Finite-Horizon Synthesis for Probabilistic Manipulation Domains

Andrew M. Wells, Zachary Kingston, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi

***Abstract*— Robots have begun operating and collaborating with humans in industrial and social settings. This collaboration introduces challenges: the robot must plan while taking the human's actions into account. In prior work, the problem was posed as a 2-player deterministic game, with a limited number of human moves. The limit on human moves is unintuitive, and in many settings determinism is undesirable. In this paper, we present a novel planning method for collaborative human-robot manipulation tasks via probabilistic synthesis. We introduce a probabilistic manipulation domain that captures the interaction by allowing for both robot and human actions with states that represent the configurations of the objects in the workspace. The task is specified using Linear Temporal Logic over finite traces (LTL$_f$). We then transform our manipulation domain into a Markov Decision Process (MDP) and synthesize an optimal policy to satisfy the specification on this MDP. We present two novel contributions: a formalization of probabilistic manipulation domains allowing us to apply existing techniques and a comparison of different encodings of these domains. Our framework is validated on a physical UR5 robot.**

I. Introduction

In the near future, robots and humans will collaborate in complex environments to achieve shared tasks in the factory or at home. This introduces new challenges in planning. When humans are present, the robot must take them into account, not only for safety, but also because humans can intervene, changing the state of the world. Because these interventions could result in an undesired state, traditional planning methods that output a fixed sequence of actions are not sufficient to achieve the task. This paper focuses on the challenge of modeling and planning for robotic manipulators in such scenarios, where a high-level task is given and performance guarantees are required.

In such scenarios, a common approach is to search for a *strategy* or *policy* rather than a sequential plan. One class of approaches is based on reactive synthesis [1]–[3]. These algorithms originate from automatic verification [4], and have recently been applied in robotics, e.g., [5]–[8]. Here, the task is specified by a *linear temporal logic* (LTL) [9] formula and the robot model is abstracted to a finite discrete structure called a *domain abstraction*. Existing methods view human interference as non-deterministic environment actions. To find a policy, these algorithms construct a game between the robot and the environment and compute a *winning strategy* for the robot–a strategy that guarantees completion of the task by choosing the best action for the robot in reaction to a human (environment) action– if one exists. This approach, however, leads to conservative policies for the robot, and if the environment is not limited in its choices, a winning strategy may not exist.

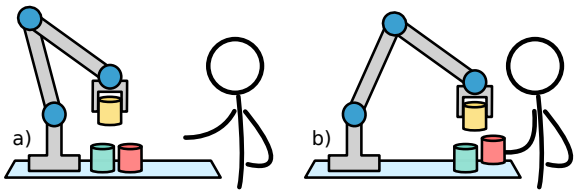


Fig. 1: *Reactive vs Probabilistic Synthesis:* Unlike prior works in synthesis for human-robot collaboration, we consider probabilistic models of interaction. In reactive synthesis **(a)** the robot is conservative, shown as the robot building the arch away from the human to avoid intervention. In probabilistic synthesis we can model a cooperative human, allowing the robot to collaboratively build the arch near the human **(b)**.

Consider a robot with a construction task and a collaborating human as shown in Fig. 1. The robot, whose plan is computed with reactive synthesis (Fig. 1a), assumes adversarial moves by the human. If the human is not adversarial, this pessimistic assumption may lead to inefficient collaboration. In contrast probabilistic synthesis allows the robot to rely on the human, taking actions that anticipate human aid. Furthermore, for the robot to succeed in the face of an adversarial human, reactive synthesis needs to assume an upper bound on the number of human actions [10].

In prior work [10]–[12], we introduced a cost-based reactive synthesis framework for robotic manipulation. The focus in these works is on tasks that can be accomplished in finite time, given as LTL *over finite traces* (LTL$_f$) [13] formulas. A symbolic approach [12] is used to mitigate the *state explosion* of an increasing number of objects in a manipulation domain. An upper bound on the number of human actions was, however, required to compute a winning strategy. Moreover, these methods could not incorporate probabilistic effects that we wish to model (e.g., mechanical failure of the robot).

Other works have included probabilistic effects in human-robot collaboration using *Markov Decision Processes* (MDPs) [14]–[16]. These works, however, focus on mobile robots, where discretizations are relatively simple (e.g., imposing a grid on the floor). Our work focuses on manipulation with arbitrary object placements, and thus has a large state space where constructing discrete abstractions is difficult.

In this work, we move from reactive synthesis to probabilistic synthesis in a fully-observable manipulation domain. This shifts the focus from the worst case to the expected case.

Authors AMW, ZK, LEK and MYV are with the CS department at Rice University. Author ML is with the Aerospace Engineering Sciences and Computer Science departments at the University of Colorado at Boulder.

Work on this project by LEK and MYV has been supported in part by NSF 1830549. Work on this project by AMW and ZK has been supported by NASA 80NSSC17K0162 and 80NSSC17K0163.

That is, instead of focusing on a winning strategy that can counter the worst human actions, we look for a policy that optimizes for the best expected performance. In this approach, the human is viewed as a probabilistic agent to enable more efficient collaboration (Fig. 1b). We assume the domain can be represented as a turn-based interaction (see Def. III-A).

This leads to a probabilistic manipulation domain that has received little attention in the literature. There are two main challenges in this problem domain: (i) representing the probabilistic manipulation domain in the form of an MDP is non-trivial; (ii) the state-explosion problem is inherited from both manipulation domain and LTL synthesis, leading to computational tractability issues.

The contributions of this work are threefold. First, we introduce a formalization of probabilistic manipulation domains that allows us to apply existing techniques. Specifically, we show how a human-robot collaborative manipulation domain can be modeled as an MDP and synthesize policies to maximize the probability of satisfying an LTL_f formula. Second, we remove the assumed limit on human actions required in previous works [10]–[12] and allow an arbitrary finite number of actions for the human. Third, we empirically compare different modeling techniques and provide a series of complex pick-and-place human-robot experiments on a UR5 robot to illustrate the power and scalability of the synthesis framework.

II. Preliminaries

We are interested in manipulation domains with a high-degree-of-freedom robot manipulator and multiple objects. This problem is inherently continuous, but we can simplify the problem considerably, while capturing its essential features by considering discrete abstractions [11]. In this section, we first define such an abstraction based on prior work. Then we present Linear Temporal Logic over finite traces (LTL_f) and Markov Decision Processes (MDPs), which we use to formally model our problem in the next section.

A. Discrete Abstraction of Manipulation Domain

The most common tool for defining discrete planning domains is the Planning Domain Definition Language (PDDL) [17]. We can use PDDL to define states and actions, along with preconditions and effects of these actions. The description essentially encodes what actions are possible in order to prevent the human or robot from moving objects in ways that are not physically feasible (e.g., removing a support out from under an object to reach an illegal state with a floating object). Many existing techniques can reason about domains described in this way [18]. We augment standard PDDL by allowing a partition of the set of actions into human actions and robot actions.

A consideration of manipulation domains must respect the physical constraints described by PDDL. Underlying the PDDL definition of the domain, there is some geometric domain consisting of the robot and the robot's workspace. Prior work [11] showed an automated procedure for constructing a discrete manipulation abstraction from this manipulation domain. We build on this abstraction in our work.

Definition 1 (Abstraction of a Manipulation Domain)**.** *An* abstract manipulation domain *is a tuple:* $G_{det} = (S, s_0, A_s, A_e, T, AP, L)$ *where,*

- S *is a finite set of states,*
- $s_0 \in S$ *is the initial state,*
- A_s *is the finite set of system (robot) actions,*
- A_e *is the finite set of environment (human) actions,*
- $T : S \times (A_s \cup A_e) \mapsto S$ *is the transition function,*
- AP *is the set of task-related atomic propositions,*
- $L : S \mapsto 2^{AP}$ *is the labeling function that maps each state to a set of atomic propositions.*

Manipulation domain G_{det} extends the classical PDDL by incorporating the set of environment (human) actions A_e as well as the system (robot) actions A_s.

Each action transitions to some reachable state according to PDDL semantics. Because we focus on interaction rather than interference (e.g., the human's hand blocking the robot's path), we follow [10]–[12] and discretize the manipulation domain by considering each change in the discrete state as the beginning of a new time step.

Example 1. *As a concrete example, consider the robot in Fig. 2 whose goal is to build an arch out of three blocks. There are five locations of interest, including the Robot end-effector (Loc_0) and the Else region (Loc_1), which contains all regions not otherwise specified (c.f., blue block in Fig. 2).*

Fig. 2: Manipulation example: (left) the locations of interest, where the Else location (Loc_1) contains all objects not otherwise shown. (right) Initial state with red and yellow blocks at Loc_2 and Loc_3 and the blue block at Loc_1.

Locations may have a maximum capacity. The Robot end-effector (Loc_0) has a limit of one and $Else$ has no limit on the number of objects placed therein.

The set A_s consists of robot pick and place actions in standard PDDL. These actions have preconditions (e.g., you cannot pick a block from Loc_3 if Loc_4 is occupied). A_e consists of actions that transition to reachable states according to PDDL semantics. From state s with $act \in (A_s \cup A_e)$, transition function T gives the result s_{i+1}. We assume actions in $(A_s \cup A_e)$ are atomic, since our focus here is on interaction between robot and human and not on interference. Fig. 3 shows a portion of the transition system for a deterministic manipulation domain. Here, the double-edged state indicates a goal (accepting) state (see Example 2).

B. Linear Temporal Logic over finite traces

While regular languages are typically specified using tools such as regular expressions, many tasks can be expressed more naturally in a temporal logic [12]. Linear temporal

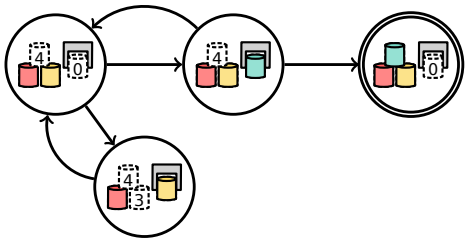


Fig. 3: Example portion of abstraction of manipulation domain G_{det} for a deterministic pick-and-place domain.

logic (LTL) is a popular formalism used to specify temporal properties. Here, we are interested in LTL interpreted over finite traces (LTL_f) [13].

Definition 2 (LTL_f Syntax)**.** *An LTL_f formula is built from a set of atomic propositions AP and is closed under the Boolean connectives as well as the "next" operator X and the "until" operator U:*

$$\phi ::= \top \mid p \mid (\neg\phi) \mid (\phi_1 \wedge \phi_2) \mid (X\phi) \mid (\phi_1 U \phi_2)$$

where $p \in AP$, $\top$ is "true" or a tautology, and $\neg$ and $\wedge$ are the "negation" and "and" operators in Boolean logic respectively.

The common temporal operators "eventually" (F) and "globally" (G) are defined as: $F\,\phi = \top\, U\, \phi$ and $G\,\phi = \neg F\, \neg\phi$.

The semantics of LTL_f are defined as follows.

Definition 3 (LTL_f Semantics)**.** *The semantics of an LTL_f formula are defined over finite traces. A trace ρ is a word in $(2^{AP})^*$. Let $|\rho|$ denote the length of trace ρ and $\rho[i]$ be the i^{th} symbol of ρ. Further, $\rho, i \models \phi$ is read as: "the i^{th} step of trace ρ is a model of ϕ.":*

- $\rho, i \models \top$;
- $\rho, i \models p$ *iff* $p \in \rho[i]$;
- $\rho, i \models \neg\phi$ *iff* $\rho, i \not\models \phi$;
- $\rho, i \models \phi_1 \wedge \phi_2$, *iff,* $\rho, i \models \phi_1$ *and* $\rho, i \models \phi_2$;
- $\rho, i \models X\phi$ *iff* $|\rho| > i+1$ *and* $\rho, i+1 \models \phi$;
- $\rho, i \models \phi_1 U \phi_2$ *iff* $\exists j$ *s.t.* $i \leq j < |\rho|$ *and* $\rho, j \models \phi_2$ *and* $\forall k,\ i \leq k < j, \quad \rho, k \models \phi_1$,

Finite trace ρ satisfies ϕ, denoted by $\rho \models \phi$, if $\rho, 0 \models \phi$. An LTL_f formula ϕ defines a language $\mathcal{L}(\phi)$ over the alphabet 2^{AP}. $\mathcal{L}(\phi)$ is a regular language, more specifically, $\mathcal{L}(\phi) = \{\rho \in (2^{AP})^* \mid \rho \models \phi\}$.

Example 2 (LTL_f Example)**.** *Continuing Example 1, suppose our task is: "Eventually complete an arch and Globally, when the top of the arch is placed, the supports must also be placed." We use loc_i as shorthand for $(blueblock\_at\_loc_i \vee yellowblock\_at\_loc_i \vee redblock\_at\_loc_i)$. Then our task is:$(F(loc_4) \wedge G(loc_4 \rightarrow (loc_3 \wedge loc_2)))$ A satisfying path from s_0 in Example 1 would be: $(\langle s_0, pick(blue, Else), no\_op\rangle, \langle s_1, place(blue, Loc_4), no\_op\rangle)$. This path is only satisfying if both actions succeed.*

C. Markov Decision Processes

Markov Decision Processes (MDPs) are commonly used to model stochastic systems with action choices. As such, they are a good fit for modeling our problem. In Sec. III, we introduce a probabilistic manipulation domain and show how to model it as an MDP. Then we apply existing tools to solve an LTL_f synthesis problem on this MDP.

Definition 4 (MDP)**.** *A labeled* Markov Decision Process *(MDP) is a tuple: $\mathcal{M} = (S, A, P, s_{init}, AP, L)$, where:*

- *S is a finite set of states;*
- *A is a finite set of actions, and $A(s) \subseteq A$ denotes the set of actions enabled at state $s \in S$;*
- *$P : S \times A \times S \rightarrow [0, 1]$ is the transition probability function where $\sum_{s' \in S} P(s, a, s') = 1$ for all $s \in S$ and $a \in A(s)$.*
- *$s_{init} \in S$ is the initial state;*
- *AP is the set of atomic propositions;*
- *$L : S \rightarrow 2^{AP}$ is the labeling function.*

Now, following [19], we define finite satisfaction (of an LTL_f formula) as follows.

Definition 5 (Path satisfying LTL_f)**.** *Given an MDP $\mathcal{M}$, and an LTL_f formula ϕ where the alphabet of the labeling function of $\mathcal{M}$ matches the propositions of ϕ, we say that a (possibly finite) path w through $\mathcal{M}$ satisfies specification ϕ if some prefix of w is in the language of ϕ, i.e.,*

$$L(w) \models \phi \quad \Leftrightarrow \quad \exists w' \in pre(w)\ s.t.\ L(w') \in \mathcal{L}(\phi),$$

where $L(w) \in (2^{AP})^*$ is the word obtained by applying the labeling function to the states of w and $pre(w)$ is the set of prefixes of w.

Let $paths_{fin}$ denote the set of finite paths of $\mathcal{M}$. Then a policy is defined as:

Definition 6 (Policy)**.** *A policy $\pi : paths_{fin} \mapsto A$ is a function that given a finite path through $\mathcal{M}$, assigns an action to the current state. $\mathcal{M}^\pi$ denotes $\mathcal{M}$ under policy π.*

A policy π induces a Markov chain over the paths in $\mathcal{M}$, with a well-defined probability measure [20]. Let $\mathcal{M}^\pi$ denote the Markov chain induced by π. Following Def. 5, we write $Pr(\mathcal{M}^\pi \models \phi)$ for the probability that a path satisfies ϕ.

Informally, we want to compute the best actions for the robot to take in any state to maximize the probability of satisfying the goal. Based on the preliminaries presented here, in Sec. III we introduce a probabilistic manipulation domain and formally define $\text{MDP}_{\text{manip}}$ to model this problem. We then solve the problem using existing tools.

III. Problem Modeling

We begin by extending Def. 1 to consider the probabilistic abstraction of a manipulation domain. We simplify the problem considerably, while capturing its essential features by considering discrete, turn-based, abstractions [11].

A. Probabilistic Abstraction of Manipulation Domain

We extend our manipulation domain abstraction G_{det} by introducing probabilities to capture stochastic events, such as failed robot execution or to capture human behaviors.

We introduce two probability distributions. $P_s(s_{i+1} \mid act_s, s_i)$ is a probability distribution giving the probability of transitioning from state s_i to state s_{i+1} on robot action $act_s \in A_s$, assuming no human action is taken. This corresponds to a probability distribution that describes the likelihood of different possible outcomes when the robot attempts action act_s in state s_i. This can be used to model, e.g., robot actions that might fail to execute properly or mechanical imprecision (the trembling-hand phenomenon).

We consider another probability distribution, P_e, to model human behaviors. Modeling humans probabilistically is challenging and is not the focus of our work. Nevertheless, we need some model for our experiments. We assume we are given a human model that gives $P_e(act_e \mid s_i)$, the probability that the human performs an action act_e at some state s_i. Each human action transitions to some reachable state s_{i+1}.

We now define the probabilistic abstraction of a manipulation domain, adding probability distributions P_s, P_e to G_{det}.

Definition 7 (Probabilistic Abstraction of a Manipulation Domain)**.** *The probabilistic abstraction of a manipulation domain is a tuple:* $G_{prob} = (S, s_0, A_s, A_e, T_e, AP, L, P_s, P_e)$ *where* S, s_0, A_s, A_e, AP *and* L *are defined as in Def. 1 and*

- P_s *is a probability function* $P_s : S \times A_s \times S \to [0, 1]$ *that gives the probability distribution of effects of system (robot) actions as the probability* $P_s(s_{i+1} \mid act_s, s_i)$*,*
- P_e *is a probability function* $P_e : S \times A_e \to [0, 1]$ *that gives the probability of an environment (human) action being applied at the current state* $P_e(act_e \in A_e \mid s_i)$*.*
- $T_e : S \times A_e \times S \to \{0, 1\}$ *gives the result of applying human action* $act_e \in A_e$ *at state* $s \in S$*.*

We assume that human actions always succeed (i.e., they are deterministic). It is rather the choice of the human action that is probabilistic. Once a human action act_e is chosen, the transition function $T_e(s_i, act_e, s_{i+1}) = 1$ if s_{i+1} is the result of applying act_e at s and 0 otherwise. Additionally, we assume the human moves faster than the robot. Because the robot should react to new information as soon as it is available, this means the human can "interrupt" the robot. This allows us to model human-robot interaction as a sequential, turn-based game. More complex models that consider arbitrary overlaps of concurrent human and robot actions and e.g., Timed Game Automata [21] are left to future work.

B. Combining Human and Robot Transitions

To build $\text{MDP}_{\text{manip}}$ from G_{prob}, we need to combine P_s, P_e and T_e into a single distribution. The resulting MDP is $\text{MDP}_{\text{manip}} = (S, A, P, s_{\text{init}}, AP, L)$, where S, s_{init}, AP, and L are as in Def. 1, $A = A_s$, and P is the combined probability distribution based on P_s, P_e and T_e. Consider Example 3:

Example 3 (Probabilistic Manipulation MDP I)**.** *Let us return to Example 1.* P_s *and* P_e *are described in Fig. 4 and Fig. 5. First consider a simplified MDP with only the robot actions:*

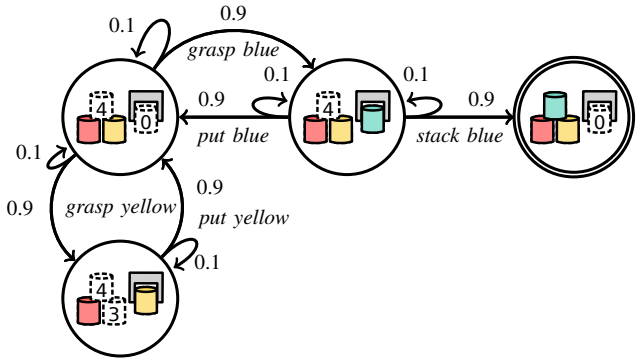


Fig. 4: Partial MDP for pick-and-place domain with no human actions. Action names are shown in italics.

Next consider a simplified MDP with only the human actions. Note the human cannot alter the robot's end-effector:

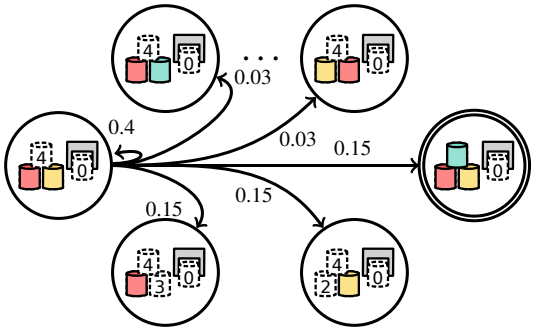


Fig. 5: Example portion of the MDP with only human actions. In the top row, the human has moved two blocks. In the remaining states, the human has moved one or no blocks.

The challenge comes in combining P_s and P_e. We assume the robot halts execution during human actions, except during the visual servoing stage of grasping/ungrasping, which we treat as atomic, since our focus here is on interaction. In $\text{MDP}_{\text{manip}}$, the robot actions are strategic, while the human actions are probabilistic. Thus, we define the transition probability P of $\text{MDP}_{\text{manip}}$ as follows:

Let $I(act_e) = 1$ if $act_e = no\_op$ and 0 otherwise. Then:

$$P(s_i, act_s, s_{i+1}) = \sum_{act_e \in A_e} P_e(s_i, act_e) \cdot \big(I(act_e) P_s(s_i, act_s, s_{i+1}) + T_e(s_i, act_e, s_{i+1})\big) \quad (1)$$

The reasoning behind this follows [12]. We would like the robot to incorporate new information as soon as possible. This means that whenever the robot observes that the state has changed, the robot should halt its current execution and re-query the optimal policy. On the other hand, if the human has not changed the state, then the robot will continue its current execution. Below, we give an example of our combined MDP.

Example 4 (Probabilistic Manipulation MDP II)**.** *Here we complete the example started above Example 3. The combined MDP has probability distribution* P *defined according to Equation 1. To simplify the figure, assume that the robot action chosen is "pick_blue". See Fig. 6.*

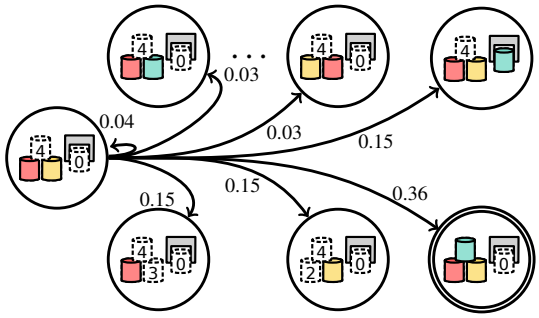


Fig. 6: Example portion of the MDP for our pick-and-place domain with both human and robot actions. The robot action is "pick_blue." There is a 36% chance that this succeeds, a 4% chance that it is attempted without interruption but fails. The remaining states are the results of human actions.

By transforming our probabilistic manipulation domain to an $\text{MDP}_{\text{manip}}$, we reduce the problem to a probabilistic synthesis problem that can be solved with existing tools [22]. Note that this problem is not only theoretically complex (it is 2-EXPTIME complete [23]), but also practically complex as planning in a high-dimensional manipulation domain is itself PSPACE-complete [24] and the state space grows exponentially in the number of objects. In this paper, we are concerned with the practical complexity of probabilistic manipulation domains, where building a tractable model is extremely challenging. In the next section, we present techniques for constructing a tractable $\text{MDP}_{\text{manip}}$ in Sec. IV-A and compare their performance in Sec. V.

IV. Solving the Problem

Once we have constructed $\text{MDP}_{\text{manip}}$ from G_{prob}, we use [22] to synthesize an optimal policy using a Deterministic Finite Automaton (DFA).

1) LTL_f to DFA: In [19], Zhu et al. showed that a symbolic approach to DFA construction based on this direct translation to Monadic Second-order Logic worked well for LTL_f synthesis. In [12], we showed a similar approach applied to reactive synthesis for robotics.

2) MDP $\times$ DFA Product: Once we have obtained the DFA corresponding to our LTL_f formula ϕ, we take the product with the MDP $\mathcal{M}$ in the following way:

The product of MDP $\mathcal{M} = (S, A, P, s_{\text{init}}, AP, L)$ and DFA $\mathcal{A}_\phi = (Q, \Sigma, q_0, \delta, F)$ is an MDP

$$\mathcal{M} \times \mathcal{A}_\phi = (S \times Q, A, P^{\mathcal{M}\times\mathcal{A}_\phi}, (s_{\text{init}}, q_{\text{init}})),$$

where $q_{\text{init}} = \delta(q_0, L(s_{\text{init}}))$, and

$$P^{\mathcal{M}\times\mathcal{A}_\phi}((s,q),a,(s',q')) = \begin{cases} P(s,a,s') & \text{if } q' = \delta(q, L(s)) \\ 0 & \text{otherwise} \end{cases}$$

Using this MDP, we now compute an optimal policy using existing tools. Specifically, we use PRISM [25] with an specialized LTL_f pipeline [22] to improve scalability.

A. Encoding a Scalable MDP_{manip}

Existing tools such as PRISM [25] that solve the LTL_f synthesis problem on $\text{MDP}_{\text{manip}}$ rely on symbolic computations using binary decision diagrams (BDD's) and arithmetic decision diagrams (ADD's) to improve scalability, which means our specific encoding of the model impacts the scalability. We now discuss the details of our method for building a tractable $\text{MDP}_{\text{manip}}$. We compare the default modeling of [22] as well as two models we developed for robot manipulation.

The first question we face in encoding the problem is what elements to include in $\text{MDP}_{\text{manip}}$ and what elements to include in the LTL_f specification. Note that while the specification in Example 2 allows $redblock\_at\_loc_4 \wedge redblock\_at\_loc_3$, we can prevent this by adding LTL_f clauses or by using the PDDL specification to construct a model with only valid transitions, ensuring no block is in two places at once. Because, [12] showed that limiting the model rather than the specification leads to better scalability, we follow the latter approach.

Previous works have shown that using symbolic structures for probabilistic synthesis frequently leads to more scalable tools, both in terms of runtime and memory usage. We use a mature tool for MDP synthesis [25], where numerous heuristics are used to make computation more efficient. Specifically relevant to our work, PRISM utilizes symbolic computations using BDDs and ADDs [26], [27]. BDDs are sensitive to variable ordering, but finding an optimal variable ordering is NP-hard. Thus, PRISM heuristically guesses an effective variable ordering. Specifically, variable ordering is based on the order of modules declared within PRISM and on the order of variables declared within modules. This motivates our choice of models.

In the *integer* encoding, states of $\text{MDP}_{\text{manip}}$ are enumerated by breadth-first search, defining a mapping from S to $\mathbb{N}$, which we use to model the problem within PRISM.

In the *object* encoding, we use a tuple mapping each object to its location. We represent our states as tuples of object locations (i.e., $(1,2,0)$ shows that $object_a$ is at Loc_1, $object_b$ is at Loc_2 and the human is at $humanLocation_0$). Because BDDs (and ADDs) are sensitive to variable ordering, we attempt to represent our states in such a way as to keep elements of the tuple that are essential to general problem (e.g., whether the robot is grasping an object) well positioned in the BDD. Specifically, we always use 0 as the end-effector location as this is the most important location. We use 1 as the else region, as this is the most common. The choice of other locations is not significant.

In the *location* encoding, we use a tuple mapping each location to the number of objects therein. (i.e., $(0,1,1,0)$, from the previous example, indicates that one block is in Loc_1 and one block is in Loc_2. The human location is again represented by the final number). Here, we optimize by observing that the number of blocks remains constant. This allows us to omit the counter for the "else" region (which is the most common, but least task-relevant), without affecting the state, as it can be inferred from the remaining variables. In Sec. V, we compare these three encodings.

V. Empirical Evaluation

We implement our MDP generation in Python and test on scenarios inspired by [12] as well as on other domains. The

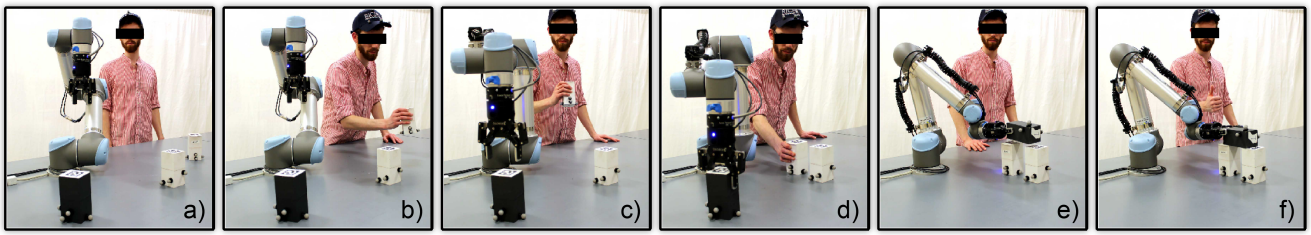


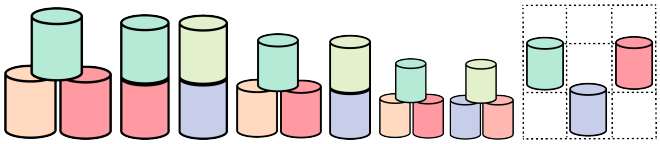
Fig. 7: At state (a), we anticipate human cooperation (b-d) leading to faster expected task completion (e-f).

	3 blocks	4 blocks	5 blocks	6 blocks	3 temporal	3 diagonal	5 compound
Time Integer	0.7s	121.89s	TO	TO	513.19s	316.97s	TO
Time Object	0.44s	1.38s	14.56s	520.64s	191.6s	**7.83s**	20.61s
Time Location	**0.42s**	**1.09s**	**7.81s**	**227.72s**	**154.66s**	9.88s	**14.7s**
Size Integer	**8MB + 19MB**	**64MB + 115MB**	TO	TO	1GB + 1.67GB	1GB + .39GB	TO
Size Object	16MB + 19MB	128MB + 95MB	**2GB + .47GB**	**24GB + 1.09GB**	**1GB + .98GB**	**1GB + .37GB**	3GB + .84GB
Size Location	16MB + 18MB	256MB + 48MB	2GB + .33GB	28GB + .81GB	3GB + .78GB	3GB + .32GB	**3GB + .65GB**

TABLE I: Average runtimes and memory usage of our three encodings. Memory is reported as JVM usage + CUDD usage.

MDP is input into PRISM [25], where synthesis is performed using an external automata tool for LTL_f [28]. We aim to answer three questions: 1) Which encoding is most efficient? 2) What benefit do we get from encoding information in the model as opposed to the specification? 3) How does probabilistic synthesis compare to reactive synthesis?

We consider stacking scenarios for 3 through 6 blocks (goals shown in Fig. 8a-d), based on [12]. We additionally consider grid-based problems (see Fig. 8e): *temporal* with a complex goal based on wiping each part of table and *diagonal* where blocks are placed along diagonals. Finally, we consider the original 5 blocks, but using a compound specification, rather than the model, to enforce pick and place constraints.



(a) 3 block (b) 4 block (c) 5 block (d) 6 block (e) Grid

Fig. 8: Scenes used in our experiments

Table I shows the relative scalability of our three encodings. A timeout of 20hrs is used. Note that while the integer encoding uses less memory on some domains, its runtime is significantly worse than the other approaches. The location encoding typically has the best runtime, but uses slightly more memory than the object encoding. In Table I, comparing "5 compound" to "5 blocks" shows that we get significant speedup and memory reduction by encoding constraints in the model rather than the specification. This was expected following the results in [12].

In reactive synthesis [12], the robot assumes an adversarial human and chooses its policy accordingly. In our experiments, the human is modeled probabilistically. If the human is adversarial, the robot will compute a conservative plan; however, if the human model is cooperative, the robot will be less conservative. In the best case, the robot may rely on the human to help, taking actions that anticipate human aid. In Fig. 7, the robot anticipates human cooperation at the initial state and thus chooses to grasp the black block, which must be the top of the arch. If the human was not helpful, the robot would need to replace the black block, place the white block, then place the black block at the top of the arch. Here the robot has an 80% chance of success, due to the possibility that human actions or stochastic robot executions may violate the specification. In reactive synthesis, any scenario with probability less than 1 would merely be unrealizable.

In terms of runtime, the complexity of P_s, P_e, and [12]'s k, which limits the number of human actions, affect the comparison. In our 3-6 block experiments, which parallel Fig. 2.b of [12], probabilistic synthesis is more efficient than reactive synthesis, though the runtimes are within an order of magnitude. Note that [12] uses the *object* encoding. 6 blocks with $k = 3$ takes approximately $660s$ in reactive synthesis vs $520s$ for probabilistic synthesis with the same encoding.

A. Physical Validation

We validate out method on a physical UR5 robot. We use Vicon cameras to track the locations of several objects and one human around the robot's workspace. We then extract the semantic meaning of the scene, and identify the corresponding state in our MDP. We use online motion planning with Robowflex [29] to follow the optimal policy computed by PRISM. A video is available [30].

VI. Conclusion

We formalized the problem of probabilistic synthesis for robotic manipulation. Based on this formalism, we presented a method to construct an MDP, which allows us to solve the problem using existing tools. We examined the scalability of three encodings of our model. Compared to reactive synthesis, probabilistic synthesis is less pessimistic, and can model different properties about the world. Additionally, it allows us to remove the limit on the number of human actions and to synthesize policies for problems that would be unrealizable for reactive synthesis. For future work, we will combine models of reactive synthesis and of the current probabilistic synthesis into a stochastic game.

REFERENCES

[1] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-logic-based reactive mission and motion planning," in *IEEE Transactions on Robotics*, vol. 25, no. 6, 2009-12, pp. 1370–1381.

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

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

[4] D. Harel and A. Pnueli, "On the development of reactive systems," in *Logics and models of concurrent systems*. Springer, 1985, pp. 477–498.

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

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

[7] M. Gharbi, R. Lallement, and R. Alami, "Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search." in *IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 6360–6365.

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

[9] A. Pnueli, "The temporal logic of programs," in *IEEE Annual Symposium on Foundations of Computer Science*, 1977, pp. 46–57.

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

[11] ——, "Automated abstraction of manipulation domains for cost-based reactive synthesis," in *IEEE Robotics and Automation Letters*, vol. 4, no. 2, 2018, pp. 285–292.

[12] K. He, A. M. Wells, L. E. Kavraki, and M. Y. Vardi, "Efficient symbolic reactive synthesis for finite-horizon tasks," in *IEEE Int. Conf. on Robotics and Automation*, 2019, pp. 8993–8999.

[13] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces." in *Int. Joint Conf. on Artificial Intelligence*, vol. 13, 2013, pp. 854–860.

[14] E. M. Wolff, U. Topcu, and R. M. Murray, "Robust control of uncertain markov decision processes with temporal logic specifications," in *IEEE Conf. on Decision and Control*, 2012, pp. 3372–3379.

[15] J. Fu and U. Topcu, "Pareto efficiency in synthesizing shared autonomy policies with temporal logic constraints," in *IEEE Int. Conf. on Robotics and Automation*, 2015, pp. 361–368.

[16] D. Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. Vincentelli, S. Sastry, and S. Seshia, "Data-driven probabilistic modeling and verification of human driver behavior," *AAAI Spring Symposium - Technical Report*, 2014.

[17] D. L. Kovacs, "BNF definition of PDDL 3.1," *Manuscript from the IPC-2011 website*, 2011.

[18] M. Helmert, "The fast downward planning system." *J. of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[19] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y. Vardi, "Symbolic LTLf synthesis," in *Int. Joint Conf. on Artificial Intelligence*. AAAI Press, 2017, pp. 1362–1369.

[20] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[21] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[22] A. M. Wells, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "LTLf synthesis on probabilistic systems," in Proceedings 11th International Symposium on *Games, Automata, Logics, and Formal Verification,* Brussels, Belgium, September 21-22, 2020, ser. Electronic Proceedings in Theoretical Computer Science, J.-F. Raskin and D. Bresolin, Eds., vol. 326. Open Publishing Association, 2020, pp. 166–181.

[23] G. De Giacomo and M. Y. Vardi, "Synthesis for LTL and LDL on finite traces," in *Int. Joint Conf. on Artificial Intelligence*, vol. 15, 2015, pp. 1558–1564.

[24] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA, USA: MIT Press, 1988.

[25] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Int. Conf. on Computer Aided Verification*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[26] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," in *IEEE Transactions on Computers*, vol. 100, no. 8, 1986, pp. 677–691.

[27] F. Somenzi, "CUDD: CU decision diagram package release 2.2.0," 1998.

[28] A. M. Wells, "LTLf for PRISM." [Online]. Available: https://github.com/andrewmw94/ltlf_prism

[29] Z. Kingston and L. E. Kavraki, "Robowflex: Robot motion planning with *MoveIt* made easy," *IEEE Robotics and Automation Letters*, 2021, under Review. [Online]. Available: https://arxiv.org/abs/2103.12826

[30] A. M. Wells, "Probabilistic manipulation with LTLf." [Online]. Available: https://www.andrewmwells.com/probabilistic-manipulation-with-ltlf/