To appear in the 2015 Proc. IEEE Intl. Conf. Robotics and Automation (ICRA)

Towards Manipulation Planning with Temporal Logic Specifications

Keliang He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi

Abstract— **Manipulation planning from high-level task specifications, even though highly desirable, is a challenging problem. The large dimensionality of manipulators and complexity of task specifications make the problem computationally intractable. This work introduces a manipulation planning framework with *linear temporal logic* (LTL) specifications. The use of LTL as the specification language allows the expression of rich and complex manipulation tasks. The framework deals with the state-explosion problem through a novel abstraction technique. Given a robotic system, a workspace consisting of obstacles, manipulable objects, and locations of interest, and a co-safe LTL specification over the objects and locations, the framework computes a motion plan to achieve the task through a synergistic multi-layered planning architecture. The power of the framework is demonstrated through case studies, in which the planner efficiently computes plans for complex tasks. The case studies also illustrate the ability of the framework in intelligently moving away objects that block desired executions without requiring backtracking.**

I. INTRODUCTION

In recent years, there has been an increasing interest in the *integration of task and motion planning* (ITMP) (e.g., [1]–[7]). The ultimate goal of ITMP is to enable the specification of robotic tasks at a high level and the automatic generation of satisfying motion plans. This is, however, a challenging problem due to the complexity of combining the robot's continuous motion planning with discrete task reasoning. For instance, consider a robot behind a counter at a sushi bar serving several customers. On the counter, there exist several glasses, sushi plates, a tip jar, and a tray for dirty dishes. An example of a desirable task specification for this robot is

"Serve the customers such that each one first receives a glass of water and then a sushi plate. Eventually, show the tip jar to the customers, but it must not be placed in front of a customer unless she is served both drinks and sushi. Finally, all the dirty dishes must be collected and then placed on the tray."

The specification expresses the task at a high level and does not detail a sequence of movements to be performed. This paper focuses on a manipulation planning framework that enables planning from such high-level specifications expressed in temporal logics, namely *linear temporal logic* (LTL) [8].

The power of LTL as the specification language in ITMP has been demonstrated in applications other than manipulation, particularly mobile robotics (e.g., [1]–[3], [9]–[11]). LTL allows Boolean and temporal constraints, accommodating rich specifications such as the example above. In addition to high expressivity, LTL planners provide strong correctness and completeness guarantees [1]–[3]. These methods, however, face a combinatorial blow up of the state space, commonly known as the *state-explosion problem* [9]. Therefore, LTL planning frameworks usually involve a construction of discrete abstraction of the robotic system [1]–[3]. Generally, the construction of such abstractions is nontrivial. For simple mobile systems, a *(bi)similar* abstraction is typically possible, guaranteeing completeness of the LTL planners (e.g., [1], [2]). For complex mobile systems, sampling-based motion planners are usually employed to provide probabilistic completeness guarantees [3], [10], [11]. The abstraction techniques developed for mobile robotics rely on discretizations of the state-spaces, making them intractable for manipulation planning due to the large dimensionality in the manipulation problem.

ITMP has received a lot of attention in robot manipulation planning (e.g., [4]–[7], [12]–[14]). These works typically employ classical AI planners [15], such as FF [16], at the task level for discrete planning and use a motion planner at the low level to find paths in the robot's continuous space. Even though these ITMP frameworks enable manipulation planning for high-level tasks, their specifications are restricted to reaching a set in the configuration space. The temporal requirements over the events along the path are typically encoded as a part of the action requirements.

Popular architectures in ITMP include hierarchical two-layer planners and cooperative three-layer planners (e.g., [6], [7], [13]). Particularly, the work in [13] introduces an aggressive hierarchical approach to ITMP. The framework makes choices at the high level and commits to them in a top-down fashion. This architecture is relaxed in [6] by first constructing partial plans (skeletons) and then solving a constraint-satisfaction problem. The recent work in [7] proposes a three-layer planning framework to take advantage of the existing off-the-shelf task and motion planners by introducing an interface layer. This interface layer allows communication between task and motion planners through identifying failures of the motion planner and asking the task planner for a different high-level plan. The process repeats until either a motion plan is successfully computed or all of the high-level plans are exhausted. In the latter case, the interface layer resorts to moving a removable object to another location and reinitializing the planning process. This removal action, however, may cause a blocking of future motions. This problem is commonly referred to as *backtracking*.

This paper improves the state-of-the-art by allowing more expressivity, specifically enabling temporal reasoning at the task level and significantly relaxing action requirements. The main contribution of this work is an LTL planning framework

This work is supported in part by NSF 1317849, 1139011, and 1018798.

The authors are with the Department of Computer Science at Rice University, Houston, TX, USA, Email: {keliang.he,morteza,kavraki,vardi}@rice.edu.

for robotic manipulation. Different from [17] that uses LTL to specify controller properties for dexterous manipulation, where the focus is on securing the grasp of an object with provably correct finger gaits, the focus of this work is to generate the necessary grasp and transfer of objects between locations of interest in order to accomplish a task specified in LTL. To the best of our knowledge, this is the first attempt in employing temporal logics in this context of manipulation planning. The use of LTL allows the expression of complex manipulation tasks with temporal constraints, which existing ITMP solutions for manipulation are not able to accommodate. Since LTL is a formal language, by expressing the task in LTL, we remove any ambiguity from the specifications and formalize the planning procedure. Even though the problem of state explosion [18] of the LTL planners is inherited in this framework, this problem is partially alleviated by a novel abstraction technique and a synergistic multi-layered planning structure. This structure follows the synergistic framework introduced in [3] and is particularly modified for manipulation planning. The abstraction is developed in this paper and is essentially a coarse representation of all the ways that the robot can manipulate the objects. Consequently, the planner gains a global awareness with this abstraction. By combining the abstraction with the automaton that represents the LTL specification [19], the planner finds all possible methods of satisfying the specification at a high level. It then uses the planning layers to generate a satisfying continuous path. Subsequently, the common occurring problem of backtracking in ITMP as encountered in [7] does not arise in this framework. The power of our method is illustrated through case studies, in which motion plans for complex manipulation tasks were efficiently computed for a PR2 in simulation. The case studies also demonstrate the capabilities of the framework in intelligently removing objects blocking desired paths without requiring backtracking in the remaining steps of planning.

II. Problem Formulation

Let R be a robot in a three-dimensional workspace. The workspace consists of a finite set of obstacles, a finite set of manipulable objects denoted by $O = \{o_1, \ldots, o_n\}$, and a set of locations of interest for the objects to be placed. We assume that the locations are mutually exclusive, and each one is large enough to be occupied by an object and only one object. Each location is given a set of labels from the set $Ł = \{l_1, \ldots, l_m\}$. Note that the same label l_i could be assigned to multiple locations, allowing a large location that may contain several objects to be represented as several locations with the same label.

To illustrate these concepts, consider again the robot server scenario in Section I. The manipulable objects in this example are the glasses, the sushi plates, and the tip jar. The locations of interest are the counter-top areas in front of the customers, the tray, and the initial locations of the glasses and plates.

A. Manipulation Planning

To define the manipulation problem, the configurations of the objects being manipulated are represented in addition to the configuration of the robot. The configuration space (*C-space* denoted by $\mathcal{C}$) of the system is the Cartesian product of the C-spaces of the robot ($\mathcal{C}_R$) and the joint C-space of the objects ($\mathcal{C}_O = \mathcal{C}_{o_1} \times \ldots \times \mathcal{C}_{o_n}$), i.e., $\mathcal{C} = \mathcal{C}_R \times \mathcal{C}_O$. Under this formulation, the manipulation planning problem is to find a path $P : [0,1] \to \mathcal{C}$ such that $P(0)$ is the initial configuration of the system, and P satisfies some constraints. One constraint is that the objects can only move while being manipulated by the robot. Another constraint imposes that the path is collision free. The final constraint is that the path must satisfy a specified task. In this work, the task is expressed using co-safe LTL, which is defined below.

B. Task Specification Language - Co-safe LTL

Co-safe LTL is a fragment of LTL and combines Boolean operators with temporal reasoning. Let Π be a set of Boolean atomic propositions (detailed below for manipulation problems). The syntax of co-safe LTL formula φ over Π is inductively defined as:

$$\varphi = \pi \mid \neg\pi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \mathcal{U} \varphi \mid \mathcal{X}\varphi \mid \mathcal{F}\varphi$$

where $\pi \in \Pi$. The Boolean operators are "negation" ($\neg$), "and" ($\wedge$, true if both operands are true), and "or" ($\vee$, true if either operands are true). The temporal operators are "until" ($\varphi_1 \mathcal{U} \varphi_2$, true if φ_1 holds true until φ_2 is true), "next" ($\mathcal{X}\varphi$, true if φ is true on the next time step), and "eventually" ($\mathcal{F}\varphi$, true if φ is true at some point in the future). See [19] for formal definitions of the syntax and semantics of co-safe LTL.

The semantics of LTL formulas are defined over infinite words, which are infinite sequences of letters from the alphabet 2^Π. Nevertheless, each word that is accepted by a co-safe LTL formula can be detected by one of its finite prefixes [19]. Thus, co-safe LTL is a desirable specification language to express robotic tasks that must be accomplished in finite time.

In manipulation tasks, the interest is in the manipulation of the objects between the labeled locations. We define the propositions of the co-safe LTL formulas to be of the form $o_i \in l_j$, which reads as, "object o_i is in a location with the label l_j." Therefore, the set of all such propositions is Π. For each configuration of the objects, we define a labeling function $\mathfrak{L} : \mathcal{C}_O \to 2^\Pi$ that maps the configuration to the set of all propositions that hold true in the configuration. Such a set of valid propositions at a configuration is called a letter from the alphabet 2^Π. We define the word generated by a continuous path in $\mathcal{C}_O$ as the sequence of letters assigned by $\mathfrak{L}$ along this path, where a letter is appended to the word only if it differs from the previous letter. Recall that P is a mapping to $\mathcal{C} = \mathcal{C}_R \times \mathcal{C}_O$. Thus, P generates a word through its projection onto $\mathcal{C}_O$ that represents the sequence of truth assignments to the propositions along the execution.

C. Problem Definition

Given a robot with configuration space $\mathcal{C}_R$, a set of manipulable objects O with joint configuration space $\mathcal{C}_O$, a set of propositions Π, a labeling function $\mathfrak{L} : \mathcal{C}_O \to 2^\Pi$, and a co-safe LTL formula φ over Π, find a valid path $P : [0,1] \to \mathcal{C}$ such that the word generated by P using $\mathfrak{L}$ satisfies φ.

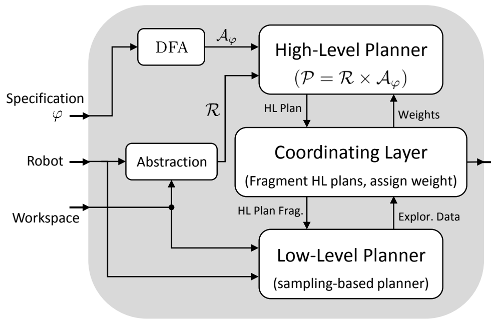


Fig. 1: LTL manipulation planning framework.

III. Planning Framework

To approach the above problem, we draw inspiration from the synergistic framework in [3] and propose a planning framework that consists of three layers: high-level planner, coordinating layer, and low-level search layer. The discretization method proposed in [3], however, is not an appropriate abstraction for manipulation due to the state-explosion problem caused by the high dimensionality of the C-space. Instead, we introduce a suitable abstraction that is essentially a composition of the permutations of the objects in the locations and the fundamental robot motions required to manipulate the objects. This abstraction captures all possible robotic manipulations of the objects. The high-level planner operates on a graph, which is the product of this abstraction with the automaton that represents the specification. This *product graph* captures all possible ways that the robot can move the objects between the locations to satisfy the specification. The high-level planner searches for a path over this graph and passes it to the coordinating layer, which decomposes the path into segments. The segments that need motion planning are passed to the low-level planner. These segments are generally the motions that the robot needs to perform between locations. The low-level layer searches $\mathcal{C}$ by extending a sampling-based tree to find a path that realizes the corresponding high-level plan segment. During this search, the coordinating layer collects the exploration information and assigns weights to the abstraction graph edges of the corresponding plan segments, representing their realizability difficulty. As a result, the high-level planner learns the feasibility of the high-level plans and suggests the ones for which motion planning is more likely to succeed. Hence, the synergy between these planning layers results in the generation of a satisfying continuous path. Figure 1 shows a block diagram representation of this planning framework, whose components are detailed in the following sections.

A. Abstraction

Recall that the major challenge in LTL planning for continuous systems is state explosion. In manipulation, this issue becomes even more significant than in mobile robotics due to the large number of degrees of freedom of typical manipulators. Therefore, the abstraction techniques used for mobile robots, which consider only the workspace of the robot, become ineffective as such abstractions cannot be computed in

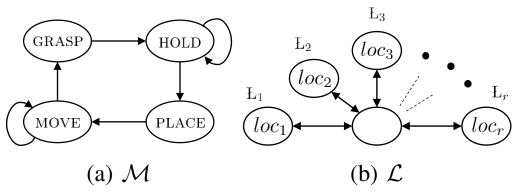


Fig. 2: (a) Motion primitive graph $\mathcal{M}$. Motion of the robot must follow a path on $\mathcal{M}$. (b) A possible location graph $\mathcal{L}$. The locations of interest are labeled with $Ł_i \subseteq Ł$ for $i \in \{1, \dots, r\}$ and are connected by the intermediate area.

high dimensional configuration spaces [18]. Here, we present the construction of a suitable abstraction.

To obtain this abstraction, we first discretize the actions of the robot to the set of motion primitives GRASP, PLACE, HOLD, and MOVE and produce the graph $\mathcal{M}$ to encode the allowed sequence of these motions as shown in Figure 2a. These primitives capture the ways in which the robot can interact with the world. Given the robot is within a set of configurations called the pre-image of a location of interest, and certain criteria regarding the state of the system are satisfied, the primitives GRASP (PLACE) are guaranteed to perform the necessary actions to grasp (place) the object from (in) the location, and end in a configuration called the post-image of the location. The criteria to perform each of the actions are detailed later in this section. These actions are precomputed, similar to [7], for pre-images close to the locations of interest. The primitive HOLD represents the motions from one location to another while holding an object. Similarly, MOVE represents the robot motions between locations while no objects is in the gripper. The HOLD and MOVE actions are created by the low-level planner during planning. In this work, we consider only these four actions, but the framework could allow the inclusion of other ones, such as pushing and pulling.

Next, we construct the location graph $\mathcal{L}$, which captures the locations of interest in the workspace of the robot. The location graph consists of one node for each location of interest, and an intermediate area representing the free-space that connects the locations of interest. The edges of $\mathcal{L}$ represent the adjacency of the locations. For the simplest case, $\mathcal{L}$ consists of the locations of interest and an intermediate area that connects all these locations (when they are non-adjacent). In our setting, an object belongs to the intermediate area when it is in the manipulator's gripper. Recall from Section II that each location is associated with a set of labels. Figure 2b illustrates the location graph of the simplest case.

We then combine these graphs through an automatic procedure to obtain a coarse abstraction, which is essentially a discrete representation of all possible ways that the robot can manipulate the objects between the locations of interests. We define the abstraction graph $\mathcal{R} = (V, v_0, E, L, W)$, where

- $V = \mathcal{M} \times \mathcal{L} \times (O \cup \{\emptyset\}) \times \prod_{o_i \in O} \mathcal{L}$ is the set of abstraction nodes. A node in V can be represented as the tuple $(\mathsf{action}, \mathsf{eeLoc}, \mathsf{grpObj}, \mathsf{obj}_1\mathsf{Loc}, \cdots, \mathsf{obj}_n\mathsf{Loc})$. The first component $\mathsf{action} \in \mathcal{M}$ is the motion primitive being performed. Second component $\mathsf{eeLoc} \in \mathcal{L}$ is the location of the end effector. Third component $\mathsf{grpObj} \in (O \cup \{\emptyset\})$

is the object being manipulated, which could be the empty set. The final components $\mathsf{obj}_i\mathsf{Loc} \in \mathcal{L}$ for $i \in \{1, \dots, |O|\}$ are the objects' locations.

- $v_0 \in V$ is the initial node. Note that every continuous state of $\mathcal{C}$ can be mapped onto a node in V by examining the configuration of the robot, the objects' locations, and the action being performed. The initial configuration $c_0 \in \mathcal{C}$ is mapped to v_0.
- $E \subseteq V \times V$ is the set of edges of the abstraction graph. This captures all possible moves from an abstraction node in V to another. The rules of the existence of an edge between two nodes are detailed below.
- $L : V \to 2^{\Pi}$ is the labeling function from the abstraction nodes to the set of valid propositions at the node induced by the labeling function $\mathfrak{L}$. The definition of this function is also detailed below.
- $W : E \to \mathbb{R}$ is the weight function which assigns a weight to each edge of the graph. Intuitively, the weight of the edge between two nodes represents the difficulty level of generating a motion plan between the corresponding configurations.

Note that not all nodes in V are physically possible. For example, the case where o_1 is in l_1 and the gripper object is also o_1 is not possible. The number of valid nodes in the abstraction is $2(|\mathcal{L}| + 1)\mathrm{P}_{|O|}^{(|\mathcal{L}|+1)}$, where P_n^k is the k-permutation from n elements. We implicitly construct the abstraction by defining the possible edges of $\mathcal{R}$. Therefore, we can construct the reachable set of valid nodes by starting from v_0 and following transitions that only point to valid nodes. A valid transition exists between nodes $v, v' \in V$, i.e., $(v, v') \in E$ if one of the following conditions is satisfied. Each condition corresponds to one of the transitions on the motion primitive graph $\mathcal{M}$. These conditions are similar to pre- and post-conditions in AI planning. In these conditions, all the unmentioned components of v' are required to be the same as v. The conditions are:

- $v.\mathsf{action} = \text{GRASP}$, $v'.\mathsf{action} = \text{HOLD}$, $v.\mathsf{grpObj} = \emptyset$, for some i, $v.\mathsf{eeLoc} = v.\mathsf{obj}_i\mathsf{Loc}$ and $v'.\mathsf{grpObj} = o_i$.
- $v.\mathsf{action} = \text{HOLD}$, $v'.\mathsf{action} = \text{HOLD}$, and $v.\mathsf{eeLoc}$ is connected to $v'.\mathsf{eeLoc}$ by an edge in $\mathcal{L}$.
- $v.\mathsf{action} = \text{HOLD}$, $v'.\mathsf{action} = \text{PLACE}$, and $v.\mathsf{eeLoc}$ is not the intermediate area.
- $v.\mathsf{action} = \text{PLACE}$, $v'.\mathsf{action} = \text{MOVE}$, for all j, $v.\mathsf{obj}_j\mathsf{Loc} \neq v.\mathsf{eeLoc}$, and for some i, $v.\mathsf{grpObj} = o_i$ and $v'.\mathsf{obj}_i\mathsf{Loc} = v.\mathsf{eeLoc}$.
- $v.\mathsf{action} = \text{MOVE}$, $v'.\mathsf{action} = \text{MOVE}$, and $v.\mathsf{eeLoc}$ is connected to $v'.\mathsf{eeLoc}$ by an edge in $\mathcal{L}$.
- $v.\mathsf{action} = \text{MOVE}$, $v'.\mathsf{action} = \text{GRASP}$, and $v.\mathsf{eeLoc}$ is not the intermediate area.

We can extract the set of propositions in the form $o_i \in l_j$ that are valid in v by examining its components $\mathsf{obj}_i\mathsf{Loc}$. Therefore, the labeling function of the abstraction nodes $L : V \to 2^{\Pi}$ is defined as $L(v) = \{(o_i \in l_j) \mid l_j \text{ is a label of location } v.\mathsf{obj}_i\mathsf{Loc}\}$. The edge weights are initially assigned a value of one, and they are updated during the planning process (see Section III-D).

B. Product Graph

In order to find a sequence of abstraction states to satisfy the co-safe LTL formula φ, we use a high-level planner to search a graph composed of the abstraction and the specification as in [3] (see Figure 1). From a co-safe LTL formula, a *deterministic finite automaton* (DFA) that accepts precisely all the satisfying words of φ can be constructed [19]. This DFA is defined as $\mathcal{A}_\varphi = (Z, z_0, \Sigma, \delta, F)$, where Z is the finite set of states, and z_0 is the initial state. Σ is the alphabet of the LTL formula (recall from Section II-B that $\Sigma = 2^{\Pi}$). The letters of Σ are the possible truth assignments of the atomic propositions. $\delta : Z \times \Sigma \to Z$ is the transition function. F is the set of final states (also called accepting states).

The accepting runs of $\mathcal{A}_\varphi$ are paths from z_0 to a state in F following the transitions in δ. The letters along the path represent the sequence of truth assignments of the propositions to satisfy the specification. However, such assignments may not respect the physical world. For example, the truth assignment on one transition may require an object to be in multiple locations at the same time ($o_i \in l_j \wedge o_i \in l_k$).

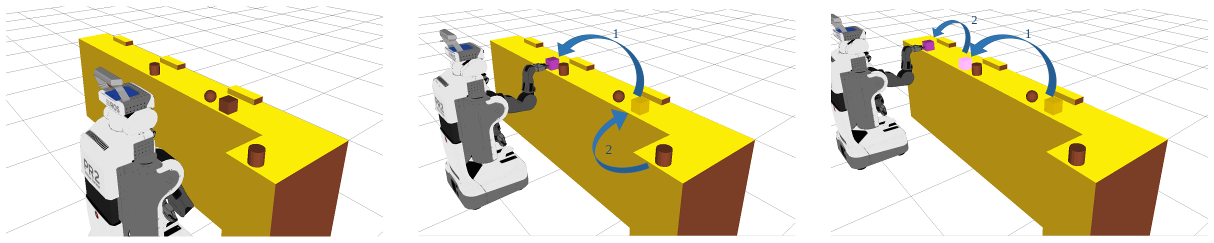
In order to find plans that satisfy both the specification and the manipulation environment, we construct the product graph $\mathcal{P}$ between the abstraction $\mathcal{R}$ and the DFA $\mathcal{A}_\varphi$, i.e., $\mathcal{P} = \mathcal{R} \times \mathcal{A}_\varphi$. A product graph node p has two components, $p = (v, z)$, where v is a node in $\mathcal{R}$, and z is a state in $\mathcal{A}_\varphi$. An edge from $p = (v, z)$ to $p' = (v', z')$ exists iff there is an edge $(v, v') \in E$ and $\delta(z, L(v')) = z'$. Furthermore, edges of $\mathcal{P}$ inherit the weights of the corresponding edges in $\mathcal{R}$.

The start node of $\mathcal{P}$ is $p_0 = (v_0, z_0)$, and the goal nodes of $\mathcal{P}$ are the nodes (v_i, z_j), where $z_j \in F$. An accepting path $p_0 p_1 \dots p_k$ from the start node to a goal node on $\mathcal{P}$ induces a path $v_0 v_1 \dots v_k$ on $\mathcal{R}$ and a run $z_0 z_1 \dots z_k$ on $\mathcal{A}_\varphi$. Note that $z_0 \dots z_k$ is necessarily an accepting run of $\mathcal{A}_\varphi$. Moreover, the path $v_0 \dots v_k$ in $\mathcal{R}$ respects the constraints of (1) each object being at exactly one location, and (2) objects can move only by the manipulator. Therefore, the motion plan that realizes the edges (v_i, v_{i+1}) for all $i \in \{0, \dots, k-1\}$ satisfies the specification φ.

In our implementation, only the reachable portion of $\mathcal{P}$ from p_0 is explicitly generated using $\mathcal{A}_\varphi$ and an implicit representation of the edges of $\mathcal{R}$. Dijkstra's algorithm is used to find an accepting path on $\mathcal{P}$ with minimum total edge weight. This path from the high-level planner is then used as a *guide* for the low-level continuous planner to implement each of the motion primitives required [3]. Recall that the size of the abstraction is $2(|\mathcal{L}|+1)\mathrm{P}_{|O|}^{(|\mathcal{L}|+1)}$; therefore the size of the product graph $\mathcal{P}$ grows significantly with the increase in the number of objects and locations, as well as the size of the specification. In such cases, the product graph can be represented implicitly, and a heuristic search algorithm can be used to search $\mathcal{P}$ for a satisfying path. The selection of a good heuristic remains for future work.

C. Realization of Guide

When a guide is generated by the high-level planner, it is passed to the coordinating layer (see Figure 1). The coordinating layer breaks the guide into segments according



(a) Initial Configuration (b) Sample execution of φ_1 (c) Sample execution of φ_2

Fig. 3: (a) Initial configuration. Cubical object: o_{snack}; spherical object: o_{tipjar}; cylinder farther from camera: o_{drink1}; cylinder closer to camera: o_{drink2}. Horizontal bars on the table mark the customer locations, from customer 1 (closest) to customer 3 (farthest). (b) An execution of Specification 1. The robot moves o_{snack} out of the way for o_{drink2} to be served to customer 1. (c) An execution of Specification 2. The robot moves o_{snack} to customer 2 and then to customer 3 to offer them snacks.

to the motion primitives. For a GRASP or PLACE action, the coordinating layer inserts the precomputed motion primitive.

For a segment of the guide that consists of a sequence of HOLD or MOVE actions, the coordinating layer translates the abstraction edges involved into a continuous space planning query from the post-image of the first node to the pre-image of the last node. The coordinating layer then allocates some time for the low-level planner to attempt to solve this continuous motion planning query. The low-level planner uses sampling based motion planning techniques to implement the path required (e.g., [20], [21]). The coordinating layer starts with the first action and continues until all segments are implemented, in which case a continuous path that satisfies the specification is generated. If a low-level planning query fails within the given time, the coordinating layer asks the high-level planner to generate a new guide.

D. Feedback from Low to High

The failure of the motion planning query in the given amount of time may occur due to several reasons. One reason is that the obstacles in the environment and the robot's physical constraints allow no possible solution. Another reason is that the high dimensionality (generally 6 or higher) of the manipulation planning problem and the possible existence of narrow passages may require more planning time than is given to the low-level planner. Specifically, if the low-level planner uses sampling based techniques, the planning time may vary for the same query. Therefore, when low-level planning fails in the given time, it is crucial for the framework to decide whether to continue with the same query or to generate a new high-level plan. The synergistic framework in [3] provides an answer to this problem. For each failure of a planning query, the coordinating layer increases the weight of the corresponding abstraction edge proportional to the planning time, and asks for a new high-level plan. Therefore, if a query fails multiple times, the query is interpreted as a difficult planing problem and becomes less likely to be included in the next high-level plan. On the other hand, if a segment of the high-level plan is successfully found, the weight of the segment is set to 0. The high-level planner considers this edge to be desirable in the future searches. This feedback is represented in Figure 1 as the edges from low-level planner to the coordinating layer and the coordinating layer to the high-level planner.

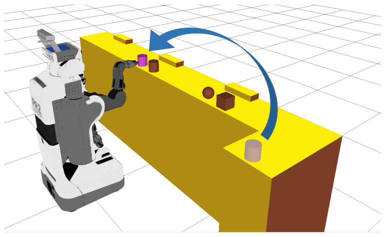
E. Completeness

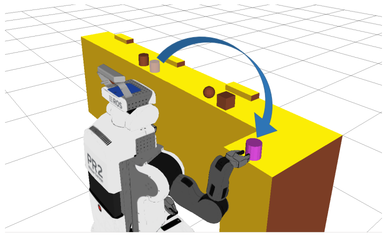
The proposed LTL planning framework is probabilistically complete for the manipulation problem defined in Section II-C, assuming all possible ways to manipulate the objects are captured by the motion primitives GRASP and PLACE. That is, if the grasping or placing of an object exists from a configuration, then it can be achieved through a MOVE or HOLD primitive and the precomputed GRASP or PLACE primitive. Under this assumption, if there exists a continuous path that satisfies the LTL specification, then the probability of failing to find the path approaches zero as planning time increases. This is because the planning layers work in a synergistic manner. By assigning weights to the abstraction edges according to the exploration data, the high-level planner eventually considers all satisfying paths on $\mathcal{P}$ infinitely often. The path segments with HOLD and MOVE actions are assigned to the low-level planner, which is probabilistically complete, infinitely often. The motion plans for the path segments corresponding to GRASP and PLACE are simply inserted, as they are assumed to exist.

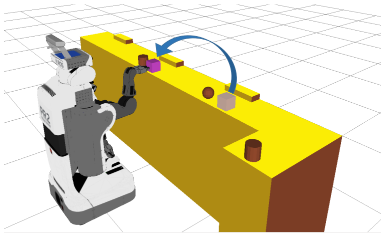
IV. Experiments

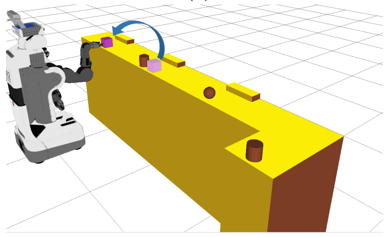
Our planning framework is implemented in C++. Translation from co-safe LTL to DFA is done using Spot [22]. The high-level planner is implemented in the Open Motion Planning Library (OMPL) [23]. The coordinating layer is implemented as a ROS package. Low-level planning is done through MoveIt! using the LBKPIECE [20] and RRTConnect [21] planners. The framework is used to plan for a PR2 in simulation, visualized using RViz. All of the experiments are perform on an AMD FX-4100 QUAD-CORE processor with 16GB RAM and Ubuntu 12.04 x64.

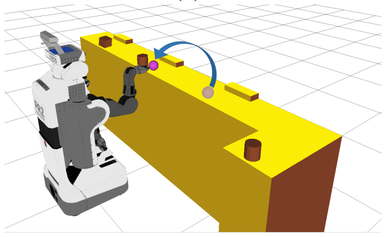
To test our planning framework, we created a scenario where a PR2, which is a mobile manipulation platform, was asked to perform a set of waiting tasks. As shown in Figure 3a, the scene consisted of an L-shape counter with two drink glasses o_{drink1} and o_{drink2}, a snack box o_{snack}, a tip jar o_{tipjar}, and three customer seats. Each customer was given a unique ID, and two locations in front of each customer were made available for objects to be placed. The locations were labeled with their corresponding customer ID. Initially, the snack box and the tip jar are in front of customer 1, drink 1 is in front of customer 2, and drink 2 is in a preperation area. We asked the robot to perform the following tasks:


(a)


(b)


(c)


(d)


(e)

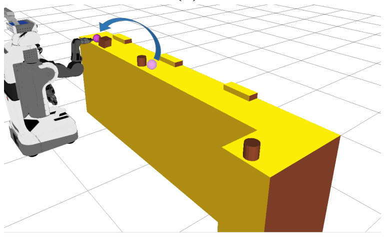

(f)

Fig. 4: One execution of specification 3. An object in purple (brighter) indicates the object has just been moved. The initial configuration is illustrated in Figure 3a. First, the robot moves o_{drink2} to customer 2 (a) and then moves o_{drink1} back to the preparation area (b). Next, the robot offers o_{snack} to each customer (c,d), and shows o_{tipjar} to each customer (e,f). Note that this execution is only a sample of many motion plans that can have different sequences of moves, depending on the high-level plans found and the low-level queries solved.

Specification 1: Serve customer 1 her drink.

Specification 2: Offer snacks to every customer.

Specification 3: First, take drink 2 to customer 2 and bring the empty drink 1 back to the preparation area. Next, offer snacks to each of the customers and show the tip jar to the ones whom have already been served snacks.

These specifications can be expressed in co-safe LTL using the following formulas, for φ_1, φ_2, and φ_3 respectively,

$$\begin{aligned} \varphi_1 &= \mathcal{F}(o_{\text{drink1}} \in l_1), \qquad \varphi_2 = \textstyle\bigwedge_{i=1}^{3} \mathcal{F}(o_{\text{snack}} \in l_i), \\ \varphi_3 &= \mathcal{F}\big(o_{\text{drink2}} \in l_2 \wedge o_{\text{drink1}} \in l_4 \wedge \\ &\qquad \textstyle\bigwedge_{i=1}^{3} \mathcal{F}\big(o_{\text{snack}} \in l_i \wedge \mathcal{F}(o_{\text{tipjar}} \in l_i)\big)\big). \end{aligned}$$

Note that for these specifications, the robot has the freedom to choose the order in which the sub-tasks are performed. In this case study, the order of sub-tasks to satisfy the specification is not unique, and the planner must find such an order for each of the three formulas. In Specification 1, since both locations for customer 1 are occupied initially, even though the task is to serve the drink to customer 1, the planner must make a decision to remove one of the occupying objects first. Specification 2 does not indicate a particular order of customers to serve snacks. The decision is left to the system to pick an order that works. In Specification 3, the robot could give snacks to everyone first, then show the tip jar in one execution. Alternatively, the robot could serve each customer one by one, or a combination of the two ways. The planner has to intelligently pick which order to execute the sub-tasks to avoid motions that violate the specification. Note that some of the existing manipulation ITMP frameworks such as the one introduced in [7] can support Specification 1, where the task can be reduced to reaching a set in $\mathcal{C}$. However, to the best of our knowledge, no existing framework can plan for Specifications 2 and 3. We used the framework proposed in this paper to plan for all the specifications.

We performed 50 motion planning runs for each specification. In all the cases, the framework successfully produced a satisfying plan within 100 seconds. The size of DFAs and the product graphs, along with the average computation times are shown in Table I. The times shown here are the total plan time of the high-level and low-level planners, accounting for all the calls during the creation of one motion plan. The coordinating layer time is not measured as it is simply constructing low-level planning queries and setting weights on the abstraction graph. For Specification 1, which can be solved using other manipulation planning frameworks (e.g., [7]), the planning time of our method is comparable to the other methods.

Note that the sizes of $\mathcal{A}_\varphi$ and $\mathcal{P}$ increase with the length of the formula. For φ_3, the number of states in $\mathcal{A}_\varphi$ and $\mathcal{P}$ are an order of magnitude larger than the ones for φ_1. The increase in the size of $\mathcal{P}$ causes an increase in the high-level planning time because when the high-level planner is called, we run Dijkstra's algorithm on a larger graph.

An execution of φ_1 and an execution of φ_2 are illustrated in Figures 3b and 3c. Note that the actions performed in these figures may be different from other executions since the robot can satisfy the specifications in multiple ways. Snapshots of the execution of one of the plans for Specification 3 are shown in Figure 4. In this plan, o_{drink1} was moved out of the way immediately after o_{drink2} was given to customer 2. The robot then offered snacks to everyone and then the tip jar. This is a shorter motion plan than serving the customers

Spec.	$\lvert Z_{\mathcal{A}_\varphi}\rvert$	$\lvert E_{\mathcal{A}_\varphi}\rvert$	$\lvert V_{\mathcal{P}}\rvert$	T_{HL}(s)	T_{LL}(s)
φ_1	2	3	44,100	2.76	12.32
φ_2	8	27	75,511	4.48	8.07
φ_3	27	290	498,000	33.12	31.15

TABLE I: Planning data for Specifications 1, 2, and 3. $\lvert Z_{\mathcal{A}_\varphi}\rvert$ and $\lvert E_{\mathcal{A}_\varphi}\rvert$ are the number of states and edges in the DFA, respectively. $\lvert V_{\mathcal{P}}\rvert$ is the number of nodes constructed in the reachable portion of the product graph. T_{HL} and T_{LL} are the average planning times for the high-level and low-level planners, respectively, over 50 runs.

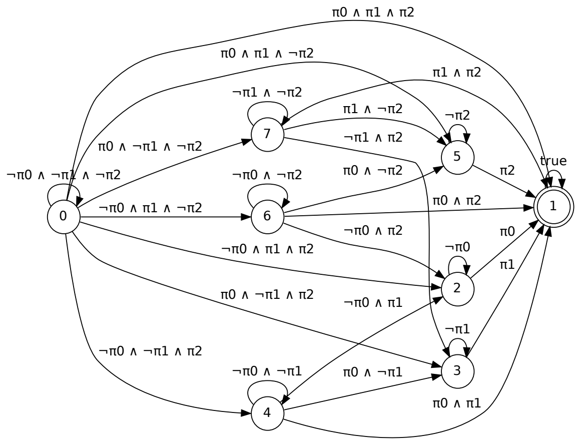


Fig. 5: $\mathcal{A}_{\varphi_2}$ generated using Spot [22]. Propositions π_i for $i \in \{0,1,2\}$ represent $o_{\text{snack}} \in l_{i+1}$. The shortest path from initial state 0 to accepting state 1 is through the edge with proposition assignment $\pi_0 \wedge \pi_1 \wedge \pi_2$ (top-most edge). This path, however, requires the snack box being in 3 different locations at the same time, which is physically impossible.

one by one, as drink 2 would need to be removed for both the snack box and tip jar to be offered to customer 2. This illustrates that by considering the location of each object for the entire duration of the task, our approach avoids potential backtracking where objects are unnecessarily removed to a location blocking future trajectories.

One may propose to plan for these specifications by using the DFA $\mathcal{A}_{\varphi_i}$ as a monitor for the sampling-based continuous planner. By keeping the progress of the nodes of the search tree over $\mathcal{A}_{\varphi_i}$, a solution can be reported as soon as a node reaches an accepting state of $\mathcal{A}_{\varphi_i}$. Our initial experiments show that such an approach would be computationally infeasible due to the large dimensionality of this manipulation problem. In such a method, the low-level planner has no guidance (bias) in expanding the search tree in the joint configuration space of the robot and the objects, which is 34 dimensional in our case study. Another naive method is to generate an accepting run on $\mathcal{A}_{\varphi_i}$ and attempt to implement this run using low-level planners. This method is also likely to fail because (1) $\mathcal{A}_{\varphi_i}$ has no notion of objects and locations, and thus the run may include impossible propositional assignments, as shown in Figure 5 and discussed in Section III-B; (2) the chosen run may have possible propositional assignments but not implementable in the continuous space due to robot's physical constraints and obstacles in the environment. Therefore, our synergistic multi-layer planner enables the computation of satisfying plans more efficiently than such methods.

V. Discussion

In this work, we introduce a manipulation planning framework for co-safe LTL specifications. The architecture of the framework includes a novel abstraction technique and a synergistic planning framework. With this abstraction, we are able to generate motion plans that do not exhibit backtracking behavior. The planner, however, suffers from poor runtime when the numbers of objects and locations are large, and the LTL specification is complex. To address this problem, possible extensions could include an implicit representation of the product graph and the use of heuristic search. These are to be explored in future works.

References

[1] H. Kress-Gazit, G. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *Int. Conf. on Robotics and Automation*. Rome, Italy: IEEE, 2007, pp. 3116–3121.

[2] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[3] A. Bhatia, M. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 55–64, Sep. 2011.

[4] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel, "Integrating symbolic and geometric planning for mobile manipulation," in *Safety, Security & Rescue Robotics, Int. Workshop on*. IEEE, 2009, pp. 1–6.

[5] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *Int. Conf. On Robotics and Automation*. IEEE, 2011, pp. 4575–4581.

[6] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2014.

[7] 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 *Int. Conf. on Rob. and Automation*. IEEE, 2014.

[8] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 1999.

[9] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Int. Conf. on Hybrid Systems: Computation and Control*, 2010, pp. 101–110.

[10] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *Int. Conf. on Hybrid Systems: Computation and Control*. ACM, Apr. 2013, pp. 353–362.

[11] C. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, Nov 2013, pp. 4817–4822.

[12] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[13] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Int. Conf. on Robotics and Automation*. IEEE, 2011, pp. 1470–1477.

[14] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *Int. Conf. on Robotics and Automation*, 2010, pp. 5002–5008.

[15] M. Ghallab, D. Nau, and P. Traverso, *Automated planning: theory & practice*. Elsevier, 2004.

[16] J. Hoffmann, "FF: The fast-forward planning system," *AI magazine*, vol. 22, no. 3, p. 57, 2001.

[17] S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick, and R. M. Murray, "Towards formal synthesis of reactive controllers for dexterous robotic manipulation." in *Int. Conf. on Robotics and Automation*, 2012.

[18] M. Rungger, M. Mazo Jr, and P. Tabuada, "Scaling up controller synthesis for linear systems and safety specifications." in *Conf. on Decision and Control*. IEEE, 2012, pp. 7638–7643.

[19] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, pp. 291–314, 2001.

[20] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[21] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Int. Conf. on Robotics and Automation*, vol. 2. IEEE, 2000, pp. 995–1001.

[22] A. Duret-Lutz and D. Poitrenaud, "Spot: An extensible model checking library using transition-based generalized buchi automata," in *Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. IEEE, 2004, pp. 76–83.

[23] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.