?plank_p1))
                (not (plank_plank_1 ?plank_p2 ?plank_p1))
                (not (plank_plank_2 ?plank_p2 ?plank_p1))
                (not (plank_plank_3 ?plank_p2 ?plank_p1))
                (not (plank_plank_4 ?plank_p2 ?plank_p1))
                (not (plank_plank_5 ?plank_p2 ?plank_p1))
                (not (plank_plank_6 ?plank_p2 ?plank_p1))
                (not (plank_plank_8 ?plank_p2 ?plank_p1))
```

```
                (not (plank_plank_9 ?plank_p2 ?plank_p1))
                (not (clear3_plank_plank_7 ?plank_p2))
    )
)

(:action a11
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p1 - plank )
    :precondition (and
                (loc_plank_0 loc_plankTarget_Const ?plank_p1)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p1)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p1))
    )
)

(:action a12
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p3 - plank  ?plank_p2
    ↪  - plank
            ?plank_p1 - plank )
:precondition (and
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p1)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_7 ?plank_p1 ?plank_p3)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_4 ?plank_p3 ?plank_p2)
                (plank_plank_5 ?plank_p1 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_1 ?plank_p3 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p1 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p1 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
```

```
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (plank_plank_9 ?plank_p3 ?plank_p2))
                (not (clear3_plank_plank_4 ?plank_p3))
                (not (clear3_plank_plank_5 ?plank_p1))
    )
)

(:action a13
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p3 - plank  ?plank_p2
    ↪  - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p3)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_4 ?plank_p3 ?plank_p1)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_8 ?plank_p1 ?plank_p2)
                (plank_plank_9 ?plank_p3 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_1 ?plank_p3 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p1 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p1 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (clear3_plank_plank_8 ?plank_p1))
                (not (clear3_plank_plank_9 ?plank_p3))
```

```
    )
)

(:action a14
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p4 - plank  ?plank_p3
    ↪  - plank
    ?plank_p2 - plank  ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p4 ?plank_p3))
                (not (= ?plank_p4 ?plank_p2))
                (not (= ?plank_p4 ?plank_p1))
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (loc_plank_1 loc_plankTarget_Const ?plank_p4)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p4)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p2 ?plank_p4)
                (plank_plank_0 ?plank_p3 ?plank_p1)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_0 ?plank_p4 ?plank_p1)
                (plank_plank_0 ?plank_p4 ?plank_p2)
                (plank_plank_0 ?plank_p4 ?plank_p3)
                (plank_plank_0 ?plank_p4 ?plank_p4)
                (plank_plank_2 ?plank_p1 ?plank_p4)
                (plank_plank_4 ?plank_p1 ?plank_p3)
                (plank_plank_6 ?plank_p3 ?plank_p4)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_1 ?plank_p3 ?plank_p2)
                (plank_plank_3 ?plank_p1 ?plank_p2)
                (plank_plank_4 ?plank_p4 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_0 ?plank_p4 ?plank_p2))
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_1 ?plank_p4 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_2 ?plank_p4 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p4 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
```

```
                (not (plank_plank_5 ?plank_p4 ?plank_p2))
                (not (plank_plank_6 ?plank_p1 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p4 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p4 ?plank_p2))
                (not (plank_plank_8 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p4 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (plank_plank_9 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p4 ?plank_p2))
                (not (clear3_plank_plank_1 ?plank_p3))
                (not (clear3_plank_plank_3 ?plank_p1))
                (not (clear3_plank_plank_4 ?plank_p4))
     )
)

(:action a15
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p3 - plank  ?plank_p2
    ↪  - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p1)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_4 ?plank_p1 ?plank_p3)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_1 ?plank_p3 ?plank_p2)
                (plank_plank_3 ?plank_p1 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p1 ?plank_p2))
```

```
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (plank_plank_9 ?plank_p3 ?plank_p2))
                (not (clear3_plank_plank_1 ?plank_p3))
                (not (clear3_plank_plank_3 ?plank_p1))
    )
)

(:action a16
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p4 - plank  ?plank_p3
    ↪  - plank  ?plank_p2 - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p4 ?plank_p3))
                (not (= ?plank_p4 ?plank_p2))
                (not (= ?plank_p4 ?plank_p1))
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (loc_plank_1 loc_plankTarget_Const ?plank_p4)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p4)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p3)
                (plank_plank_0 ?plank_p1 ?plank_p4)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p2 ?plank_p4)
                (plank_plank_0 ?plank_p3 ?plank_p1)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p4)
                (plank_plank_0 ?plank_p4 ?plank_p2)
                (plank_plank_0 ?plank_p4 ?plank_p3)
                (plank_plank_0 ?plank_p4 ?plank_p4)
                (plank_plank_4 ?plank_p4 ?plank_p1)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_2 ?plank_p4 ?plank_p2)
                (plank_plank_4 ?plank_p3 ?plank_p2)
                (plank_plank_6 ?plank_p1 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
                (not (plank_plank_0 ?plank_p4 ?plank_p2))
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_1 ?plank_p3 ?plank_p2))
```

```
                (not (plank_plank_1 ?plank_p4 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p1 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p4 ?plank_p2))
                (not (plank_plank_4 ?plank_p1 ?plank_p2))
                (not (plank_plank_4 ?plank_p4 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p4 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p4 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_7 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p4 ?plank_p2))
                (not (plank_plank_8 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_8 ?plank_p4 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (plank_plank_9 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p4 ?plank_p2))
                (not (clear3_plank_plank_2 ?plank_p4))
                (not (clear3_plank_plank_4 ?plank_p3))
                (not (clear3_plank_plank_6 ?plank_p1))
    )
)

(:action a17
    :parameters ( ?gripper_extra_p1 - gripper  ?plank_p3 - plank  ?plank_p2
    ↪  - plank
            ?plank_p1 - plank )
    :precondition (and
                (not (= ?plank_p3 ?plank_p2))
                (not (= ?plank_p3 ?plank_p1))
                (not (= ?plank_p2 ?plank_p1))
                (loc_plank_0 loc_plankTarget_Const ?plank_p2)
                (loc_plank_1 loc_plankTarget_Const ?plank_p1)
                (loc_plank_1 loc_plankTarget_Const ?plank_p3)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p1)
                (gripper_plank_0 ?gripper_extra_p1 ?plank_p3)
                (gripper_plank_1 ?gripper_extra_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p1)
                (plank_plank_0 ?plank_p1 ?plank_p2)
                (plank_plank_0 ?plank_p1 ?plank_p3)
                (plank_plank_0 ?plank_p2 ?plank_p1)
                (plank_plank_0 ?plank_p2 ?plank_p2)
                (plank_plank_0 ?plank_p2 ?plank_p3)
                (plank_plank_0 ?plank_p3 ?plank_p1)
                (plank_plank_0 ?plank_p3 ?plank_p2)
                (plank_plank_0 ?plank_p3 ?plank_p3)
    )
    :effect (and
                 (loc_plank_1 loc_plankTarget_Const ?plank_p2)
                (plank_plank_4 ?plank_p1 ?plank_p2)
                (plank_plank_7 ?plank_p3 ?plank_p2)
                (not (loc_plank_0 loc_plankTarget_Const ?plank_p2))
                (not (plank_plank_0 ?plank_p1 ?plank_p2))
                (not (plank_plank_0 ?plank_p3 ?plank_p2))
```

```
                (not (plank_plank_1 ?plank_p1 ?plank_p2))
                (not (plank_plank_1 ?plank_p3 ?plank_p2))
                (not (plank_plank_2 ?plank_p1 ?plank_p2))
                (not (plank_plank_2 ?plank_p3 ?plank_p2))
                (not (plank_plank_3 ?plank_p1 ?plank_p2))
                (not (plank_plank_3 ?plank_p3 ?plank_p2))
                (not (plank_plank_4 ?plank_p3 ?plank_p2))
                (not (plank_plank_5 ?plank_p1 ?plank_p2))
                (not (plank_plank_5 ?plank_p3 ?plank_p2))
                (not (plank_plank_6 ?plank_p1 ?plank_p2))
                (not (plank_plank_6 ?plank_p3 ?plank_p2))
                (not (plank_plank_7 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p1 ?plank_p2))
                (not (plank_plank_8 ?plank_p3 ?plank_p2))
                (not (plank_plank_9 ?plank_p1 ?plank_p2))
                (not (plank_plank_9 ?plank_p3 ?plank_p2))
                (not (clear3_plank_plank_4 ?plank_p1))
                (not (clear3_plank_plank_7 ?plank_p3))
    )
))
```

H.4 Building Structures with Jenga Planks

```
(define (domain Jenga)
(:requirements :strips :typing :equality
:conditional-effects :existential-preconditions :universal-preconditions)
(:types
        loc
        jenga
        gripper
)

(:constants
           loc_jengaTarget_Const - loc
 )

(:predicates
            (loc_jenga_0 ?x - loc ?y - jenga)
            (loc_jenga_1 ?x - loc ?y - jenga)
            (loc_jenga_2 ?x - loc ?y - jenga)
            (jenga_jenga_0 ?x - jenga ?y - jenga)
            (jenga_jenga_1 ?x - jenga ?y - jenga)
            (jenga_jenga_2 ?x - jenga ?y - jenga)
            (jenga_jenga_3 ?x - jenga ?y - jenga)
            (jenga_jenga_4 ?x - jenga ?y - jenga)
            (jenga_jenga_5 ?x - jenga ?y - jenga)
            (jenga_jenga_6 ?x - jenga ?y - jenga)
            (jenga_jenga_7 ?x - jenga ?y - jenga)
            (jenga_jenga_8 ?x - jenga ?y - jenga)
            (gripper_jenga_0 ?x - gripper ?y - jenga)
            (gripper_jenga_1 ?x - gripper ?y - jenga)
            (gripper_jenga_2 ?x - gripper ?y - jenga)
            (clear3_gripper_jenga_1 ?x - gripper)
            (clear3_jenga_jenga_6 ?x - jenga)
            (clear3_jenga_jenga_2 ?x - jenga)
            (clear3_jenga_jenga_3 ?x - jenga)
            (clear3_jenga_jenga_1 ?x - jenga)
            (clear3_jenga_jenga_8 ?x - jenga)
            (clear3_gripper_jenga_2 ?x - gripper)
```

```
            (clear3_jenga_jenga_4 ?x - jenga)
            (clear3_jenga_jenga_7 ?x - jenga)
            (clear3_jenga_jenga_5 ?x - jenga)
)

(:action a1
    :parameters ( ?jenga_p1 - jenga  ?gripper_extra_p1 - gripper )
    :precondition (and
                (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1)
                (gripper_jenga_1 ?gripper_extra_p1 ?jenga_p1)
    )
    :effect (and
                 (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (not (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1))
                (not (loc_jenga_2 loc_jengaTarget_Const ?jenga_p1))
    )
)

(:action a2
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (gripper_jenga_0 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_1 ?gripper_p1)
                (clear3_gripper_jenga_2 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_0 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_1 ?gripper_p1 ?jenga_p1))
                (not (clear3_gripper_jenga_2 ?gripper_p1))
    )
)

(:action a3
    :parameters ( ?jenga_p2 - jenga  ?jenga_p3 - jenga  ?jenga_p1 - jenga
        ?gripper_extra_p1 - gripper )
    :precondition (and
                (not (= ?jenga_p2 ?jenga_p3))
                (not (= ?jenga_p2 ?jenga_p1))
                (not (= ?jenga_p3 ?jenga_p1))
                (loc_jenga_0 loc_jengaTarget_Const ?jenga_p2)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p3)
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p1)
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p3)
                (gripper_jenga_1 ?gripper_extra_p1 ?jenga_p2)
                (jenga_jenga_0 ?jenga_p1 ?jenga_p2)
                (jenga_jenga_0 ?jenga_p3 ?jenga_p2)
                (clear3_jenga_jenga_1 ?jenga_p1)
                (clear3_jenga_jenga_1 ?jenga_p2)
                (clear3_jenga_jenga_1 ?jenga_p3)
                (clear3_jenga_jenga_3 ?jenga_p1)
                (clear3_jenga_jenga_3 ?jenga_p2)
                (clear3_jenga_jenga_3 ?jenga_p3)
    )
    :effect (and
                 (loc_jenga_1 loc_jengaTarget_Const ?jenga_p2)
                (jenga_jenga_1 ?jenga_p1 ?jenga_p2)
                (jenga_jenga_3 ?jenga_p3 ?jenga_p2)
```

```
                (not (loc_jenga_0 loc_jengaTarget_Const ?jenga_p2))
                (not (loc_jenga_2 loc_jengaTarget_Const ?jenga_p2))
                (not (jenga_jenga_0 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_0 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_1 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_2 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_2 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_3 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_4 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_4 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_5 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_5 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_6 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_6 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_7 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_7 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_8 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_8 ?jenga_p3 ?jenga_p2))
                (not (clear3_jenga_jenga_1 ?jenga_p1))
                (not (clear3_jenga_jenga_3 ?jenga_p3))
     )
)

(:action a4
    :parameters ( ?jenga_p2 - jenga  ?jenga_p3 - jenga  ?jenga_p1 - jenga
        ?gripper_extra_p1 - gripper )
    :precondition (and
                (not (= ?jenga_p2 ?jenga_p3))
                (not (= ?jenga_p2 ?jenga_p1))
                (not (= ?jenga_p3 ?jenga_p1))
                (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p2)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p3)
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p2)
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p3)
                (gripper_jenga_1 ?gripper_extra_p1 ?jenga_p1)
                (jenga_jenga_0 ?jenga_p2 ?jenga_p1)
                (jenga_jenga_0 ?jenga_p3 ?jenga_p1)
                (clear3_jenga_jenga_2 ?jenga_p1)
                (clear3_jenga_jenga_2 ?jenga_p2)
                (clear3_jenga_jenga_2 ?jenga_p3)
                (clear3_jenga_jenga_8 ?jenga_p1)
                (clear3_jenga_jenga_8 ?jenga_p2)
                (clear3_jenga_jenga_8 ?jenga_p3)
    )
    :effect (and
                 (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (jenga_jenga_2 ?jenga_p2 ?jenga_p1)
                (jenga_jenga_8 ?jenga_p3 ?jenga_p1)
                (not (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1))
                (not (loc_jenga_2 loc_jengaTarget_Const ?jenga_p1))
                (not (jenga_jenga_0 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_0 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_1 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_1 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_2 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_3 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_3 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_4 ?jenga_p2 ?jenga_p1))
```

```
                (not (jenga_jenga_4 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_5 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_5 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_6 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_6 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_7 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_7 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_8 ?jenga_p2 ?jenga_p1))
                (not (clear3_jenga_jenga_2 ?jenga_p2))
                (not (clear3_jenga_jenga_8 ?jenga_p3))
     )
)

(:action a5
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (gripper_jenga_1 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_2 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_0 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_1 ?gripper_p1 ?jenga_p1))
                (clear3_gripper_jenga_1 ?gripper_p1)
                (not (clear3_gripper_jenga_2 ?gripper_p1))
    )
)

(:action a6
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_1 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_0 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_1 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_2 ?gripper_p1 ?jenga_p1))
                (clear3_gripper_jenga_2 ?gripper_p1)
    )
)

(:action a7
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_1 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_1 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_0 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_2 ?gripper_p1 ?jenga_p1))
                (clear3_gripper_jenga_2 ?gripper_p1)
                (not (clear3_gripper_jenga_1 ?gripper_p1))
    )
)
```

```
(:action a8
    :parameters ( ?jenga_p2 - jenga  ?jenga_p3 - jenga  ?jenga_p1 - jenga
        ?gripper_extra_p1 - gripper )
    :precondition (and
                (not (= ?jenga_p2 ?jenga_p3))
                (not (= ?jenga_p2 ?jenga_p1))
                (not (= ?jenga_p3 ?jenga_p1))
                (loc_jenga_0 loc_jengaTarget_Const ?jenga_p2)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p3)
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p1)
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p3)
                (gripper_jenga_1 ?gripper_extra_p1 ?jenga_p2)
                (jenga_jenga_0 ?jenga_p1 ?jenga_p2)
                (jenga_jenga_0 ?jenga_p3 ?jenga_p2)
                (clear3_jenga_jenga_4 ?jenga_p1)
                (clear3_jenga_jenga_4 ?jenga_p2)
                (clear3_jenga_jenga_4 ?jenga_p3)
                (clear3_jenga_jenga_6 ?jenga_p1)
                (clear3_jenga_jenga_6 ?jenga_p2)
                (clear3_jenga_jenga_6 ?jenga_p3)
    )
    :effect (and
                 (loc_jenga_1 loc_jengaTarget_Const ?jenga_p2)
                (jenga_jenga_4 ?jenga_p1 ?jenga_p2)
                (jenga_jenga_6 ?jenga_p3 ?jenga_p2)
                (not (loc_jenga_0 loc_jengaTarget_Const ?jenga_p2))
                (not (loc_jenga_2 loc_jengaTarget_Const ?jenga_p2))
                (not (jenga_jenga_0 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_0 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_1 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_1 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_2 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_2 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_3 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_3 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_4 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_5 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_5 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_6 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_7 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_7 ?jenga_p3 ?jenga_p2))
                (not (jenga_jenga_8 ?jenga_p1 ?jenga_p2))
                (not (jenga_jenga_8 ?jenga_p3 ?jenga_p2))
                (not (clear3_jenga_jenga_4 ?jenga_p1))
                (not (clear3_jenga_jenga_6 ?jenga_p3))
    )
)

(:action a9
    :parameters ( ?jenga_p2 - jenga  ?jenga_p3 - jenga  ?jenga_p1 - jenga
            ?gripper_extra_p1 - gripper )
    :precondition (and
                (not (= ?jenga_p2 ?jenga_p3))
                (not (= ?jenga_p2 ?jenga_p1))
                (not (= ?jenga_p3 ?jenga_p1))
                (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p2)
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p3)
```

```
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p2)
                (gripper_jenga_0 ?gripper_extra_p1 ?jenga_p3)
                (gripper_jenga_1 ?gripper_extra_p1 ?jenga_p1)
                (jenga_jenga_0 ?jenga_p2 ?jenga_p1)
                (jenga_jenga_0 ?jenga_p3 ?jenga_p1)
                (clear3_jenga_jenga_5 ?jenga_p1)
                (clear3_jenga_jenga_5 ?jenga_p2)
                (clear3_jenga_jenga_5 ?jenga_p3)
                (clear3_jenga_jenga_7 ?jenga_p1)
                (clear3_jenga_jenga_7 ?jenga_p2)
                (clear3_jenga_jenga_7 ?jenga_p3)
    )
    :effect (and
                 (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (jenga_jenga_5 ?jenga_p2 ?jenga_p1)
                (jenga_jenga_7 ?jenga_p3 ?jenga_p1)
                (not (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1))
                (not (loc_jenga_2 loc_jengaTarget_Const ?jenga_p1))
                (not (jenga_jenga_0 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_0 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_1 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_1 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_2 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_2 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_3 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_3 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_4 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_4 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_5 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_6 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_6 ?jenga_p3 ?jenga_p1))
                (not (jenga_jenga_7 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_8 ?jenga_p2 ?jenga_p1))
                (not (jenga_jenga_8 ?jenga_p3 ?jenga_p1))
                (not (clear3_jenga_jenga_5 ?jenga_p2))
                (not (clear3_jenga_jenga_7 ?jenga_p3))
    )
)

(:action a10
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_1 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_1 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_0 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_2 ?gripper_p1 ?jenga_p1))
                (clear3_gripper_jenga_2 ?gripper_p1)
                (not (clear3_gripper_jenga_1 ?gripper_p1))
    )
)

(:action a11
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (gripper_jenga_0 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_1 ?gripper_p1)
```

```
                (clear3_gripper_jenga_2 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_0 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_1 ?gripper_p1 ?jenga_p1))
                (not (clear3_gripper_jenga_2 ?gripper_p1))
    )
)

(:action a12
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (gripper_jenga_1 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_2 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_0 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_1 ?gripper_p1 ?jenga_p1))
                (clear3_gripper_jenga_1 ?gripper_p1)
                (not (clear3_gripper_jenga_2 ?gripper_p1))
    )
)

(:action a13
    :parameters ( ?jenga_p1 - jenga  ?gripper_extra_p1 - gripper )
    :precondition (and
                (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1)
                (gripper_jenga_1 ?gripper_extra_p1 ?jenga_p1)
    )
    :effect (and
                 (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (not (loc_jenga_0 loc_jengaTarget_Const ?jenga_p1))
    )
)

(:action a14
    :parameters ( ?jenga_p1 - jenga  ?gripper_p1 - gripper )
    :precondition (and
                (loc_jenga_1 loc_jengaTarget_Const ?jenga_p1)
                (gripper_jenga_2 ?gripper_p1 ?jenga_p1)
                (clear3_gripper_jenga_1 ?gripper_p1)
    )
    :effect (and
                 (gripper_jenga_0 ?gripper_p1 ?jenga_p1)
                (not (gripper_jenga_1 ?gripper_p1 ?jenga_p1))
                (not (gripper_jenga_2 ?gripper_p1 ?jenga_p1))
                (clear3_gripper_jenga_2 ?gripper_p1)
    )
))
```

H.5 Packing a Box

```
(define (domain Packing)
(:requirements :strips :typing :equality :conditional-effects
↪  :existential-preconditions :universal-preconditions)
```

```
(:types
        can
        gripper
        surface
)

(:predicates
        (gripper_can_0 ?x - gripper ?y - can)
        (gripper_can_1 ?x - gripper ?y - can)
        (gripper_can_2 ?x - gripper ?y - can)
        (can_surface_0 ?x - can ?y - surface)
        (can_surface_1 ?x - can ?y - surface)
        (clear3_gripper_can_1 ?x - gripper)
        (clear3_gripper_can_2 ?x - gripper)
)

(:action a1
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper )
:precondition (and
        (gripper_can_2 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_1 ?gripper_p1)
)
:effect (and
         (gripper_can_1 ?gripper_p1 ?can_p1)
        (not (gripper_can_0 ?gripper_p1 ?can_p1))
        (not (gripper_can_2 ?gripper_p1 ?can_p1))
        (clear3_gripper_can_2 ?gripper_p1)
        (not (clear3_gripper_can_1 ?gripper_p1))
 )
)

(:action a2
:parameters ( ?can_p1 - can  ?gripper_p1 - gripper )
:precondition (and
        (gripper_can_0 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_1 ?gripper_p1)
        (clear3_gripper_can_2 ?gripper_p1)
)
:effect (and
         (gripper_can_2 ?gripper_p1 ?can_p1)
        (not (gripper_can_0 ?gripper_p1 ?can_p1))
        (not (gripper_can_1 ?gripper_p1 ?can_p1))
        (not (clear3_gripper_can_2 ?gripper_p1))
 )
)

(:action a3
:parameters ( ?can_p1 - can  ?surface_extra_p1 - surface  ?gripper_p1 -
↪  gripper )
:precondition (and
        (can_surface_1 ?can_p1 ?surface_extra_p1)
        (gripper_can_2 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_1 ?gripper_p1)
)
:effect (and
         (gripper_can_0 ?gripper_p1 ?can_p1)
        (not (gripper_can_1 ?gripper_p1 ?can_p1))
        (not (gripper_can_2 ?gripper_p1 ?can_p1))
        (clear3_gripper_can_2 ?gripper_p1)
```

```
 )
)

(:action a4
:parameters ( ?can_p1 - can  ?surface_extra_p1 - surface  ?gripper_p1 -
↪  gripper )
:precondition (and
        (can_surface_1 ?can_p1 ?surface_extra_p1)
        (gripper_can_1 ?gripper_p1 ?can_p1)
        (clear3_gripper_can_2 ?gripper_p1)
)
:effect (and
         (gripper_can_2 ?gripper_p1 ?can_p1)
        (not (gripper_can_0 ?gripper_p1 ?can_p1))
        (not (gripper_can_1 ?gripper_p1 ?can_p1))
        (clear3_gripper_can_1 ?gripper_p1)
        (not (clear3_gripper_can_2 ?gripper_p1))
 )
)

(:action a5
:parameters ( ?can_p1 - can  ?gripper_extra_p1 - gripper  ?surface_p1 -
↪  surface )
:precondition (and
        (can_surface_0 ?can_p1 ?surface_p1)
        (gripper_can_1 ?gripper_extra_p1 ?can_p1)
)
:effect (and
         (can_surface_1 ?can_p1 ?surface_p1)
        (not (can_surface_0 ?can_p1 ?surface_p1))
 )
))
```

I Code as Policies Evaluation Prompts

I.1 Delivering Items in a Cafe

Domain-specific Prompt for Robot Actions

```
jointnames = (["torso_lift_joint", "shoulder_pan_joint", "shoulder_lift_joint",
            "upperarm_roll_joint", "elbow_flex_joint", "forearm_roll_joint",
            "wrist_flex_joint", "wrist_roll_joint"])

# define function: openGripper(robot)
def openGripper(robot):
    taskmanip = interfaces.TaskManipulation(robot)
    with robot:
        taskmanip.ReleaseFingers()
    robot.WaitForController(0)

# define function: grab_success_flag = grab_object(object_name)
def grab_object(object_to_grab):
    o = env.GetKinBody(object_to_grab)
    ot = o.GetTransform()
    robot_t = robot.GetLink("wrist_roll_link").GetTransform()
    euclidean_distance = np.linalg.norm(robot_t[:3,3]-ot[:3,3])
    obj_type = object_to_grab.split("_")[0]
    grab_range = [0.20,0.24]
    if euclidean_distance<grab_range[1] and euclidean_distance>grab_range[0]:
        robot.Grab(o)
        return True
```

```
    else:
        print("object out of grasp range")
        return False

# define function: un_grab(object_name)
def un_grab(object_name):
    robot.ReleaseAllGrabbed()

# define function: current_pose_of_object =
↪  get_current_pose_of_object(object_name)
def get_current_pose_of_object(object_name):
    obj = env.GetKinBody(object_name)
    obj_T = obj.GetTransform()
    obj_pose = get_pose_from_transform(obj_T)

    return obj_pose

# define function: robot_ik = get_ik(pose)
def get_ik(pose):
    end_effector_solution = get_transform_from_pose(pose)
    activate_manip_joints()
    # filter_option = IkFilterOptions.CheckEnvCollisions
    filter_option = IkFilterOptions.IgnoreEndEffectorCollisions

    with env:
        ikmodel = databases.inversekinematics.InverseKinematicsModel(robot,
                        iktype=IkParameterization.Type.Transform6D)

        if not ikmodel.load():
            ikmodel.autogenerate()
        try:
            solutions = ikmodel.manip.FindIKSolutions(end_effector_solution,
            ↪  filter_option)
        except:
            print("error")

    if len(solutions) == 0:
        print("NO IKs found, Probably Un-reachable transform")

    if len(solutions) > 0:
        if len(solutions) == 1:
            i = 0
        else:
            i = np.random.randint(0,len(solutions))
    else:
        return []

    return solutions[i]

# define function: go_to_gripper_pose(pose)
def go_to_gripper_pose(pose):
    activate_manip_joints()
    robot.SetActiveDOFValues(pose)

# define function: go_to_base_pose(pose)
def go_to_base_pose(pose):
    activate_base_joints()
    robot.SetActiveDOFValues(pose)

# define function: current_arm_joint_values =
↪  get_current_arm_joint_values_of_robot()
def get_current_arm_joint_values_of_robot():
    activate_manip_joints()
    return robot.GetActiveDOFValues()
```

```
# define function: current_base_joint_values = get_current_base_joint_values()
def get_current_base_joint_values_of_robot():
    activate_base_joints
    return robot.GetActiveDOFValues()

# define function: pose = get_pose_from_transform(transform)
def get_pose_from_transform(transform):
    quat = poseFromMatrix(transform)[:4]
    eul = axisAngleFromQuat(quat)
    pose = []
    pose.extend(poseFromMatrix(transform)[4:])
    pose.extend(eul)

    return pose

# define function: collision_flag = collision_check(object_name)
def collision_check(object_name):
    collision_flag = False
    obj = env.GetKinBody(object_name)

    for obj2 in env.GetBodies():
        if obj != obj2:
            collision = env.CheckCollision(obj,obj2)
            if collision:
                if (obj2 == robot and obj in robot.GetGrabbed()) or
                (obj == robot and obj2 in robot.GetGrabbed()):
                    collision = False
                    continue
                collision_flag = True
                break
    return collision_flag

# define function: activate_base_joints()
def activate_base_joints():
    robot.SetActiveDOFs([], DOFAffine.X | DOFAffine.Y | DOFAffine.RotationAxis)

# define function: activate_arm_joints()
def activate_manip_joints():
    robot.SetActiveDOFs([robot.GetJoint(name).GetDOFIndex() for name in
    ↪  jointnames])

# define function: grasp_pose_for_object = generate_grasp_pose(object_to_grab)
def generate_grasp_pose(object_to_grab):
    activate_manip_joints()
    rot_Z = matrixFromAxisAngle([0, 0, -np.pi/2])
    gripper_offset = -0.04
    world_T_obj = env.GetKinBody(object_to_grab).GetTransform()

    rot_ang = np.random.uniform(low = -np.pi, high = np.pi)
    obj_T_gripper = matrixFromPose([1, 0, 0, 0, gripper_offset, 0, 0.2/2.0])
    rot_mat = matrixFromAxisAngle([0, 0, rot_ang])

    wrist_roll_pose = robot.GetLink("wrist_roll_link").GetTransform()
    gripper_pose = robot.GetLink("gripper_link").GetTransform()
    wrist_pose_wrt_gripper = np.matmul(np.linalg.inv(gripper_pose),
    ↪  wrist_roll_pose)

    grasp_T = world_T_obj.dot(rot_mat).dot(rot_Z).dot(obj_T_gripper)
    grasp_T = np.matmul(grasp_T,wrist_pose_wrt_gripper)

    grasp_pose = get_pose_from_transform(grasp_T)

    return grasp_pose
```

```
# define function: transform = get_tranform_from_pose(pose)
def get_transform_from_pose(pose):
    quat = quatFromAxisAngle(pose[3:])
    pos = pose[:3]
    pose = []
    pose.extend(quat)
    pose.extend(pos)
    transform = matrixFromPose(pose)

    return transform


# define function: putdown_pose_for_object =
            generate_put_down_pose(object_name,target_name)
def generate_put_down_pose(object_name,target_name):
    target_id = int(target_name.split("_")[1])
    obj = env.GetKinBody(object_name)
    can_radius = 0.03
    sampling_range = [[-0.175,0.175], [-0.175,0.175]]
    target_t = env.GetKinBody(target_name).GetTransform()

    x = np.random.uniform(low=sampling_range[0][0]+can_radius,
        high=sampling_range[0][1]-can_radius)
    y = np.random.uniform(low=sampling_range[1][0]+can_radius,
        high=sampling_range[1][1]-can_radius)



    target_id = int(target_name.split("_")[1])
    if target_id == 0:
        z = 0
        rot_z = np.eye(4)
    else:
        z = (0.1/2.0) + 0.005
        rot_angle = np.random.uniform(low=-np.pi,high=np.pi)
        rot_z = matrixFromAxisAngle([0,0,rot_angle])



    t = matrixFromPose([1,0,0,0,x,y,z])
    t = target_t.dot(t)
    t = t.dot(rot_z)

    rot_Z = matrixFromAxisAngle([0, 0, -np.pi/2])
    gripper_offset = -0.04
    world_T_obj = t

    rot_ang = np.random.uniform(low = -np.pi, high = np.pi)
    obj_T_gripper = matrixFromPose([1, 0, 0, 0, gripper_offset, 0, 0.2/2.0])
    rot_mat = matrixFromAxisAngle([0, 0, rot_ang])

    wrist_roll_pose = robot.GetLink("wrist_roll_link").GetTransform()
    gripper_pose = robot.GetLink("gripper_link").GetTransform()
    wrist_pose_wrt_gripper = np.matmul(np.linalg.inv(gripper_pose),
    ↪  wrist_roll_pose)

    grasp_T = world_T_obj.dot(rot_mat).dot(rot_Z).dot(obj_T_gripper)
    grasp_T = np.matmul(grasp_T,wrist_pose_wrt_gripper)

    pose = get_pose_from_transform(grasp_T)

    return pose
```

```
# define function: base_pose_around_surface =
        generate_base_pose_around_surface(target_name)
def generate_base_pose_around_surface(target_name):
    target_id = int(target_name.split("_")[1])
    diff = 0.35
    x_dim = 0.45
    x_offset = -diff-x_dim
    y_offset = 0
    target_t = env.GetKinBody(target_name).GetTransform()
    diff_translation_matrix = matrixFromPose([1,0,0,0,x_offset,y_offset,0])

    if target_id == 0:
        rot_angle = (2*np.pi) / 4.0
    else:
        rot_angle = np.random.uniform(low=-np.pi,high=np.pi)

    rot_Z = matrixFromAxisAngle([0, 0, -np.pi/2])
    rot_mat = matrixFromAxisAngle([0,0,rot_angle])
    t = np.eye(4)
    t = target_t.dot(rot_mat).dot(rot_Z).dot(diff_translation_matrix)


    _x = t[0,3]
    _y = t[1,3]
    _yaw = axisAngleFromRotationMatrix(t[:3,:3])[-1]
    pose = [_x,_y,_yaw]

    return pose

# define function: tuck_arm()
def tuck_arm():
    activate_manip_joints()
    dof_values = [0, 1.32, 1.4, -0.2, 1.72, 0, 1.3599999999999999, 0.0]

    robot.SetActiveDOFValues(dof_values)

grabbed_armTuckDOFs = [0, 1.32, 1.4, -0.2, 1.72, 0, 1.3599999999999999, 0.0]

# example: tuck the robot arm to a tuck pose.
tuck_arm()


# example: go to "surface_1"
p = generate_base_pose_around_surface("surface_1")
go_to_base_pose(p)

# example: grab can_1 which is currently on "surface_0".
p = generate_base_pose_around_surface("surface_2")
go_to_base_pose(p)
p_g = generate_grasp_pose("can_1")
ik = get_ik(p_g)
if len(ik) != 0:
    go_to_gripper_pose(ik)
grabbed_flag = grab_object("can_1")

# example: pickup a grabbed object "can_2", which is on "surface_1"
                when robot is near "surface_1".
p_g = generate_grasp_pose("can_2")
ik = get_ik(p_g)
if len(ik) != 0:
    go_to_gripper_pose(ik)
grabbed_flag = grab_object("can_2")
if grabbed_flag:
    tuck_arm()
```

```
# example: put down "can_1", which is already being held by the
            robot gripper on "surface_1".
p = generate_base_pose_around_surface("surface_1")
go_to_base_pose(p)
p_g = generate_put_down_pose("can_1","surface_1")
ik = get_ik(p_g)
if len(ik) != 0:
    go_to_gripper_pose(ik)
un_grab("can_1")
tuck_arm()

# example: check collisions for "can_1".
collision_flag = collision_check("can_1")

# example: pickup can_1, which is on surface_0 and place it on surface_2
p = generate_base_pose_around_surface("surface_0")
go_to_base_pose(p)
p_g = generate_grasp_pose("can_1")
ik = get_ik(p_g)
if len(ik) != 0:
    go_to_gripper_pose(ik)
grabbed_flag = grab_object("can_1")
if grabbed_flag:
    tuck_arm()
p = generate_base_pose_around_surface("surface_2")
go_to_base_pose(p)
p_g = generate_put_down_pose("can_1","surface_2")
ik = get_ik(p_g)
if len(ik) != 0:
    go_to_gripper_pose(ik)
un_grab("can_1")
tuck_arm()
```

Prompt for Delivering 1 Item

```
Using the functions given above solve the following problem: #
There are 2 cans, can_1, and can_2, and three surfaces, surface_1, surface_2,
and surface_0 in the environment. In the initial state, both the cans, can_1 and
can_2 are on the surface - surface_2. the function returns True if robot has
successfully picked up both cans from surface_2 and placed can_1 on surface_0 and
can_2 on surface_1. success_flag is True if the function succeeds in task and
there is no object in collision with any other object in the environemnt.
# The signature for the function required is:
success_flag = place_cans_on_goal_surfaces().
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

Prompt for Delivering 2 Item

```
Using the functions given above solve the following problem: #
There are 2 cans, can_1, and can_2, and three surfaces, surface_1, surface_2,
and surface_0 in the environment. In the initial state, can_1 is on surface_2
and can_2 is on surface_1. the function returns True if robot has successfully
↪  picked
up can_1 from surface_2 and placed it on surface_1, and also has picked can_2
↪  from
surface_1 and placed it on surface_0. success_flag is True if the function
↪  succeeds
```

```
in task and there is no object in collision with any other object in the
↪  environemnt.
# The signature for the function required is:
success_flag = place_cans_on_goal_surfaces().
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

I.2 Building Keva Structures

Domain-specific Prompt for Robot Actions

```
# define function: success_flag = tuck_arm()
def tuck_arm(arm="left"):
    release(arm)
    if arm == "left":
        solution = left_arm_tuck_DOFs
    elif arm == "right":
        solution = right_arm_tuck_DOFs

    try:
        robot.SetActiveDOFValues(solution)
    except:
        pass

    openGripper()
    return True

# define function: grab(object_name)
def grab(obj,arm="left"):
    o = env.GetKinBody(obj)
    if arm == "left":
        gripper_link = left_gripper_link
    elif arm == "right":
        gripper_link = right_gripper_link

    robot_t = gripper_link.GetTransform()
    ot = o.GetTransform()
    euclidean_distance = np.linalg.norm(robot_t[:3,3]-ot[:3,3])
    if euclidean_distance<grab_range[1] and euclidean_distance>grab_range[0]:
        robot.Grab(o)
    else:
        print("object out of grasp range")

# define function: release(object_name)
def release(object_name="object_name",arm="left"):
    robot.ReleaseAllGrabbed()

# define function: collision_flag = collision_check(object_name)
def collision_check(object_name):
    collision_flag = False
    obj = env.GetKinBody(object_name)

    for obj2 in env.GetBodies():
        if obj != obj2:
            collision = env.CheckCollision(obj,obj2)
            if collision:
                if (obj2 == robot and obj in robot.GetGrabbed()) or
                (obj == robot and obj2 in robot.GetGrabbed()):
                    collision = False
                    continue
                collision_flag = True
                break
```

```
    return collision_flag

# define function: robot_ik = get_ik_solutions(pose)
def get_ik_solutions(end_effector_solution,robot_param="left"):
    end_effector_solution = get_transform_from_pose(end_effector_solution)
    current_state = robot.GetActiveDOFValues()
    collision = True
    if robot_param not in manipulator_groups:
        robot_param = "left"

    if robot_param == "left":
        ik_solver = left_ik_solver
    elif robot_param == "right":
        ik_solver = right_ik_solver

    ik_count = 0

    required_T = np.linalg.pinv(robot_init_transform).dot(end_effector_solution)
    pose = get_pose_from_transform(required_T)
    pos = pose[:3]
    orn = quatFromAxisAngle(pose[3:])

    while collision:
        seed_state = [np.random.uniform(-3.14, 3.14)] *
        ↪  ik_solver.number_of_joints
        joint_values = ik_solver.get_ik(seed_state,
                                                pos[0], pos[1], pos[2],
                                                orn[1], orn[2], orn[3], orn[0]
                                                )

        if joint_values is not None:
            robot.SetActiveDOFValues(joint_values)
            if collision_check(str(robot.GetName())):
                collision=True
            else:
                collision = False
        else:
            print("no joint_values")
            joint_values = []
            break

    robot.SetActiveDOFValues(current_state)
    return joint_values

# define function: pose = get_pose_from_transform(transform)
def get_pose_from_transform(transform):
    quat = poseFromMatrix(transform)[:4]
    eul = axisAngleFromQuat(quat)
    pose = []
    pose.extend(poseFromMatrix(transform)[4:])
    pose.extend(eul)

    return pose

# define function: go_to_gripper_pose(pose)
def go_to_gripper_pose(pose,arm="left"):
    robot.SetActiveDOFValues(pose)

# define function: current_pose_of_object =
↪  get_current_pose_of_object(object_name)
def get_current_pose_of_object(object_name):
    obj = env.GetKinBody(object_name)
    obj_T = obj.GetTransform()
    obj_pose = get_pose_from_transform(obj_T)
```

```
    return obj_pose

# define function: transform = get_tranform_from_pose(pose)
def get_transform_from_pose(pose):
    quat = quatFromAxisAngle(pose[3:])
    pos = pose[:3]
    pose = []
    pose.extend(quat)
    pose.extend(pos)
    transform = matrixFromPose(pose)

    return transform

# define function: p = sample_grasp_pose(object_name)
def sample_grasp_pose(object_name):
    world_T_obj = env.GetKinBody(object_name).GetTransform()

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: set_plank(object_name)
def set_plank(plank_name):
    plank = env.GetKinBody(plank_name)
    x_offsets=[0.15,0.45]
    y_offsets=[0.65,0.05]

    while True:
        t = np.eye(4)
        t[0,3] = np.random.uniform(low = -0.45+x_offsets[0], high =
        ↪  0.45-x_offsets[1])
        t[1,3] = np.random.uniform(low = -0.45+y_offsets[0], high =
        ↪  0.45-y_offsets[1])
        t[2,3] = 0.0135

        t1 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])
        t = t.dot(t1)
        plank.SetTransform(t)
        t2 = orpy.matrixFromAxisAngle([0,-np.pi/2, 0])
        plank.SetTransform(t.dot(t2))
        if not(collision_check(plank_name)):
            break

# define function: p = get_goal_put_down_pose_for_plank_1()
def get_goal_put_down_pose_for_plank_1():
    world_T_obj = env.GetKinBody("goalLoc_1").GetTransform()

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])
```

```
    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: p = sample_plank_on_left_of_other_plank(plank_1,plank_2)
def sample_plank_on_left_of_other_plank(plank_1,plank_2):
    plank_1 = env.GetKinBody(plank_1)
    relative_transform = get_transform_from_pose([0, 0, 0.077474999073727457, 0,
    ↪  0, 0])

    required_t = plank_1.GetTransform().dot(relative_transform)

    world_T_obj = required_t

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: p = sample_plank_on_right_of_other_plank(plank_1,plank_2)
def sample_plank_on_right_of_other_plank(plank_1,plank_2):
    plank_1 = env.GetKinBody(plank_1)
    relative_transform =
                get_transform_from_pose([0, 0, -0.077474999073727457, 0, 0, 0])

    required_t = plank_1.GetTransform().dot(relative_transform)

    world_T_obj = required_t

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: p =
sample_plank_horizontally_on_top_of_other_two_vertical_planks
                ("plank_1","plank_2","plank_3")
def sample_plank_horizontally_on_top_of_other_two_vertical_planks
        (plank_1,plank_2,plank_3):
    plank_1 = env.GetKinBody(plank_1)
    relative_transform = get_transform_from_pose([0.059907747268674261, 0,
```

```
            0.04373009875416578, 0, 1.5707963267948959, 0])
    required_t = plank_1.GetTransform().dot(relative_transform)

    world_T_obj = required_t

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: p =
sample_plank_horizontally_on_top_of_other_two_horizontal_planks_on_
                left_side("plank_1","plank_2","plank_3")
def sample_plank_horizontally_on_top_of_other_two_horizontal_planks_on_
                left_side(plank_1,plank_2,plank_3):
    plank_1 = env.GetKinBody(plank_1)
    relative_transform = get_transform_from_pose([0.04, -0.023439999999999989,
    ↪  0.04,
            0, 1.5707963267948963, 0])
    required_t = plank_1.GetTransform().dot(relative_transform)

    world_T_obj = required_t

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)
    return pose

# define function: p =
sample_plank_horizontally_on_top_of_other_two_horizontal_planks_on_
            right_side("plank_1","plank_2","plank_3")
def sample_plank_horizontally_on_top_of_other_two_horizontal_planks_on_
            right_side(plank_1,plank_2,plank_3):
    plank_1 = env.GetKinBody(plank_1)
    relative_transform = get_transform_from_pose([-0.04, -0.023439999999999935,
            0.04, 0, 1.5707963267948968, 0])

    required_t = plank_1.GetTransform().dot(relative_transform)

    world_T_obj = required_t

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
```

```
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: p =
sample_plank_perpendicularly_on_top_of_a_horizontal_plank_on_its_left_side
    (plank_1,plank_2)
def sample_plank_perpendicularly_on_top_of_a_horizontal_plank_on_its_
            left_side(plank_1,plank_2):
    plank_1 = env.GetKinBody(plank_1)
    relative_transform = get_transform_from_pose([-0.033744900319561663,
        -4.9568220254682954e-16, 0.06650477647781355 4, 1.0368663869534288e-14,
            -1.5707963267948974, -9.9776971655314691e-15])

    required_t = plank_1.GetTransform().dot(relative_transform)

    world_T_obj = required_t

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)
    return pose

# define function: p =
sample_plank_perpendicularly_on_top_of_a_horizontal_plank_on_its_
            right_side(plank_1,plank_2)
def sample_plank_perpendicularly_on_top_of_a_horizontal_plank_on_its_
            right_side(plank_1,plank_2):
    plank_1 = env.GetKinBody(plank_1)
    relative_transform = get_transform_from_pose([0.04373009875416739,
        -4.7544147175681199e-16, 0.06633448600769040 2, 1.0368663869534288e-14,
        -1.5707963267948974, -9.9776971655314691e-15])

    required_t = plank_1.GetTransform().dot(relative_transform)

    world_T_obj = required_t

    world_T_robot = get_transform_from_pose(robot.GetActiveDOFValues())
    robot_T_world = np.linalg.inv(world_T_robot)

    obj_T_robot = np.eye(4)
    obj_T_robot[1,3]= grasping_offset

    t1 = orpy.matrixFromAxisAngle([ 0, -np.pi/2, 0])
    t2 = orpy.matrixFromAxisAngle([-np.pi/2, 0, 0])

    obj_T_robot = np.matmul(np.matmul(obj_T_robot,t1),t2)
    t = np.matmul(world_T_obj,obj_T_robot)
```

```
    pose = get_pose_from_transform(t)
    return pose

# example: place plank_1 and plank_2 parallel to each other
set_plank("plank_1")
p = sample_grasp_pose("plank_1")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
grab("plank_1")
p = get_goal_put_down_pose_for_plank_1()
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
release()
set_plank("plank_2")
p = sample_grasp_pose("plank_2")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
grab("plank_2")
p = sample_plank_on_left_of_other_plank("plank_1","plank_2")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
release()

# example: place plank_3 horizontally on top of vertically placed plank_1 and
↪  plank_2
set_plank("plank_3")
p = sample_grasp_pose("plank_3")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
grab("plank_3")
p = sample_plank_on_top_of_other_planks_in_pose_1("plank_1", "plank_2",
↪  "plank_3")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
release()

# example: place plank_2 vertically on plank_1 on the left
set_plank("plank_2")
p = sample_grasp_pose("plank_2")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
grab("plank_2")
p = sample_plank_perpendicularly_on_top_of_a_horizontal_plank_on_its_
        left_side("plank_1","plank_2")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
release()

# example: place plank_2 vertically on plank_1 on the right
set_plank("plank_2")
p = sample_grasp_pose("plank_2")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
grab("plank_2")
p = sample_plank_perpendicularly_on_top_of_a_horizontal_plank_on_its_
```

```
        right_side("plank_1","plank_2")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
release()

# example: place plank_3 horizontally on top of horizontally placed
        plank_1 and plank_2
set_plank("plank_3")
p = sample_grasp_pose("plank_3")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
grab("plank_3")
p = sample_plank_on_top_of_other_planks_in_pose_2("plank_1", "plank_2",
↪  "plank_3")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
release()

# example: place plank_3 horizontally on top of horizontally placed
            plank_1 and plank_2
set_plank("plank_3")
p = sample_grasp_pose("plank_3")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
grab("plank_3")
p = sample_plank_on_top_of_other_planks_in_pose_3("plank_1", "plank_2",
↪  "plank_3")
ik = get_ik_solutions(p)
if len(ik) > 0:
    go_to_gripper_pose(ik)
release()

# example: tuck the robot arm to a tuck pose.
tuck_arm()

# example: check collisions for plank_1
collision_flag = collision_check("plank_1")
```

Prompt for Building Structure 1

```
Using the functions given above solve the following problem: #
There are 3 planks in the environment. The function should arrange these planks
↪  such
that all the planks are kept at their corresponding goal locations. plank_1 being
↪  at
goal location corresponding to itself. I.e., plank_1 and plank_2 are placed
↪  vertically
on the table parallel from each other and plank_3 should be placed horizontally
↪  on
both of these planks. The function returns true if the planks are placed in a
collision-free configuration, otherwise returns false.
# The signature for the function required is:
success_flag = place_planks_in_a_goal_structure().
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

Prompt for Building Structure 2

```
Using the functions given above solve the following problem: #
There are 4 planks in the environment. The function should arrange these planks
such that all the planks are kept at their corresponding
goal locations. plank_1 and  plank_2 are placed horizontally on top of the table,
↪  with
plank_1 being at goal location corresponding to itself and plank_2 placed
↪  parallelly
on left of plank_1. plank_4  is placed on the plank_2 and plank_1 horizontally on
its side on the left and plank_3 is placed on the plank_2 and plank_1
↪  horizontally on
its side on the right. plank_3  is also placed parallelly on the left of the
↪  plank_4.
# The signature for the function required is:
success_flag = place_planks_in_a_goal_structure().
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

Prompt for Building Structure 3

```
Using the functions given above solve the following problem: #
There are 6 planks in the environment. The function should arrange these planks
such that all the planks are kept at their corresponding goal locations such that
↪  when
kept together, the planks create a structure of a 2D drawing of a house.
# The signature for the function required is:
success_flag = place_planks_in_a_goal_structure().
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

Prompt for Building Structure 4

```
Using the functions given above solve the following problem: #
There are 6 planks in the environment. The function should arrange these planks
such that all the planks are kept at their corresponding goal locations.
plank_1 and plank_2 are placed horizontally on top of the table, with plank_1
↪  being
at goal location corresponding to itself and plank_2 placed parallelly on left of
plank_1. plank_3 is placed on the plank_2 and plank_1 horizontally on its side on
the left and plank_4 is placed on the plank_2 and plank_1 horizontally on its
side on the right. plank_4 is also placed parallelly on the left of the plank_3.
plank_4 and plank_5 are kept parallelly and vertically on top of plank_3, with
↪  plank_4
kept on right side and plank_5 placed on left side of plank_4. plank_6 is placed
horizontally on plank_4 and plank_5.
# The signature for the function required is:
success_flag = place_planks_in_a_goal_structure().
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

I.3 Packing a Box

Domain-specific Prompt for Robot Actions

```

# define function: grab_object(object_to_grab)
def grab_object(object_to_grab):
    o = env.GetKinBody(object_to_grab)
    robot.Grab(o)
```

```
# define function: un_grab(object_name)
def un_grab(object_name):
    robot.ReleaseAllGrabbed()

# define function: grasp_pose_for_object = generate_grasp_pose(object_to_grab)
def generate_grasp_pose(object_to_grab):
    world_T_obj = env.GetKinBody(object_to_grab).GetTransform()
    obj_T_robot = np.eye(4)
    obj_T_robot[2,3] = 0.21
    t1 = matrixFromAxisAngle([0, 0, -np.pi/4.0])
    obj_T_robot = obj_T_robot.dot(t1)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: putdown_pose_for_object = generate_putdown_pose
                    (object_name,target_name)
def generate_putdown_pose(object_name,target_name):
    obj_dims = [0.0325,0.0325]
    droparea_dims = 0.075

    drop = env.GetKinBody(target_name)
    drop_t = drop.GetTransform()

    x_edge_offset = abs(droparea_dims-obj_dims[0])
    y_edge_offset = abs(droparea_dims-obj_dims[1])

    x = np.random.uniform(low=-x_edge_offset,high=x_edge_offset)
    y = np.random.uniform(low=-y_edge_offset,high=y_edge_offset)
    z = 0.001



    world_T_obj = matrixFromPose([1,0,0,0,x,y,z])
    world_T_obj = drop_t.dot(world_T_obj)

    obj_T_robot = np.eye(4)
    obj_T_robot[2,3] = 0.21
    t1 = matrixFromAxisAngle([0, 0, -np.pi/4.0])
    obj_T_robot = obj_T_robot.dot(t1)
    t = np.matmul(world_T_obj,obj_T_robot)
    pose = get_pose_from_transform(t)

    return pose

# define function: current_pose_of_object =
↪  get_current_pose_of_object(object_name)
def get_current_pose_of_object(object_name):
    obj = env.GetKinBody(object_name)
    obj_T = obj.GetTransform()
    obj_pose = get_pose_from_transform(obj_T)

    return obj_pose

# define function: robot_ik = get_ik(pose)
def get_ik(pose):
    return pose

# define function: go_to_pose(pose)
def go_to_pose(pose):
    ik = get_ik(pose)
    robot.SetActiveDOFValues(ik)
```

```
# define function: current_joint_values = get_current_joint_values_of_robot()
def get_current_joint_values_of_robot():
    return robot.GetActiveDOFValues()

# define function: pose = get_pose_from_transform(transform)
def get_pose_from_transform(transform):
    quat = poseFromMatrix(transform)[:4]
    eul = axisAngleFromQuat(quat)
    pose = []
    pose.extend(poseFromMatrix(transform)[4:])
    pose.extend(eul)

    return pose

# define function: collision_flag = collision_check(object_name)
def collision_check(object_name):
    collision_flag = False
    obj = env.GetKinBody(object_name)

    for obj2 in env.GetBodies():
        if obj != obj2:
            collision = env.CheckCollision(obj,obj2)
            if collision:
                if (obj2 == robot and obj in robot.GetGrabbed())
                    or (obj == robot and obj2 in robot.GetGrabbed()):
                    collision = False
                    continue
                collision_flag = True
                break

    return collision_flag

# example: go to grasp pose of can_1.
p = generate_grasp_pose("can_1")
go_to_pose(p)

# example: grab can_1.
grab_object("can_1")

# example: pickup can_1.
p = generate_grasp_pose("can_1")
go_to_pose(p)
grab_object("can_1")

# example: putdown can_1 in box.
p = generate_putdown_pose("can_1","box")
go_to_pose(p)
un_grab("can_1")

# example: check collisions for can_1.
collision_flag = collision_check("can_1")
\end{promptbox}
\subsubsection*{Prompt For Packing 1 Can}
\begin{promptbox}
Using the functions given above solve the following problem: #
There is 1 can, can_1, on the table. The function returns true if it can
↪  accommodate
all the cans in a box at a pose such that each can is not in collision with every
other can in the box. The function needs to accommodate all objects in the box.
# The signature for the function required is:
collision_flag = put_all_cans_in_box("box").
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

Prompt For Packing 2 Can

```
Using the functions given above solve the following problem: #
There are 2 cans, can_1, and can_2, on the table. The function returns true if it
↪  can
accommodate all the cans in a box at a pose such that each can is not in
↪  collision
with every other can in the box. The function needs to accommodate all objects in
the box.
# The signature for the function required is:
collision_flag = put_all_cans_in_box("box").
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

Prompt For Packing 3 Can

```
Using the functions given above solve the following problem: #
There are 3 cans, can_1, can_2, and can_3 on the table. The function returns true
if it can accommodate all the cans in a box at a pose such that each can is not
in collision with every other can in the box. The function needs to accommodate
↪  all
objects in the box.
# The signature for the function required is:
collision_flag = put_all_cans_in_box("box").
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```

Prompt For Packing 4 Can

```
Using the functions given above solve the following problem: #
There are 3 cans, can_1, can_2, and can_3 on the table. The function returns true
if it can accommodate all the cans in a box at a pose such that each can is not
in collision with every other can in the box. The function needs to accommodate
↪  all
objects in the box.
# The signature for the function required is:
collision_flag = put_all_cans_in_box("box").
# only give the code and no other text including comments, markdowns or for which
language the code is given. Also, at the end call the function as well.
```