Automated Abstraction of Manipulation Domains for Cost-Based Reactive Synthesis

Keliang He[1], Morteza Lahijanian[2], Lydia E. Kavraki[1], and Moshe Y. Vardi[1]

***Abstract*—When robotic manipulators perform high-level tasks in the presence of another agent, e.g., a human, they must have a strategy that considers possible interferences in order to guarantee task completion and efficient resource usage. One approach to generate such strategies is called *reactive synthesis*. Reactive synthesis requires an *abstraction*, which is a discrete structure that captures the domain in which the robot and other agents operate. Existing works discuss the construction of abstractions for mobile robots through space decomposition; however, they cannot be applied to manipulation domains due to the curse of dimensionality caused by the manipulator and the objects. In this work, we present the *first* algorithm for automatic abstraction construction for reactive synthesis of manipulation tasks. We focus on tasks that involve picking and placing objects with possible extensions to other types of actions. The abstraction also provides an upper bound on path-based costs for robot actions. We combine this abstraction algorithm with our reactive synthesis planner to construct correct-by-construction plans. We demonstrate the power of the framework on a UR5 robot, completing complex tasks in face of interferences by a human.**

***Index Terms*—Formal Methods in Robotics and Automation, Manipulation Planning, Motion and Path Planning**

I. Introduction

ROBOTIC manipulators no longer need to be confined in cages. Recent developments have enabled robots to safely operate in shared workspaces alongside humans, expanding their applications to assistive and service robots, e.g., in assembly lines that are shared with humans, in restaurants, and at homes. In such scenarios, manipulators need to interact with a dynamic world and perform complex tasks in face of possible interferences by humans. This poses a challenge from the planning perspective because traditional algorithms that produce plans with fixed sequences of motions can no longer be employed. Instead, the robot must have a *strategy* that is *reactive*, i.e., performing motions to respond to the changes caused by humans or other agents.

Consider a scenario where a manufacturing robot that works alongside a human needs to perform a complex assembly task that may be expressed in a high-level language. A trained human may want to participate and help the robot achieve the task faster, or an untrained human may unknowingly interfere with the task by displacing assembly parts. Fig. 1 shows an in-lab example of this scenario, where an arch needs to be constructed with the white blocks as the base columns and the black block at the top. During task execution, a human may interfere and mistakenly place the black block as the base column or displace a column. Assuming a finite number of such interferences, the robot must complete the task within a given amount of energy. In this paper, we focus on such a reactive planning problem for manipulators and seek strategies that guarantee task completion.

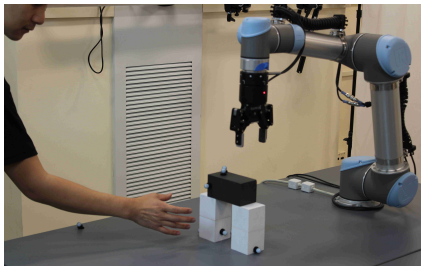


Fig. 1: The arch problem.

Reactive synthesis, a class of algorithms for finding reactive strategies, has been well studied in the field of program verification and synthesis [1]. Several works have applied reactive synthesis for strategy generation in mobile robotics [2]–[4]. Those works focus on infinite-horizon tasks expressed as *linear temporal logic* (LTL) [5] formulas, and assume finite, discrete structures that represent the continuous motions of the robot in the environment called *abstractions*. Given an LTL formula and an abstraction, the reactive planning algorithms construct a discrete game between the robot and the human and then compute a *winning strategy*, which guarantees to win the game by choosing an appropriate action for the robot in response to every action of the human. To enable reactive planning for manipulators, a similar game-based technique has been developed for robots with finite tasks and cost constraints [6]. That work assumes the existence of a discrete abstraction of the robotic manipulator. In general, the correct mapping of the discrete winning strategies to the continuous domain depends on the guarantees of the abstraction both for mobile robots and for manipulators. However, the construction of such an abstraction for continuous high-dimensional systems, such as manipulators, in a changing environment is non-trivial.

In this work, we introduce a novel algorithm for manipulation abstraction construction for reactive planning. The focus is on pick-and-place types of tasks with possible extensions to other actions such as pushing and pulling. The proposed abstraction preserves a simulation relation between the continuous and

Manuscript received: August, 14, 2018; Revised November, 11, 2018; Accepted December, 4, 2018.

This paper was recommended for publication by Editor Dezhen Song upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by NSF IIS, 1317849 NSF 1830549 and Rice University Funds.

[1]K. He, L. Kavraki, M. Vardi are with the Dept. of Computer Science, Rice University, Houston, TX, USA. keliang.h@gmail.com, kavraki@rice.edu, vardi@cs.rice.edu.

[2]M. Lahijanian is with the Dept. of Smead Aerospace Engineering Sciences, University of Colorado Boulder, CO, USA morteza.lahijanian@colorado.edu.

Digital Object Identifier (DOI): see top of this page.

discrete planning domains, hence guaranteeing the correctness of mapping the synthesized strategies to the continuous domain. In this abstraction, a discretization of the continuous domain is achieved by grouping states according to the locations of the objects. Transitions between states are captured by two sets of actions: robot actions and human actions. The feasibility of the robot actions is checked to ensure validity. In addition, to reason about resource consumption over strategies by the discrete reactive planner, approximations for the resource cost of robot actions are included in the abstraction. Two algorithms are proposed for the offline computation of feasibility and cost of robot actions. One enables fast computation, while the other provides better approximations. With this abstraction, we can map strategies for the discrete reactive synthesis problem to the continuous domain with correctness guarantees.

The main contribution of this work is an automated technique for abstraction of the manipulation domain for reactive synthesis problems. This is the first fully-automated abstraction technique for manipulation planning to the best of our knowledge. This work allows us to study reactive synthesis in the context of manipulation and understand its advantages and disadvantages. It also supports human-robot collaborative environments and paves the path for formal reasoning for such interactions. Another contribution of the paper is the formalization of the necessary simulation conditions for manipulation abstractions in order to preserve correctness of reactive synthesis. We show that such conditions are indeed satisfied with our abstraction technique. Our abstraction algorithm is incorporated within an end-to-end reactive planner. The efficacy of this planner is demonstrated on a UR5 robot for complex pick-and-place tasks in the presence of a human who interferes with the task.

II. Related Work

Reactive synthesis has been widely used for finding strategies for mobile robots [2]–[4]. Recent works examined the extension of these techniques to the manipulation domain [6], [7]. In all of these works, an abstraction is required to map the continuous physical world to a discrete domain. This abstraction needs to maintain a simulation relation [2] with the continuous domain, i.e., there must be a mapping between the states and actions of the abstraction with the states and actions of the physical world. However, the construction of this abstraction is a challenging problem itself [6], [7].

Many recent studies have focused on the construction of such abstractions for robotic systems. Works in synthesis for the manipulation domain have used discrete graphs to capture the reachability of the objects by the robot [6]–[8]. These abstraction graphs, however, are mostly hand designed for particular scenarios, and their simulation relation guarantees to the continuous physical domain are not formally studied.

To assist manual construction of abstractions for mobile robots, [9] proposes a method where a graphical interface allows a user to perform an appropriate decomposition of the workspace in addition to specifying the task regions, and then a tool constructs the abstraction from this decomposition. Another study [10] proposes algorithms for providing feedback to the human on potential error in hand-made abstractions when synthesis cannot be done. These works do not scale to manipulators and highlight the difficulty in construction of appropriate abstractions. Furthermore, manual construction of abstractions is generally cumbersome and error-prone.

Works on automated construction [11]–[14] focus on mobile robots with dynamics, where tasks are defined over the locations of the robot in the workspace. These techniques rely on a 2- or 3-dimensional decomposition of the workspace and build controllers to enable navigation between neighboring regions. These approaches, however, cannot be extended to the manipulation domain. Manipulation tasks are defined over object locations, which can only be changed indirectly through the robot end-effector. Therefore, a workspace decomposition does not capture the possible actions of the domain. Additionally, it is impossible to determine the existence of robot motion between regions of the workspace without motion planning since objects blocking relations are difficult to compute for manipulators due their high DoF.

Besides reactive synthesis, another approach for handling interferences is replanning. This line of work is orthogonal to the work presented in this paper because, instead of seeking for a reactive strategy, fixed plans are repeatedly generated to deal with the changes in the environment. For mobile robots, replanning for complex tasks has been introduced in [15], [16]. For manipulation, online replanning is also possible for simple A-to-B tasks [17], [18]. For complex tasks, however, replanning may take too long during task execution [19]. More importantly, unlike reactive planning, replanning approaches cannot provide guarantees for the completion of the task.

For example, a robot that needs to choose one of two places to temporarily place an object before picking it up again later. One is on a table far away. The other is in a shelf that is closer, but the human may close the shelf door and prevent the robot from reaching the object. Reactive synthesis will place the object on the table to ensure the object can be reached later. Replanning will place the object in the shelf, as it produces an efficient plan. However, if the human closes the shelf door, the task can no longer be completed.

III. The Reactive Synthesis Problem

Reactive synthesis considers the problem of finding an execution strategy to achieve the specified task constraints under all allowed interferences by another agent. The input to this problem contains four main components: the domain, the task, the resource cost bound, and the number of allowed actions by the other agent. The solution is a strategy that dictates an action to the robot at every instance of time.

A. Manipulation Domain

Definition 1 (Manipulation Domain)**.** *For a set of movable objects O, the manipulation domain is a transition system $D = (C, c_0, U_s, U_e, F_D, \Pi, \rho_D)$, where*

- *$C = C_R \times C_O$ is the combined continuous configuration space of the robot C_R and the objects C_O.*
- *$c_0 \in C$ is the initial configuration.*
- *$U_s \subset \{u_s | u_s : C \to C\}$ is the set of robot actions, which are partial functions from one configuration to another.*
- *$U_e \subset \{u_e | u_e : C \to C\}$ is a set of allowed actions from other agents, which are partial functions from one configuration to another.*

- $F_D : C \times U_s \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ *is a cost function that determines the resource needed to perform actions. If* u_s *is undefined from* c*, then* $F_D(c, u_s) = \infty$.
- Π *is a set of propositions that relate to the task.*
- $\rho_D : C \to 2^{\Pi}$ *is the predicate function that determines the truth value of propositions in* Π *from the configuration.*

For the 6-DoF UR5 robot in Fig. 1, the configuration space of the robot is $C_R = \mathbb{R}^6$. The set of movable objects O includes the blocks in the figure, each with configuration space $SE(3)$. These combine for $C = \mathbb{R}^6 \times (SE(3))^3$.

We assume the set of robot actions can be partitioned into two types: $U_s = U_s^{\text{mot}} \cup U_s^{\text{pri}}$. U_s^{mot} are actions that require motion planning. U_s^{pri} are given as motion primitives. In this paper, we consider pick-and-place domains with $U_s^{\text{pri}} = \{\text{GRASP}, \text{PLACE}\}$, corresponding to grasping and placing of objects. These motion primitives are given with pre- and post-images relative to the object. A *pre-image* of GRASP (PLACE) is a set of configurations relative to an object, from which the robot can grasp (place) the object using the primitive GRASP (PLACE). The *post-image* is the final configuration after executing the motion primitive. Note that U_s^{pri} can be extended to other types of motion, e.g., push and pull.

An action in U_e is considered as an instantaneous movement of an object caused by the other agent (human). The function F_D maps actions to resource cost. In the arch example, we assume that motion primitives use 5 units of resources, while moving the robot arm with U_s^{mot} uses 1 unit of resources per radian of movement. We call $L = \{l_1, l_2, \ldots, l_{|L|}\}$, where $l_i \subset C_O$, a set of locations (configurations) of interest for the objects. In the arch example, L includes the supports and the top of the arch as well as two locations on the table to place the objects temporarily. ρ_D relates configurations to the task propositions by determining whether objects are at locations in L. Π includes propositions of the form $p_{o,l}$, indicating whether object $o \in O$ is in location $l \in L$. In the arch example $p_{\text{black,top}}$ indicates whether the black block is at the top.

B. Finite-Horizon Robot Task

To express the task, we use *LTL over finite traces* (LTLf) [20]. An LTLf formula φ combines boolean operators, i.e., $\wedge$ ("and"), $\neg$ ("not"), and $\to$ ("implies"), with temporal operators, i.e., $\Diamond$ ("eventually") and $\Box$ ("always"), over a set of propositions Π to describe how the propositions change over time. For syntax and semantics of LTLf, see [20]. In this paper, we accompany the task formula with a text explanation. For the arch example in Fig. 1, the LTLf formula is

$$\varphi_{\text{Arch}} = \Diamond\big(p_{\text{black,top}} \wedge p_{\text{white,support}_1} \wedge p_{\text{white,support}_2}\big) \wedge \Box\big(\neg(p_{\text{block,support}_1} \wedge p_{\text{block,support}_2}) \to \neg p_{\text{block,top}}\big), \quad (1)$$

which reads as "eventually the black block is on top with white blocks at the supports, and never place a block on top without objects at the supports."

C. Resource Cost and Action Bounds

The resource cost limit is given as a positive real number E. We assume that the other agent (human) takes at most $k \in \mathbb{N}$ number of actions from U_e. This assumption is necessary because a robot with limited resources cannot guarantee completion of the task if the other agent is allowed an unbounded number of actions.

D. Strategy

During an execution, the robot performs actions from U_s to progress toward task completion, while the other agent may perform actions from U_e at any time. A *trajectory* $h_t : [0, t] \to C$ is an execution in D up to time t. We denote the set of all such trajectories by H_t. For an h_t, the trace $\rho_D \circ h_T$ is the sequence of assignments to propositions in Π along h_t.

A *strategy* $P : H_t \to U_s$ determines the next action $u_s \in U_s$ for the robot to apply given the current h_t. For a given manipulation domain D, an LTLf task φ, a resource cost bound E, and action bound k, a *winning strategy* is a strategy that if followed and the other agent takes at most k actions from U_e, then there always exists a time T when the trace $\rho_D \circ h_T$ satisfies φ, and the total resource used determined by F_D does not exceed E.

E. Manipulation Reactive Synthesis Problem

Problem 1 (Manipulation Reactive Synthesis)**.** *Given a manipulation domain* D*, a task* φ*, a resource cost bound* E*, and a bound* k *on the number of actions of the other agent, find a winning strategy for the robot.*

F. Abstraction

The manipulation domain has an infinite state space. Existing techniques for reactive synthesis, however, work only on finite state spaces. We reduce the manipulation domain to a finite, discrete domain called the abstraction by grouping similar states into a single discrete state.

Definition 2 (Abstraction)**.** *An abstraction is a discrete transition system* $G = (V, v_0, A_s, A_e, F, \Pi, \rho)$*, where*

- V *is a finite set of states.*
- v_0 *is the initial state.*
- $A_s \subset \{a_s | a_s : V \to V\}$ *is the set of robot actions, which are partial functions from states to states.*
- $A_e \subset \{a_e | a_e : V \to V\}$ *is a set of allowed human actions, which are partial functions from states to states.*
- $F : V \times A_s \to \mathbb{R}^{\geq 0} \cup \{\infty\}$ *is a cost function that determines the resource needed to perform actions. If an action* a_s *is invalid from a state* v*, then* $F(v, a_s) = \infty$.
- Π *is a set of task-related propositions.*
- $\rho : V \to 2^{\Pi}$ *is the predicate function that determines the truth values of the propositions in* Π *from the states.*

On this discrete abstraction, a strategy takes as input a sequence of abstraction states and outputs an action from A_s. A strategy is winning for a task φ, resource cost bound E, and action bound k if every execution that follows the strategy results in task completion using resources no more than E, as long as the other agent takes no more than k actions.

G. Abstraction Construction Problem

Problem 2 (Abstraction Construction)**.** *Given a manipulation domain* D*, find an abstraction* G *with mapping* γ *from the states of* D *to the states of* G *such that for any task* φ*, resource*

cost bound E, and bound k on the other agent's actions, if there is a winning strategy for G, then there is a winning strategy for D.

The focus of this paper is a methodology for abstraction construction that solves Problem 2. This construction is presented in Sec. IV. In Sec. V, we show how this abstraction is used in an end-to-end framework to solve the reactive synthesis Problem 1. The correctness of the end-to-end framework is also proven in Sec. V. For simplicity, we refer to the other agent as the human in the reminder of the paper.

IV. Abstraction Construction

In this section, we describe an automated method for constructing an abstraction G from domain D given the locations of interest L and the objects O. The abstraction construction procedure is shown in Algorithm 1, which first generates the discrete state space (Sec. IV-A) and the actions of the abstraction (Sec. IV-B). It then constructs a set of states $Else$ to represent the portion of D that is not task-relevant and uses it to find a costmap $\mathscr{M}$ that describes the connectivity of the various locations. We bind this costmap to generate the resource cost function F (Sec. IV-C).

A. Abstraction States

This subsection corresponds to line 2 in Algorithm 1. We group states according to the location of the objects O in relation to the poses of the locations of interest L. Note that since the human is allowed to move objects, an object may be in none of the locations of interest.

We define a state $v \in V$ in the abstraction to be a tuple $v = (\mathscr{L}_1, \mathscr{L}_2, ..., \mathscr{L}_n, \mathscr{L}_{ee})$, where $\mathscr{L}_i$ is the discrete location of the ith object, and $\mathscr{L}_{ee}$ is the discrete location of the manipulator end-effector.

$\mathscr{L}_i$ can take on the following values:

- $l_j \in L$, indicating that the object is placed at the jth pose in the set of locations of interest,
- $Else$, indicating that the object placed at a location that is not at any of the poses in L,
- ee, indicating that the object is in the end-effector.

The assignment $Else$ signifies the object is at a pose other than a location of interest. This is grouped as a single assignment of $\mathscr{L}_i$ because the exact location of an object in $Else$ is not crucial to the task.

$\mathscr{L}_{ee}$ can take on the following values:

- $l_j \in L$, indicating that the end-effector is such that the object it is interacting with is aligned with the jth pose in the set of locations of interest,
- $\{o, a\}$, where $o \in O$ and $a \in A_s$, indicating that the end-effector is within the pre-image of the primitive action a relative to the object o,
- $Else$, indicating none of the above is true.

The values of $\mathscr{L}_{ee}$ signifies that the robot configuration is within the pre-image of a primitive action. For pick-and-place domains, l_j indicates the robot configuration is in the pre-image of PLACE. $\{o, a\}$ indicates the robot configuration is in the pre-image of the primitive GRASP action for object o using grasp method a.

Algorithm 1 Abstraction Construction Procedure

1: **procedure** ConstructAbstraction(D, L, O)
2: $(V, v_0, \rho) \leftarrow$ ConstructStateSpace(D, L, O)
3: $(A_s, A_e) \leftarrow$ ConstructActions($|L|, |O|$)
4: $Else \leftarrow$ SampleConfigurations()
5: $\mathscr{M} \leftarrow$ ConstructCostMap($L \cup Else, O$)
6: $F \leftarrow$ Bind GetCost() with $\mathscr{M}$
7: **return** $D = (V, v_0, A_s, A_e, F, \Pi, \rho)$

The initial state v_0 is found by mapping the initial domain state c_0 to the abstraction. The set of propositions Π is the same as the set of propositions in the manipulation domain. The predicate function ρ is determined by the location of the objects with respect to L.

Note that not all state tuples are physically feasible. In particular, we remove all states where multiple objects are assigned the same location value. $\mathscr{L}_{ee}$ is only allowed to take on values from $\{o, a\}$ if there is no object in the end-effector, and to take on values from L if there is an object in the end-effector. We define the set V to be the set of tuples v that do not violate any of the rules above.

Lemma 3. *There is a mapping γ from the set of configurations in the manipulation domain $C = C_R \times C_O$ to the states of the abstraction V that preserves the predicate function $\rho(\gamma(c)) = \rho_D(c), \forall c \in C$.*

Proof. For any given configuration $c \in C_R \times C_O$, γ maps c to the abstraction state $v = (\mathscr{L}_1, \mathscr{L}_2, ..., \mathscr{L}_n, \mathscr{L}_{ee})$, where $\mathscr{L}_i$ is determined by the projection of c onto the components that relate to object C_{O_i}, and $\mathscr{L}_{ee}$ is determined by forward kinematics on the robot component of c onto C_R. Since each location relevant to the predicate function of the manipulation domain is a location of interest for the abstraction, ρ is naturally preserved. □

B. Human and Robot Actions

This subsection corresponds to line 3 in Algorithm 1. We define the human actions A_e as $\{a_{e,o,t} | o \in O, t \in L \cup \{Else\}\}$. Intuitively, $a_{e,o,t}$ is the human action of moving object o to target t. The location is either a location of interest L or $Else$.

Lemma 4. *The abstraction $G = (V, v_0, A_s, A_e, F, \Pi, \rho)$ simulates the manipulation domain $D = (C, c_0, U_s, U_e, F_D, \Pi, \rho_D)$ with respect to the human actions under the mapping γ, i.e., for every $c_1, c_2 \in C$ and $u_e \in U_e$ such that $u_e(c_1) = c_2$, there exists $a_e \in A_e$ such that $a_e(\gamma(c_1)) = \gamma(c_2)$.*

Proof. The manipulation domain has human actions that instantaneously move objects around. Since this only affects one object location, the corresponding state in the abstraction only changes one value. In G, A_e captures all such possibilities of moving a single object on a discrete level. □

We define the robot actions A_s as $\{Grab\} \cup \{Drop\} \cup \{Transit_{o,a}\} \cup \{Transfer_l\}$, where:

- $Grab$ corresponds to GRASP. This sets $\mathscr{L}_o$ to ee, and $\mathscr{L}_{ee}$ takes the previous value of $\mathscr{L}_o$,
- $Drop$ corresponds to PLACE. This sets $\mathscr{L}_{ee}$ to be aligned with the object o at the previous grasp a, and $\mathscr{L}_o$ becomes the previous location of the end-effector l,

- $Transit_{o,a}$ corresponds to the actions in U_s^{mot} that move the end-effector to the pre-image of applying the action a to an object $o \in O$,
- $Transfer_l$ corresponds to the actions in U_s^{mot} that move the end-effector to a location of interest $l \in L$.

The *Transit* action is only useful when taken from a state $v \in V$ with no object in the end-effector, and the *Transfer* action is only useful when there is an object in the end-effector. Otherwise, we disallow such transitions. Note that A_s is only a subset of the actions that the physical robot can perform. For example, the robot may choose to transit to an arbitrary location in its configuration space in the air. Such actions are not useful for task completion and thus omitted.

Note that objects in $Else$ can have an infinite number of configurations and may be blocking each other. Instead of listing all possible blocking scenarios, we assume that at least one primitive a_s is available for at least one of the objects o in $Else$. This assumption allows us to eventually access every object in $Else$ by moving the objects that block it to other locations of interest.

C. Robot Action Cost

We complete the abstraction by finding the cost function F (Line 4-6 in Algorithm 1). For problems where resource cost is not considered, the steps in this subsection are still needed to find the availability of the robot actions.

A fixed cost is assigned to *Grab* and *Drop* actions as they correspond to primitive actions GRASP and PLACE. The cost of actions in U_s^{mot} depends on the path the robot takes to perform them. Longer paths often require more energy. We use motion planning to find the cost for the transfer and transit actions. Note that the number of configurations that map to a given $v \in V$ may be infinite. For a state with $\mathscr{L}_o = Else$, o can be placed anywhere other than the locations of interests. Thus motion planning for every object configuration is infeasible. In practice, we sample the configurations the objects can take (Line 4 in Algorithm 1). In section VI, we show that a sampled set of the tabletop locations is sufficient to produce a correct abstraction for a practical problem.

Here we discuss three approaches for finding the cost of *Transfer* and *Transit* actions: a strawman brute-force approach, a start-goal based approach that is fast but may lead to failed synthesis, and a recursive start-goal based approach that provides a good approximation for the cost.

1) Brute Force Approach: We can find the action cost function by motion planning for each action from each state. This requires a motion planning call for each state-action pair. However, the number of states is exponential in the number of objects. Thus an exponential number of motion planner calls are needed for this approach, making it intractable.

2) Start-Goal Based Approach: To make the construction of the abstraction tractable and efficient, we rely on two insights. First, if across multiple motion planning calls, the robot end-effector is holding the same object with the same grasp, collision checking against static obstacles such as tables and immovable objects only needs to be performed once. Second, moving an object between the same pair of start and goal configurations could use the same path, if none of the other movable objects are blocking the path. Using these insights, we provide a start-goal based approach for finding the cost of actions.

Algorithm 2 Start-Goal Based Approach

```
1:  procedure ConstructCostMapSG(L+, O)
2:      M ← PRM(L+, O)
3:      Initialize cost map 𝓜 : L+ × L+ → ℝ
4:      for all (l1, l2) ∈ L+ do
5:          Create obstacles at L+ \ {l1, l2}
6:          𝓜(l1, l2) ← Cost(BestPath(M, l1, l2))
7:      return 𝓜
8:  procedure GetCost(𝓜, Before, After)
9:      {(s, g)} ← All pairs associated with Before and After
10:     return max_{s,g}(𝓜(s, g))
```

Our algorithm is shown in Algorithm 2. For each possible object-grasp as well as the arm with no object grasped, we construct a probabilistic roadmap M in the configuration space of the robot, while only considering collision with static obstacles (Line 2). This roadmap is initialized with all of the locations L^+ that include the locations of interest L and a set of configurations that represent $Else$. The roadmap is grown such that all the initial configurations are connected. For each pair of configurations in L^+, we find the best (lowest-cost) path in M given that the movable objects exists in all other places (Line 5, 6). This path is guaranteed to be available regardless of where the objects are placed. We store the cost of these paths in a costmap $\mathscr{M}$.

To find the cost of a *Transit* or *Transfer* action from a state, we find $\mathscr{L}_{ee}$ before and after the action. We then gather all the configurations corresponding to these $\mathscr{L}_{ee}$ values in the roadmap. For $\mathscr{L}_{ee} = Else$, multiple configurations may be included. We then use the maximum cost from all instantiations of (s, g), where s and g are configurations before and after the action, as the cost of the action (Line 9, 10).

Definition 5. *The domain $D = (C, c_0, U_s, U_e, F_D, \Pi, \rho_D)$ simulates the abstraction $G = (V, v_0, A_s, A_e, F, \Pi, \rho)$ with respect to the robot actions under the mapping γ if for every $c_1 \in C$, $v_1, v_2 \in V$ and $a_s \in A_s$ such that $\gamma(c_1) = v_1$ and $a_s(v_1) = v_2$, there exists $u_s \in U_s$ such that $\gamma(u_s(c_1)) = v_2$ and $F_D(u_s, c_1) \leq F(a_s, v_1)$.*

Lemma 6. *The manipulation domain D simulates the abstraction G constructed with the start-goal based approach.*

Proof. Each time we record a cost in F, we have an evidence trajectory in the roadmap that is valid even if objects exist in all of the locations in consideration. For any configuration c, if an action a_s exists in the abstraction from $\gamma(c)$, then we can find the corresponding evidence trajectory in the roadmap that costs $F(\gamma(c), a_s)$. Thus to execute the corresponding action u_s, the resource we need, $F_D(c, u_s)$, does not need to exceed $F(\gamma(c), a_s)$. □

This computation is much more efficient than the brute-force approach. The number of roadmaps required is linear in the number of objects and grasps, and the number of start-goal pairs is quadratic in $|L^+|$. Thus we reduce the total runtime

Algorithm 3 Recursive Start-Goal Based Approach

1: **procedure** CONSTRUCTCOSTMAPREC(L^+, O)
2: Initialize cost map $\mathscr{M} : L^+ \times L^+ \to 2^{\text{Conditions} \times \mathbb{R}}$
3: $M \leftarrow PRM(L^+, O)$
4: **for all** $(l_1, l_2) \in L^+$ **do**
5: $\mathscr{M}(l_1, l_2) \leftarrow$ ConditionalBestPath($M, l_1, l_2, L^+, \{\}$)
6: **return** $\mathscr{M}$
7: **procedure** CONDITIONALBESTPATH(M, l_1, l_2, L^+, obs)
8: Create obstacles at obs
9: $\mathscr{P}$ = BestPath(M, l_1, l_2)
10: $new = \{l \in L^+ \backslash obs \mid$ obstacle at l collides with $P\}$
11: Add $(new, Length(\mathscr{P}))$ to $\mathscr{M}(l_1, l_2)$
12: **for all** $l \in new$ **do**
13: ConditionalBestPath($M, l_1, l_2, L^+, obs \cup \{l\}$)
14: **procedure** GETCOST($\mathscr{M}, Before, After, v$)
15: $\{(s, g)\} \leftarrow$ All pairs associated with *Before* and *After*
16: **return** $max_{s,g}(min_{v \models \text{condition}} \mathscr{M}(s, g, \text{condition}))$

from an exponential to only a cubic number of motion-planner calls.

3) Recursive Start-Goal Based Approach: The above start-goal based approach gives a very rough over-approximation of action cost since we assume that objects always exist at all the other locations. This presents two issues: (i) some valid actions are deemed impossible, and (ii) the abstraction does not maintain insight on which location is causing a blocking. For example, if an object at location l_1 is blocking a path between l_2 and l_3, then the abstraction should capture this relation to allow the synthesis tool to infer that the objects in l_1 need to be removed to allow movement from l_2 to l_3.

We thus modify Algorithm 2 and present Algorithm 3. The key modification is that the costmap $\mathscr{M}$ no longer stores a single cost for a start-goal pair. Instead, it stores conditional action costs, where each condition is essentially a set of locations that need to be free of objects.

During costmap construction, for each start-goal pair, we now find the best path conditioned on where obstacles are. For a given set of locations with obstacles (initially empty), we compute the best (lowest-cost) path $\mathscr{P}$ in M (Lines 10, 11). We then perform collision checking for $\mathscr{P}$ against obstacles at the other locations of interest, and record the new locations *new* where collision occurs (Line 12). We add a rule indicating that if obstacles do not exist at these locations, the cost of the action is the cost of $\mathscr{P}$ (Line 13). Then, recursively for each location in *new*, we find the shortest path given that obstacles are added there (Line 15).

When retrieving the cost, we now check if the abstraction state v satisfies the condition. The cost between two locations s and g is the minimum cost of all the satisfying rules.

Lemma 7. *The manipulation domain D simulates the abstraction G constructed with the recursive start-goal based approach.*

Proof. The proof is identical to that of Lemma 6. □

In the worst case, this algorithm performs the same order of motion planning calls as the brute-force algorithm. In practice, however, we see that the least-cost path between the start and goal in the roadmap is often collision-free, and recursion rarely reaches the point where all objects are placed. In Sec. VI, we see that this approach is indeed tractable.

V. REACTIVE STRATEGY SYNTHESIS

In the previous section, we described how the abstraction is constructed, thus solving Problem 2. For the robot to perform reactive tasks, i.e., to solve Problem 1, we first perform reactive synthesis on the abstraction to find a discrete strategy. Then, we map this strategy back to the continuous domain.

A. Discrete Reactive Synthesis

The technique for performing reactive synthesis problem on the abstraction is described in [6]. Here, we provide a brief overview. First, the LTLf formula φ is converted into a deterministic finite automaton A_φ. Then, A_φ is combined with the abstraction G to build a quantitative game played between the robot and the human, where the robot wins the game if a final state is reached. A fixed-point approach, i.e., backward propagation from the final state until a fixed point is reached, is used to compute a strategy P that minimizes the resource cost while forcing the game into a final state. Strategy P guarantees the completion of φ with a resource cost at most E_p, i.e., E_p is reached only against the worst k human actions. Hence, P is insured to be a winning strategy if $E_p \leq E$. Otherwise, there is no guarantee that P is winning, but note that φ may still be achieved with a cost less than E under P if the human does not perform the worst actions.

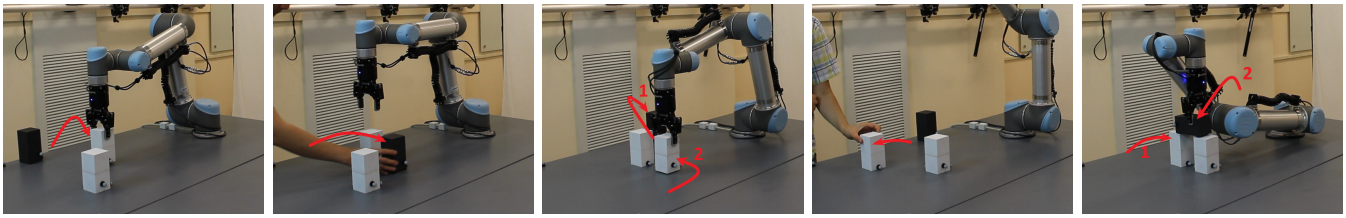
B. Mapping to a Continuous Strategy

Once a (winning) strategy P is synthesized on the abstraction G, we map P to the manipulation domain D. For an execution h_T, we record the abstraction state whenever $\gamma(h_T(t))$ changes. This produces a trace on the abstraction. We can then use this trace to find a corresponding abstraction action a_s and execute it on the robot in D. Note that P and E_p are optimal on G and may be sub-optimal on D.

In practice, a visual system can be used to monitor the state of the manipulation domain and translate this to an abstraction state. *Grab* and *Drop* actions are implemented by calling the corresponding GRASP and PLACE primitives, which involve performing pre-determined Cartesian trajectories using inverse kinematics, followed by closing and opening of the gripper. For *Transit* and *Transfer* actions which correspond to U_s^{mot}, a sampling-based optimizing motion planner is called at runtime. Since every action a_s chosen by the strategy has a finite cost $F(a_s, v)$, a probabilistic complete sampling-based optimizing planner is able to find such a path if given enough time.

Note that if objects in *Else* are blocking other objects in *Else*, we rely on the assumption in Sec. IV that at least one of these objects is reachable. Thus, when a *Transit* action to an object in *Else* is needed, we perform motion planning for all objects we are interested in at runtime and choose the best outcome to perform.

Theorem 8. *For a given LTLf task φ, a resource cost bound E, a bound on human actions k, a manipulation domain D and an abstraction G, if a strategy P is a winning strategy*



(a) (b) (c) (d) (e)

Fig. 2: Execution of the arch construction task. (a) Robot places the first support block. (b) The human incorrectly moves the black block to the support location. (c) The robot moves the black block away (behind gripper) and then places the second white support. (d) The human moves away a support. (e) The robot recovers the support and places the black block on top. Video: https://youtu.be/UJ5p-Lq4e7A.

for (G,φ,E,k)*, D simulates G for the robot actions with a mapping* γ*, and G simulates D for the human actions with* γ*, then there is a winning strategy for* (D,φ,E,k).

Proof. Let P_{cont} be the continuous strategy in D that is mapped from the winning strategy P as described above. P_{cont} is winning if every trace $h_t : [0,t] \to C$ under P_{cont} results in the completion of the task with total resource consumption less than E as long as the human performs no more than k actions. Under P_{cont}, $\gamma \circ h_t$ is a valid trace in G as every transition is either a robot action suggested by the abstraction or a human action modeled by the abstraction (Lemma 4). The task must also be completed by h_t in D because, by Lemma 3, the corresponding trace $\gamma \circ h_t$ in G generates an identical sequence of propositional assignments that satisfies φ. Finally, the total resources used is within the bound E because, by Lemmas 6 and 7, every robot action performed in D accumulates no more cost than the corresponding action in G. Hence, P_{cont} is winning in D. □

Thus, we can solve Problem 1 by constructing an abstraction G per Sec. IV, synthesizing a winning strategy P on G, and mapping P to the continuous manipulation domain D.

VI. Experiments

To demonstrate the effectiveness and efficiency of abstraction construction, we implemented our abstraction algorithm with both options of the start-goal based approach and the recursive start-goal based approach within the reactive synthesis framework in [6]. Roadmap construction was performed using MoveIt! with a custom OMPL planner. A UR5 robot was used to demonstrate the strategy. Object locations were captured using a Vicon system. Online motion planning was performed with MoveIt! with RRT* [21].

We tested our implementation on two scenarios where the UR5 had to manipulate blocks. Two grasp primitives were provided, one approaching from the top and one from the side. The two scenarios demonstrate the power of the automated abstraction construction technique and show the trade-offs between Algorithms 2 and 3.

A. Arch Construction

The first scenario we examined is the arch example in Fig. 1. The robot must create an arch with white blocks as supports and the black block on top. The LTLf formula for this task is φ_{Arch} in (1). The human is allowed 5 actions.

In this scenario, synthesis with the start-goal based abstraction fails to find a valid solution. This happens because we assume movable obstacles always exist at all locations besides the ones we are moving from and to. Thus, to place an object at the support, the algorithm assumes an object is at the top, causing a collision in the path. This shows that the non-recursive start-goal abstraction may not be able to find a solution due to the overly conservative approximation.

On the other hand, synthesis with the recursive start-goal based abstraction was able to find a valid solution. In 10 trials, the abstraction computation took 485.0 ± 68.3 seconds and the strategy resource cost was 313.5 ± 46.3. Fig. 2 shows an execution of the strategy, illustrating the completion of the arch despite human interferences.

Note that in this execution, the robot specifically reached for the black block using a side grasp, knowing that it is more efficient for placing the top of the arch. Insights such as which grasp to use can be difficult for a human to gather, especially if the number of objects is large. The automation of abstraction construction allows the user to avoid manually writing up the availability of actions and their costs.

B. Straight Line Alignment

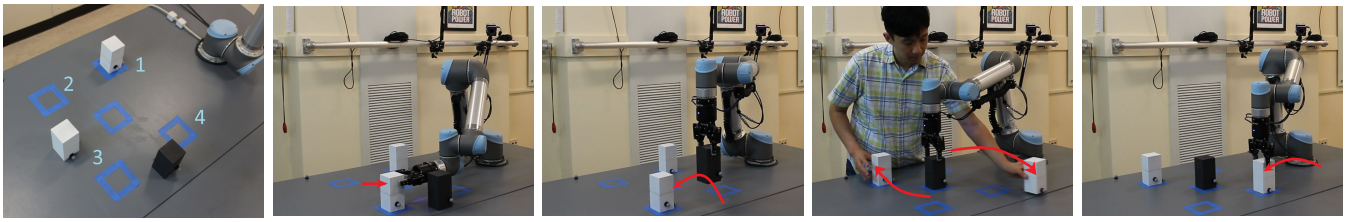
The second scenario we examined is a pick-and-place task with three objects and five locations of interest placed in a cross shape, as shown in Fig. 3a. The task requires all three blocks to be aligned in a straight line with the black block in the middle. This task is expressed as $\varphi_{\text{line}} =$

$$\Diamond\Big(p_{\text{black,center}} \wedge \big((p_{\text{white},1} \wedge p_{\text{white},3}) \vee (p_{\text{white},2} \wedge p_{\text{white},4})\big)\Big).$$

The human is allowed five actions. The reactive synthesis tool is run until convergence to examine the least amount of resource bound needed for a winning strategy to be found.

Both the non-recursive and recursive approaches were able to produce an abstraction in all the 10 trails. The computation times were 79.1 ± 4.8 and 449.7 ± 22.9 seconds for the non-recursive and recursive abstractions, respectively. These abstractions resulted in resource bounds of 593.4 ± 37.6 and 559.6 ± 34.7 units, respectively. Such a difference (trade-off) in the results of the two methods is expected since the recursive approach works harder in identifying blocking objects and finding alternative ways to perform the action, leading to higher computation times in the abstraction step. This greater effort is rewarded by lower resource bounds in the synthesis step. Given this trade-off, one can imagine using the start-goal method first. If no winning strategy can be found, then the recursive approach can be employed.

Fig. 3 shows an execution of this task with an abstraction from the recursive approach. We can see that the robot initially



(a) (b) (c) (d) (e)

Fig. 3: Execution of the line task. Locations are marked in blue. (a) Initial condition (top view). (b) The robot places the white objects at 1 and 3. (c) The robot reaches for the black object to place at the center. (d) The human tries to do the task using 2 and 4, but places one too far away. (e) The robot fixes the human's mistake by moving the white block back to 4. Video: https://youtu.be/EGjrkirLYYs.

tried to complete the task using locations 1 and 3. Yet the human wanted to complete the task using locations 2 and 4, and misplaced an object in the process. The robot completes the task efficiently by correcting the human's mistake.

To further analyze the performance of the algorithms, we iteratively added obstacles to the spaces in between the locations of interest L (1-4) in Fig. 3a. The results (mean and standard divination) based on 10 trials are shown as in Fig. 4. Overall, we observe the general trend of lower abstraction times and higher resource bounds for the non-recursive method compared to the recursive approach holds for all cases. A surprising observation is that the abstraction times are not significantly affected by the number of obstacles. That is because once a roadmap is generated, the time to find a path only depends on the blocking relations of the objects in L, not the ones outside of L. These obstacles, however, can affect the resource cost. Depending on their locations, they can either increase it due to longer paths or reduce it due to causing more roadmap samples in the free space.

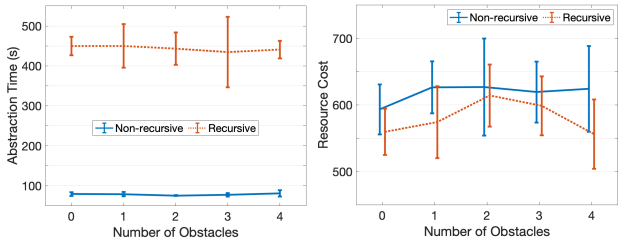


Fig. 4: Results for φ_{line} with increasing number of obstacles.

VII. Conclusion

In this paper, we introduce the first automated methods of abstraction construction of a manipulation domain for the purposes of reactive synthesis. We prove that these abstractions guarantee the correctness of the computed strategies by the reactive synthesis algorithms. Despite the focus on finite tasks with resource cost limits, our manipulation abstractions are general to all reactive synthesis formulations such as [2]–[4] and can be used to extend the capabilities of these approaches to the manipulation domain. Some possible future directions for this work are: a symbolic approach to reactive synthesis, human-robot task conflict resolution, and probabilistic abstraction for noisy manipulators.

References

[1] D. Harel and A. Pnueli, "On the development of reactive systems," in *Logics & Models of Concurrent Sys.* Springer, 1985, pp. 477–498.

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[3] C. I. Vasile and C. Belta, "Reactive sampling-based temporal logic path planning," in *Int. Conf. on Robotics & Automation*. IEEE, 2014, pp. 4310–4315.

[4] E. M. Wolff, U. Topcu, and R. M. Murray, "Efficient reactive controller synthesis for a fragment of linear temporal logic," in *Int. Conf. on Robotics and Automation*. IEEE, 2013, pp. 5033–5040.

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

[6] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Reactive synthesis for finite tasks under resource constraints," in *Int. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 5326–5332.

[7] S. Maniatopoulos, P. Schillinger, V. Pong, D. C. Conner, and H. Kress-Gazit, "Reactive high-level behavior synthesis for an atlas humanoid robot," in *Int. Conf. on Robotics & Automation*. IEEE, 2016, pp. 4192–4199.

[8] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki, "SMT-Based Synthesis of Integrated Task and Motion Plans from Plan Outlines," *Int. Conf. on Robotics & Automation*, pp. 655–662, 2014.

[9] C. Finucane, G. Jing, and H. Kress-Gazit, "Ltlmop: Experimenting with language, temporal logic and robot control," in *Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 1988–1993.

[10] V. Raman and H. Kress-Gazit, "Explaining impossible high-level robot behaviors," *IEEE Transactions on Robotics*, vol. 29, pp. 94–104, 2013.

[11] P. Tabuada, G. J. Pappas, and P. Lima, "Compositional abstractions of hybrid control systems," *Discrete event dynamic systems*, vol. 14, no. 2, pp. 203–238, 2004.

[12] A. Girard, G. Pola, and P. Tabuada, "Approximately bisimilar symbolic models for incrementally stable switched systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 1, pp. 116–126, 2010.

[13] J. A. DeCastro and H. Kress-Gazit, "Guaranteeing reactive high-level behaviors for robots with complex dynamics," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2013, pp. 749–756.

[14] E. A. Gol, X. Ding, M. Lazar, and C. Belta, "Finite bisimulations for switched linear systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3122–3134, 2014.

[15] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *Int. Conf. on Robotics and Automation*. IEEE, 2013, pp. 5025–5032.

[16] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal planning in uncertain environments with partial satisfaction guarantees," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 538–599, 2016.

[17] Y. Yang and O. Brock, "Elastic roadmapsmotion generation for autonomous mobile manipulation," *Autonomous Robots*, vol. 28, no. 1, pp. 113–130, 2010.

[18] C. Bowen and R. Alterovitz, "Closed-loop global motion planning for reactive execution of learned tasks," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2014, pp. 1754–1760.

[19] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *Int. Conf. Robotics and Automation*. IEEE, 2015, pp. 346–352.

[20] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Int. Conf. on Artificial Intelligence*. AAAI Press, 2013, pp. 854–860.

[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.