To appear in the Proceedings of the 2017 IEEE Int. Conf. on Intelligent Robots and Systems (IROS 2017)

Reactive Synthesis For Finite Tasks Under Resource Constraints

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

***Abstract*— There are many applications where robots have to operate in environments that other agents can change. In such cases, it is desirable for the robot to achieve a given high-level task despite interference. Ideally, the robot must decide its next action as it observes the changes in the world, i.e. act reactively. In this paper, we consider a reactive planning problem for finite robotic tasks with resource constraints. The task is represented using a temporal logic for finite behaviors and the robot must achieve the task using limited resources under all possible finite sequences of moves of other agents. We present a formulation for this problem and an approach based on quantitative games. The efficacy of the approach is demonstrated through a manipulation case study.**

I. INTRODUCTION

With recent advances in robotics technology, robots today have incredible capabilities to perform complex tasks. Such capabilities, however, are generally limited to static environments [1]–[4]. Major challenges arise when a robot attempts to execute tasks with similar complexity in a dynamic environment that involves other agents. For example, a robot waiter needs to handle the changes to the environment caused by humans. In these scenarios, the robot is expected to achieve high-level tasks despite interference. Thus, classical robot planning that only considers a single sequence of actions is insufficient. The robot must instead perform the task *reactively*, i.e., decide on the next action as it observes the changes in the world [5]–[10].

Another dimension of difficulty emerges when considering resource constraints such as time or energy [11], [12]. For instance, a robot waiter must serve customers within a reasonable time. Ideally, robots need to account for their remaining resource budget in their choice of the next action to avoid depleting its resources before completing the task.

In this work, we formulate a problem to demonstrate the above challenges in order to understand the methodologies needed to address them. We consider the scenario where a robot needs to achieve a finite high-level task when a human can interfere. In particular, the human is agile enough to interfere multiple times while the robot performs a single action, but the human can only interfere a bounded number of times. The robot must complete the task despite the interference, while using a limited amount of resources. To illustrate the setting, consider the following simple example:

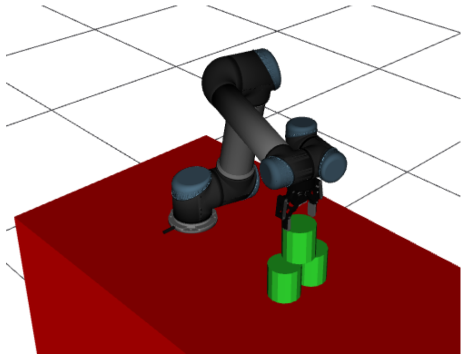
Fig. 1: The cup-stacking problem

Cup-stacking example: A robotic arm is to complete the cup-stacking challenge, where it has to stack cups in a pyramid in the middle of the table (shown in Figure 1). The task requires that a cup in a higher row must not be placed unless both its supporting cups are in place. In this challenge, a human may interfere and move any of the cups at will up to 5 times, and the robot must be able to complete the task with less than 3000 units of energy.

In this example, the robot must move several cups to complete the cup-stacking task, but the human can interfere with the task by moving the cups. If the robot does not react to the changes made by the human, it may violate its task requirement by placing a cup in the higher row when a base is already removed. The robot may also deplete its energy budget if all human moves are not carefully considered. Therefore, the robot must have a strategy that chooses its action according to not only the task but also the human's possible actions.

In the general case, finding a strategy for reactive tasks with resource constraints is difficult. In this paper, we show that if we can obtain a structure called an *abstraction* that captures the planning domain and a *linear temporal logic on finite traces* (LTLf) [13] formula that captures the task, then we can produce a strategy that guarantees the completion of the task in this domain without exceeding the resource limit.

The contributions of this paper are three-fold. First, we define the problem of reactive synthesis for finite tasks with resource constraints, a problem previously unstudied. Second, we present a framework for finding a winning strategy for such problems under the assumption that an abstraction is provided. Finally, we use a proof-of-concept manipulation example to show how robotics problems can be solved using this framework.

The structure of the paper is as follows. In Section II, we discuss previous work related to the problem above. In Section

[1]Keliang He, Lydia E. Kavraki, Moshe Y. Vardi are with the Dept. of Computer Science, Rice University, Houston, TX 77005, USA, and are supported in part by CCF 1139011, NSF IIS 1317849 and Rice University Funds. keliang.he@rice.edu, kavraki@rice.edu, vardi@cs.rice.edu.

[2]Morteza Lahijanian is with the Dept. of Computer Science, University of Oxford, UK and is supported by EPSRC Mobile Autonomy Program Grant EP/M019918/1. morteza.lahijanian@cs.ox.ac.uk.

III, we define the synthesis problem and the inputs required for us to solve the problem. In Section IV, we describe how we solve the reactive synthesis problem by combining the task with the abstraction to generate a *quantitative game* played between the robot and the human. In Section V, we present a case study where the cup-stacking problem is expressed using our formulation, and solved using our approach. Finally, we conclude in Section VI with a discussion on required properties for a scenario to be captured by an abstraction.

II. Related Work

Though the problem above is previously unconsidered, several important aspects of it have been studied in the field. Here we discuss the relation between these previous work and the problem studied in this paper.

Reactive synthesis has been studied extensively for mobile platforms [5]–[8], [14]. In these works, the specification language of choice is typically *linear temporal logic* (LTL) or its fragment GR(1) [15], which describe infinite behaviors such as surveillance and respondence to human requests. The complexity for these works is in the infinite behavior of the system, while in this paper we are interested in finite behavior along with resource constraints.

Several works in reactive synthesis consider time constraints [9], [10]. The tasks in these works are defined by satisfaction of properties (sub-tasks) within specific time windows. These problems are challenging due to the complex dynamics of the system. However, these works only reason about time constraints, and their extensions to general resource constraints that accumulate during execution is non-trivial.

There exist several works that consider finite tasks in uncertain environments [16], [17]. These works use a fragment of LTL called *co-safe* LTL [18] to specify such tasks. To deal with environment changes, they take an iterative approach, which replans every time a change is detected. With this approach, the completion of the task cannot be guaranteed. In this work, we guarantee the task is achieved despite the changes in the environment.

This work is closely related to planning for nondeterministic systems [19]. In nondeterministic systems, unintended (but modeled) events may occur at certain states during execution, similar to human interference in this paper. Such events are handled by constructing alternative plans from the points of deviation. In this work, since the human could interfere from any state, constructing alternative plans from every single point during execution would be computationally intractable.

Generation of supervisory controllers for *discrete event systems* (DES) is also closely related to synthesis for reactive systems [20]. In DES, controllable and uncontrollable events could occur depending on the state of the system, and the objective is to remove controllable events to guarantee satisfaction of certain properties [21]. This model of events is similar to the turn-taking modeled in this paper. However, for robot reactive synthesis, we need to choose exactly one action for the robot in each state, as opposed to keeping a maximal set of behavior.

In our effort to address the problem of reactive synthesis with resource constraints, we capture the planning domain with a discrete structure. This opens the door to solving the problem using approaches from quantitative games. Previous work in quantitative games [22], [23] largely focused on infinite games, making direct application of their algorithms to our problem difficult. Nonetheless, we adapt techniques from [22] to solve our problem, and show that this is a promising approach.

III. Problem Formulation

The goal of this work is to synthesize robot behavior to achieve finite reactive tasks without expending more than a limited amount of resources. To do this, the user needs to provide information regarding the planning domain as well as the task. In this work, we assume the robot and human operate in a domain that can be captured by a discrete graph called an *abstraction*. The abstraction encodes a finite subset of behavior by the robot and the human that is task relevant. To solve this problem, we take as input 1) an abstraction of the domain G, 2) a task specified as an LTLf formula ϕ, 3) a bound on resource allowed for the robot E, and 4) a bound on the number of times the human may interfere K. From these inputs, we wish to generate a strategy Str that can guarantee the completion of the task. We now describe in detail the input abstraction, how the abstraction is interpreted, the input LTLf formula, and the output strategy.

A. Abstraction

We assume we are given as input a discrete structure called the abstraction that captures the relevant parts of the planning domain. This structure is inspired by our previous work [3]. In section V, we discuss in detail how such an abstraction can be constructed using the cup-stacking problem as an example. Here, we formally define the abstraction.

The abstraction is a graph $G = (V, v_0, A_s, A_e, F_G, \Pi, \rho)$, where V is a finite set of vertices and v_0 is the initial vertex. Intuitively, each vertex in V encodes important information regarding the state of the world. For example, in the cup-stacking problem, a vertex in V encodes whether each cup is in a location of interest to the task and whether the robot gripper is aligned with any cup.

There are two sets of edges, A_s and A_e, where each $a_s \in A_s$ and each $a_e \in A_e$ are deterministic functions from V to V. These edges represent robot actions and human actions respectively. If the world is currently represented by v, and the robot takes an action a_s, the new state of the world would be represented by the vertex $a_s(v)$. In the cup-stacking example, A_s would include opening and closing of the gripper and moving the manipulator arm, while A_e would include the human moving objects around. The mapping $F_G : V \times A_s \rightarrow \mathbb{R}_{\geq 0}$ is the resource cost map that tells us the amount of resources needed in order to perform a robot action from a particular state.

Π is a set of boolean predicates that represent important facts about the world that is relevant to the task. For example, in the cup-stacking problem, one of the predicates could represent whether a particular cup is at the top of the stack. $\rho : V \to 2^{\Pi}$ is the predicate function that outputs the set of predicates that are true in the state represented by a vertex.

B. Understanding The Abstraction

We now describe the intuition behind the abstraction to show that it indeed captures behavior we wish to consider. An execution on the abstraction is a sequence of vertices $(v_0, v_1, ..., v_n) \in V^*$ where v_0 is the initial vertex, and for each v_i and v_{i+1}, there is some action $a \in A_s \cup A_e$ such that $a(v_i) = v_{i+1}$. We further restrict the evolution in the following way. If both the human and the robot wish to take an action, the human action will be taken. The robot edge is only taken when the human decides to not take an action.

Recall that we wish to model executions where the human may perform several actions while the robot performs only one. By formulating execution in the manner above with priority given to the human, the interaction between simultaneous actions from the robot and the human is correctly modeled. Consider the following scenarios from the cup-stacking problem. Starting from a vertex v, the human moves two cups using actions a_{e1}, a_{e2}, while the robot is moving its arm using action a_{s1}. In this case, the graph evolves by first the human actions, and then the robot action. Therefore the resulting vertex is $a_{s1}(a_{e2}(a_{e1}(v)))$, which is the correct behavior. Consider another scenario where from a vertex v, both the robot and the human are trying to grasp the same cup with actions a_s and a_e respectively. In this situation we must consider two possible outcomes. If the human successfully moves the cup with a_e, then the robot's action would fail. This case is represented by the human choosing to take action a_e, and the robot action is ignored. If the human is slower than the robot in grasping, the human action would fail, and the robot action will succeed. This case is represented by the human choosing to not take an action (yet), and allowing the robot to perform a_s. In both cases, the execution is encoded in the abstraction, and therefore must be considered when finding a solution.

Though giving human actions precedence over robot actions is uncommon, we find this appropriate for the scenario, as the human is often agile enough that he/she could perform multiple actions while the robot performs a single action. This modeling also makes the robot conservative, forcing it to consider all possible combinations of human actions.

C. LTLf Task Specification

To express a high-level task that requires many steps, we reason over the truth value of properties in predicates Π over the duration of the execution. In this paper, we use *linear temporal logic on finite traces*, LTLf, as the tasks we wish to express are finite.

An LTLf formula ϕ is defined over a set of boolean atomic propositions Π with the following syntax.

$$\phi = p \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \circ\psi \mid \psi_1 \mathcal{U} \psi_2,$$

where $p \in \Pi$ is an atomic proposition, and ψ, ψ_1, and ψ_2 are also LTLf formulas.

LTLf formulas reason over finite sequences of letters $w = (w_0, w_1, w_2, ..., w_n)$, called traces. Each letter $w_i \in 2^{\Pi}$ is the set of atomic propositions true at timestep i. We say a trace w satisfies a formula ϕ at step i, written as $w, i \models \phi$ when:

- $w, i \models p$ iff w_i contains p;
- $w, i \models \neg\psi$ iff $w, i \not\models \psi$ (negation);
- $w, i \models \psi_1 \wedge \psi_2$ iff $w, i \models \psi_1$ and $w, i \models \psi_2$ (conjunction);
- $w, i \models \circ\psi$ iff $w, (i+1) \models \psi$ and $i < n$ (next);
- $w, i \models \psi_1 \mathcal{U} \psi_2$ iff there exists $0 \leq j \leq n$ such that $w, k \models \psi_1$ for all $i \leq k < j$, and $w, j \models \psi_2$ (until).

We say the trace w satisfies the formula ϕ, $w \models \phi$, iff $w, 0 \models \phi$. We further include additional operators as shorthands:

- $\psi_1 \vee \psi_2 \equiv \neg\psi_1 \wedge \neg\psi_2$ (disjunction);
- $\Diamond\psi \equiv true \mathcal{U} \psi$ (eventually ψ);
- $\Box\psi \equiv \neg\Diamond\neg\psi$ (always ψ).

Note that the atomic propositions Π only represent properties regarding the state of the world, and does not directly represent human or robot actions. This allows the user to focus on the task itself, rather than properties of the robot and human when writing the formula, and thus reducing the size of the formula.

D. Winning Strategy

Given an execution in the abstraction $(v_0, v_1, ..., v_n) \in V^*$, we can apply the ρ to each v_i, and determine which predicates are true along the execution. This produces a trace $(w_0, ..., w_n)$ where $w_i = \rho(v_i)$. If at any point n during the execution, the trace $(w_0, ..., w_n)$ satisfies the formula that represents the task, we say that the robot has achieved the task. This may happen while the human has remaining moves, as the robot is not obligated to wait for the human to use all moves available. Other ways of modeling task completion is discussed in Section IV-B.

During execution, the robot reasons over the events observed so far, and chooses an action to take. A strategy $Str : V^* \to A_s$ is a mapping that chooses a robot action to perform given the execution so far. Under the assumption that the human interferes at most K times, we say that Str is a winning strategy for the task if, following the strategy Str can guarantee the trace produced by the execution satisfies the task ϕ using resources not exceeding E.

E. Problem Definition

Given an abstraction graph $G = (V, v_0, A_s, A_e, F_G, \Pi, \rho)$, an LTLf formula ϕ over Π, a maximum amount of resources available E, and a bound on the number of human moves K, find a winning strategy Str for the task.

Note that since we wish to find a strategy that guarantees task completion under all possible allowed human behavior, we view the human as adversarial. Discussion about other semantics for human behavior exists in the literature, but it is not a focus of this paper.

IV. Solution

To solve the above problem, we first convert the LTLf task formula into a *deterministic finite automaton* (DFA). Then, we compose the DFA with the abstraction to construct a quantitative game. We show that a winning strategy on this game can produce a winning strategy for the synthesis problem. Thus, the synthesis problem is reduced to finding a winning strategy of the game. Finally, we use a winning set construction to produce a strategy for the game and therefore solve the synthesis problem.

A. Task DFA

From the LTLf formula ϕ over atomic propositions Π, a *deterministic finite automaton* (DFA) $A_\phi = (Z, z_0, \Sigma, \delta, Z_f)$ can be constructed to accept exactly the traces that satisfy the LTLf formula [13]. Here, Z is a finite set of states in the DFA, with z_0 as the start state, and $Z_f \subseteq Z$ as the set of final states. $\Sigma = 2^\Pi$ is the alphabet of the LTLf formula. $\delta : Z \times \Sigma \to Z$ is the deterministic transition function.

Given a trace $(w_0, ..., w_n)$, where $w_i \in 2^\Pi$, we can produce a sequence of DFA states $(z_0, ..., z_{n+1})$, where $\delta(z_i, w_i) = z_{i+1}$. If the last state z_{n+1} is a final state of the DFA, we say that the DFA *accepts* the trace. The DFA A_ϕ accepts exactly those traces that satisfy the formula ϕ. Thus the paths from the initial state z_0 to the final states Z_f in A_ϕ capture all the different ways the task could be achieved. This translation is PSPACE [13] in the size of the formula. Since the formula only reasons over important properties of the task, and not robot and human actions, the formula is typically small. We then combine the DFA with the abstraction to capture such physical constraints.

B. Quantitative Game

The abstraction graph G captures all the ways the robot and the human can interact with the environment. The DFA A_ϕ captures all the ways the task can be achieved. Additionally, we have a bound on the number of human actions K and the resource bound E. We define a structure P called the *game* that captures all these inputs in a unified manner. The game P contains all possible ways the robot can achieve the task and all possible ways the human can interfere.

The game structure is defined as $P = (S, s_0, S_f, A_s, A_e, F_P, E)$, similar to [13], [22]. The set of game states $S = V \times Z \times \{0, 1, ..., K\}$ represent the state of the world, the current state in execution, and the number of human moves remaining. The number of states of this game is linear in K, $|Z|$, and $|V|$. The initial state $s_0 = (v_0, z_0, K)$ is constructed from the initial state of the world v_0, the initial state of the DFA z_0, and the bound on human moves K. The set of final states $S_f \subseteq S$ consists of the states whose DFA component is a final state, $S_f = \{(v, z, k) | z \in Z_f\}$. Note that a state can be a final state even if $k \neq 0$, as discussed in Section III-D. If the task requires the robot to respond to human actions after task completion, we can modify the game by enforcing $k = 0$ for final states.

With an abuse of notation, A_s and A_e are extended from the abstraction to the game, by defining $a_s((v, z, k)) = (a_s(v), \delta(z, \rho(a_s(v)), k)$, and $a_e((v, z, k)) = (a_e(v), \delta(z, \rho(a_e(v)), k - 1)$. In other words, the result of applying an action on the game is determined as follows. The action produces the next abstraction state, and the set of atomic propositions at the new abstraction state causes the transition in the DFA. We maintain the number of human moves remaining, and no human action is allowed from a game state (v, z, k) where $k = 0$.

The resource cost function F_P is extended from F_G similarly, $F_P : S \times A_s \to \mathbb{R}_{\geq 0}$, by considering only the abstraction vertex component. E is the resource limit given by the user. The goal of the robot is to drive the game into a final state using no more than the resource limit, while the human is trying to prevent the game from reaching a final state.

During a play of the game, starting from s_0, the human chooses either an action a_e from A_e or no action. If the human chooses no action, the robot chooses an action a_s from A_s. The state evolves according to a_e if the human chooses the action a_e. Otherwise, the state evolves by the action chosen by the robot, $a_s(s)$. Each time the robot takes an action, the resource cost determined by F_P is accumulated. This process is repeated until a final state is reached or the accumulated resource cost exceeds the bound E.

C. Game Winning Strategy

We now define a game winning strategy, and how this winning strategy on the game can be used to find a strategy on the synthesis problem. A strategy on the game for the robot is a mapping $Str_P : S \to A_s$ that picks the robot action given the game state. For a strategy Str_P, if all possible plays where the robot follows Str_P reach a final state with accumulated cost no more than E, then we call Str_P a winning strategy.

D. Converting Str_P To Str

A strategy Str_P on the game P can be converted to a strategy Str for the synthesis problem. Recall that a strategy Str for the original problem needs to choose an action a_s given the execution so far $(v_0, ..., v_n) \in V^*$. We can do this by generating the trace so far using the predicate function ρ, and using the DFA to find the current DFA state z_n. We can also observe the abstraction G to find the number of moves the human has remaining k. Thus we can construct the game node $s = (v_n, z_n, k)$. The strategy Str can simply choose the action determined by $Str_P(s)$. If the strategy Str_P is winning on the game, then following it will guarantee the sequence of states produced in the DFA component is a trace that satisfies the LTLf formula. Therefore, Str generated from Str_P is a winning strategy for the synthesis problem.

E. Strategy Synthesis

We now provide an algorithm for finding a winning strategy on the game. We use a dynamic programming technique similar solving DFA games [13] and quantitative synthesis [22]. The approach is to start with the final states and grow the set of states where a win is guaranteed, while maintaining the action to take from these states. The algorithm is outlined in Algorithm 1. The variable `Map` is a hashmap that looks up the action to perform ($Map[s].action$) and the worst-case resource needed ($Map[s].cost$) from a game state s. First, we map all final game states to no action needed and no resource needed (line 3-5). Then, we expand the map until the

Algorithm 1 Game Strategy Synthesis

```
1: procedure FindStrategy($G$, $A_\phi$, $K$, $E$)
2:     $Map[s] \leftarrow (NULL, \infty)$ for all $s = (v, z, k)$
3:     for all $s = (v, z_f, k)$ where $z_f$ is a final state do
4:         $Map[s] = (NULL, 0)$
5:     end for
6:     $Converged \leftarrow$ False
7:     while not $Converged$ and $Map[s_0].cost > E$ do
8:         $Converged \leftarrow$ True
9:         for all $s = (v, z, k)$ do
10:            $C_E = max(Map[s'].cost), \quad \forall s' = a_e(s)$
11:            $C_S = min(F(v, a_s) + Map[a_s(s)].cost), \forall a_s$
12:            $Map[s].cost \leftarrow max(C_E, C_S)$
13:            $Map[s].action \leftarrow a_s$ that produced $C_S$
14:            if $Map[s']$ changed value then
15:                $Converged \leftarrow$ False
16:            end if
17:        end for
18:    end while
19:    return $Map$
20: end procedure
```

initial state is included. In each iteration, we check all game states s. The resource needed from s is updated to the worse scenario between the worst-case human action (line 10) and the best-case robot action (line 11). We record the best action for the robot to take (line 13). If any state has been updated, we mark the game as not yet converged. This computation terminates with success when the initial state is included, or terminates with failure if the algorithm has converged without setting the initial state to within the resource limit. In either case, we return Map, which stores the action to take and resource needed to win from each state.

To understand why the algorithm works, consider the following. If we have included a state s in the winning set needing E_{remain} to win, then no matter which action the human takes, the new state is still in the winning set needing at most E_{remain} to win. If the human does not take an action, the robot can take the action specified by the mapping Map. The next state is still in the winning set with resource need of at most $E_{remain} - F_G(s.v, Map[s].action)$. Thus, if the initial state is in the winning set with $Map[s].cost \leq E$, we can always reach the final state within the resource bound E.

The runtime of Algorithm 1 is $O(|S|^2(|A_s + A_e|))$. Each iteration of the while loop scans no more than each edge extending from each state. The total number of iterations is no more than the number of states of the game.

Note that this game is different from a traditional turn-based game. The robot and human actions are not alternating or concurrent, therefore we cannot apply classical algorithms such as minimax directly. However, this game can be converted to a turn-based game, by creating auxiliary human actions that represent the human taking any combinations of 0 to K actions in one turn. This incurs a blowup in the number of human actions to consider from $|A_e|$ to $(|A_e|+1)^{K+1}/|A_e|$, making this approach intractable.

V. Case Study

We demonstrate the framework with a proof-of-concept case study in manipulation planning. Consider again the cup-stacking example from the introduction. Initially, the scene is as shown in Figure 2a. To complete the stacking challenge, the cups must be stacked as shown in Figure 2d. We demonstrate in this section how the cup-stacking example can be naturally expressed using an abstraction and an LTLf formula. We then solve this problem using the approach described above. Finally, we discuss how the resulting strategy is applied to solve this example.

A. Abstraction

The abstraction captures a restricted set of the configuration space that is needed for the task. We call the cups cup_0, cup_1, and cup_2, as shown in Figure 2, and the locations of the pyramid top, $leftbase$, and $rightbase$. Recall that the abstraction is a tuple $G = (V, v_0, A_s, A_e, F_G, \Pi, \rho)$. We now show one way an input abstraction can represent this problem. In Section VI, we further discuss the properties this domain that makes it possible to find such an abstraction.

The vertices $v \in V$ of the abstraction graph are tuples $(L(cup_0), L(cup_1), L(cup_2), L(gripper))$, where $L(cup_i)$ is the location of cup_i, and $L(gripper)$ is the location of the gripper. The location of a cup can take the following values:

- top, $leftbase$, $rightbase$, indicating the object is at the corresponding location;
- $gripper$, indicating the object is being held by the robot;
- $elsewhere$, indicating the object is placed somewhere on the table that is not in the pyramid locations.

The location of the gripper can take the following values:

- top, $leftbase$, $rightbase$, indicating the gripper is aligned with the corresponding location (ready for drop);
- cup_i, indicating the gripper is aligned with the cup (ready to grab);
- $free$, indicating the arm is moving freely.

In this example, the initial vertex of the abstraction is $v_0 = (rightbase, elsewhere, elsewhere, free)$.

For the set of robot edges, we consider four types of robot actions: $Drop$ and $Grab$ indicate opening and closing of the gripper, while $Transfer$, and $Transit$ represent moving the arm with and without an object in the gripper. A_s contains many instances of each type of action parametrized by the cups and locations. Note that such edges are not limited to the ones above. If the robot is able to perform other types of actions such as pushing or pulling, these types of edges can be added to the abstraction, as long as such actions can be planned for prior to execution, or can be computed online.

A_e contains many instances of the $Move$ action which is parametrized by the cups and locations. Each $Move$ models an instantaneous transportation of a cup to a different location.

We use the following resource model F_G for the robot. The robot needs 5 units of resources for opening and closing grippers, 25 units of resources to move the empty gripper, and 75 and 50 units of resources to move a cup to the top and the base locations, respectively.

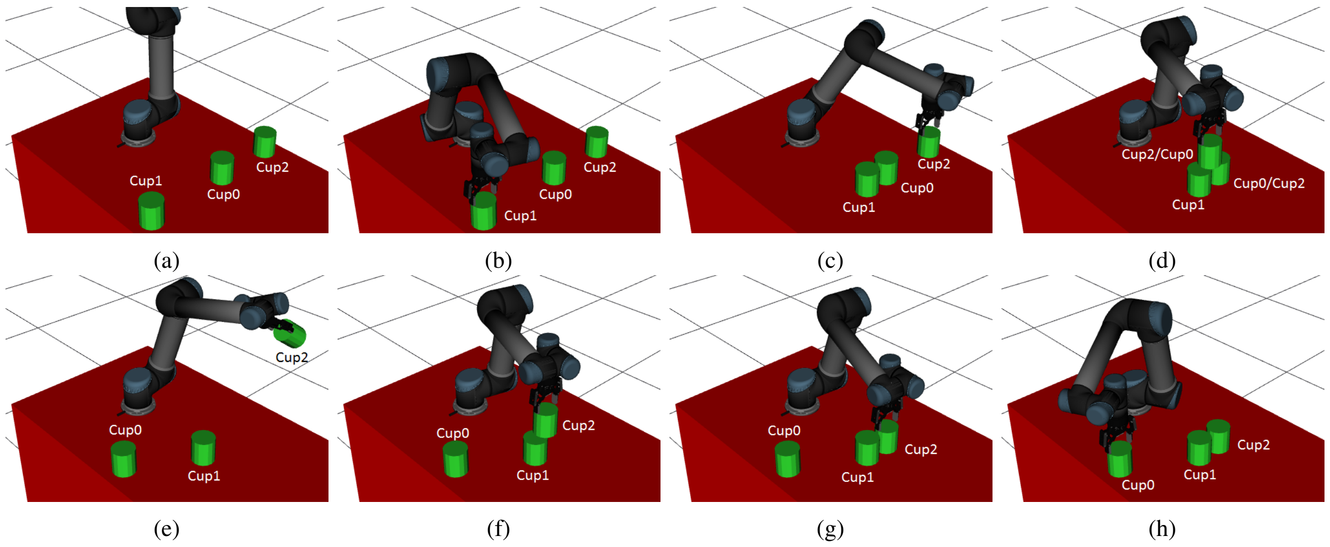

Fig. 2: Two executions of the cup-stacking policy. (a) shows the initial configuration. (d) shows the final configuration, where any object can be used for any of the locations. (a)(b)(c)(d): Policy executed without human interference. The robot first reaches for cup_1 (b) and moves cup_1 to one of the base locations (c), then moves cup_2 to the top location (d, label left of "/"). (a)(b)(c)(e)(f)(g)(h)(d): Policy execution with one human interference. When the robot is moving cup_2 to the top, the human moves cup_0 away from the base (e). When the robot moves cup_2 to the top, it recognizes that cup_2 cannot be placed on the top, due to the constraint given by the task (f). Instead, cup_2 is directly moved to the base (g), then cup_0 is retrieved for the top (h). Finally the cups are stacked (d, label right of "/"), but with cup_0 at the top location.

The atomic propositions Π determine whether or not each cup cup_i is at each location $l \in \{top, leftbase, rightbase\}$. $\rho(v)$ can be directly read v, by examining if $L(cup_i) = l$.

B. LTLf Specification

We wish to specify that the final configuration is reached, and the safety constraint of not placing the top cup unless both of the base cup are in place must be observed. In LTLf, this task is written as

$$\begin{aligned}\phi \quad = \quad & \Diamond\big(\bigwedge_{j=0,1,2}\bigvee_{i=0,1,2} P_{i,j}\big) \quad \wedge \\ & \Box\Big(\big(\bigwedge_{i=0,1,2} \neg P_{i,0}\big) \vee \big(\bigvee_{i=0,1,2} P_{i,1} \quad \wedge \quad \bigvee_{i=0,1,2} P_{i,2}\big)\Big)\end{aligned}$$

where $P_{i,j}$ denotes $L(cup_i) = l_j$, $l_0 = top$, $l_1 = leftbase$, and $l_2 = rightbase$. The formula can be read as "a cup is at each of the locations top, $leftbase$, and $rightbase$, and if a cup is at top, there must be cups at $leftbase$ and $rightbase$."

C. Strategy Application

Once the abstraction G and the LTLf formula ϕ are constructed, we use G, ϕ, $E = 3000$, $K = 5$ to construct the game and solve for a winning strategy. We then use the winning strategy for the game to instruct the robot to achieve the task. The robot initializes from the initial state of the game, and maintains the DFA A_ϕ as well as resource usage and the number of human actions remaining. Each time the strategy receives the current state of the world, the robot updates the state in A_ϕ and the the number of human actions remaining. This information is used to look up the action to take from the game strategy Str_P. The motion corresponding to this action is then found and applied.

D. Implementation

The translation from an LTLf formula to a DFA is performed using Spot [24]. Our reactive strategy synthesis algorithm is implemented in C++. Once we have constructed the strategy, we apply it to a visualized system using the method described in Section V-C.

The planning scene is constructed using the MoveIt! [25] package in ROS, and we also use MoveIt! for scene monitoring and trajectory execution. For generating motion plans, RRTConnect [26] in the OMPL library [27] is used through the MoveIt! interface. The resulting execution is visualized in RViz. Communication between the strategy and these modules is implemented using the ROS API.

E. Results

Our framework successfully found a strategy to solve the cup-stacking problem. Random valid human moves were applied at random intervals and the robot successfully achieved the task in all cases. Figure 2 shows two simple runs of the resulting strategy where the authors controlled the human actions. In the top row, the human did not interfere, and the robot successfully achieved the task using 400 units of resources. In the bottom row, the robot reacts to human interference by placing the cup originally intended for the top at the base. This end configuration is different from what the robot achieved without human intervention, but is nonetheless correct. In this execution, the robot needed 775 units of resources. In both cases, the human performed far fewer moves than is allowed, thus the robot used very little resources compared to the bound given.

In this example, an average of 21.97s (over 10 runs) was taken to synthesize the entire high-level strategy offline. At runtime, the strategy is used as a lookup table. Nonetheless, the algorithm scales exponentially with the number of objects considered. Finding techniques to speed up computation is a key point in future work. Motion planning is performed online and not included in planning time.

If instead of having a strict resource bound, we only ask the robot to execute to minimize worst-case resource usage, we can still find a strategy by providing 0 as the resource bound in the input. This will provide a correct strategy because Algorithm 1 will fail, but failure only happens at the fixed point of computation, where the resource consumption is optimally small. Therefore, the resulting strategy stored in *Map* selects the action that minimizes the worst-case resource cost. Such a strategy is often desirable if we do not have a hard constraint on resources but wish to use as little energy as possible. However, running the winning set construction algorithm until convergence could take more time than stopping when the initial state is found with satisfactory worst case resource needed to win. In the cup-stacking example, an average of 26.81s (over 10 runs) is needed to generate such a strategy.

VI. Conclusion and Discussion

In this paper, we take a step toward solving the reactive strategy synthesis problem for finite tasks with resource constraints. We introduce a framework that assumes as input an abstraction and solves the problem by reduction to a game. The game is in turn solved using winning set construction. We show that this framework can be used for robotics applications through a proof-of-concept example of a robot performing a cup-stacking task. We plan to demonstrate this framework on a physical UR5 robot in the future. We are also working on addressing the runtime performance of our algorithm.

In this paper, we make the assumption that the domain can be captured by an abstraction. The abstraction requires that all continuous instantiations the is needed by the task is given. In the cup-stacking example, this means that the robot never needs to place cups at locations other than places on the pyramid. How to generate the abstraction from the planning scene and determining if this generation is possible is a topic open for investigation. The abstraction also requires motion plans for each action we want to take. We also assumed perfect sensing in this paper. In practice, correctly sensing the state of execution is important for ensuring execution is carried out correctly. Considerations for motion planning and sensing are key areas of future work.

References

[1] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *International Conference on Robotics and Automation*. IEEE, 2011.

[2] 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 *International Conference on Robotics and Automation*. IEEE, 2014.

[3] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Towards manipulation planning with temporal logic specifications," in *International Conference on Robotics and Automation*. IEEE, May 2015.

[4] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: a constraint-based approach," in *Robotics: Science and Systems*, 2016.

[5] 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.

[6] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *International Conference on Hybrid Systems: Computation and Control*, 2010.

[7] A. Ulusoy, M. Marrazzo, and C. Belta, "Receding horizon control in dynamic environments from temporal logic specifications." in *Robotics: Science and Systems*, 2013.

[8] Y. Wang, N. T. Dantam, S. Chaudhuri, and L. E. Kavraki, "Task and motion policy synthesis as liveness games." in *International Conference on Automated Planning and Scheduling*, 2016.

[9] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *International Conference on Hybrid Systems: Computation and Control*. ACM, 2015.

[10] F. Penedo, C.-I. Vasile, and C. Belta, "Language-guided sampling-based planning using temporal relaxation," in *International Workshop on the Algorithmic Foundations of Robotics*, 2016.

[11] P. Labone and M. Ghallab, "Planning with sharable resource constraints," in *International Joint Conference on Artificial Intelligence*, 1995.

[12] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 5–33, 2001.

[13] G. De Giacomo and M. Y. Vardi, "Synthesis for LTL and LDL on finite traces." in *International Joint Conference on Artificial Intelligence*, 2015.

[14] M. Gharbi, R. Lallement, and R. Alami, "Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search." in *International Conference on Intelligent Robots and Systems*. IEEE, 2015.

[15] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation*. Charleston, SC: Springer, 2006, pp. 364–380.

[16] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki, and M. Y. Vardi, "This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction," in *Conference on Artificial Intelligence*. AAAI, 2015.

[17] 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, May 2016.

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

[19] M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "A sampling-based strategy planner for nondeterministic hybrid systems," in *IEEE Conference on Robotics and Automation*, 2014.

[20] R. Ehlers, S. Lafortune, S. Tripakis, and M. Vardi, "Bridging the gap between supervisory control and reactive synthesis: Case of full observation and centralized control," *IFAC Proceedings Volumes*, vol. 47, no. 2, pp. 222–227, 2014.

[21] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[22] A. Chakrabarti, K. Chatterjee, T. A. Henzinger, O. Kupferman, and R. Majumdar, "Verifying quantitative properties using bound functions," in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, 2005.

[23] R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann, "Better quality in synthesis through quantitative objectives," in *International Conference on Computer Aided Verification*, 2009.

[24] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 — a framework for LTL and ω-automata manipulation," in *International Symposium on Automated Technology for Verification and Analysis*, 2016.

[25] S. Chitta, I. Sucan, and S. Cousins, "Moveit!" *IEEE Robotics Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[26] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *International Conference on Robotics and Automation*. IEEE, 2000.

[27] 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.