To appear in *Proceedings of International Conference on Robotics and Automation 2019*, Montreal, Canada.

Efficient Symbolic Reactive Synthesis for Finite-Horizon Tasks

Keliang He[1], Andrew M. Wells[1], Lydia E. Kavraki[1], and Moshe Y. Vardi[1]

***Abstract*— When humans and robots perform complex tasks together, the robot must have a strategy to choose its actions based on observed human behavior. One well-studied approach for finding such strategies is reactive synthesis. Existing approaches for finite-horizon tasks have used an explicit state approach, which incurs high runtime. In this work, we present a compositional approach to perform synthesis for finite-horizon tasks based on binary decision diagrams. We show that for pick-and-place tasks, the compositional approach achieves orders-of-magnitude speed-ups compared to previous approaches. We demonstrate the synthesized strategy on a UR5 robot.**

I. Introduction

Robots are working increasingly closely with humans. In scenarios such as human-robot co-assembly and assisted living, robots share the same space as humans. Human behavior, however, is highly unpredictable. To guarantee safe completion of the tasks given to them, robots must handle changes caused by the humans.

Consider a scenario where a manufacturing robot must complete an assembly along with human workers. The assembly may require many steps, and the robot needs to guarantee the completion of the assembly, despite the varying level at which the human may be involved, and even in cases where the human is counter-productive. Figure 1 shows an example of this in the lab, where the robot must build an arch, but the human may move the objects around. In such scenarios, the robot must behave reactively, i.e., choose its action depending on the actions performed by the human.

One approach to achieving this reactive behavior is replanning [1], [2]. In replanning, the robot finds a sequence of actions to complete the task and starts executing the plan. When the human interferes with the task, the robot stops and finds another plan to execute. Replanning, however, does not guarantee task completion. Without considering possible human actions during planning, the robot could reach situations where the task is impossible to complete.

A popular approach for achieving reactive behavior is reactive synthesis [3], [4], [5], [6], [7]. These works find strategies that can determine the correct robot actions at each possible situation and guarantee the completion of the task. Reactive synthesis requires two inputs: a task and a planning domain (also called an abstraction). The task is written as a logical formula, and describes the properties that must hold and their temporal relationships. The planning domain is given as a transition system that describes, on a discrete level, the possible robot and human actions. Reactive synthesis finds the solution strategy by first combining the task and the domain into a game between the robot and the human, and then solving for a winning strategy for the robot in this game. The winning strategy can then be mapped back to the planning domain to guarantee robot task completion under all modeled human behaviors.

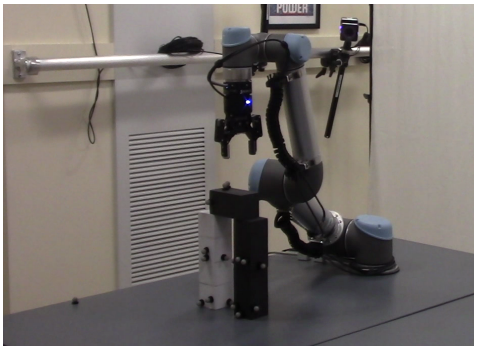


Fig. 1: An arch construction task executed on a UR5 robot. (All figures best viewed in color and on screen.)

In this work, we focus on reactive synthesis for finite-horizon tasks represented using formulas of linear temporal logic on finite traces (LTLf), as described in [7]. These tasks represent problems such as factory assemblies and object rearrangement. They are different from infinite-horizon tasks (such as surveillance tasks) in that they must be determined complete at some point during execution. Note that unlike [7], we do not constrain the amount of resources the robot consumes during execution.

Previous work introduced an algorithm to perform reactive synthesis for finite-horizon tasks and an explicit approach to implement this algorithm [7]. This explicit approach represents each state in the game separately. However, this technique lacked scalability, which prevented it from being applied to larger problems. On the other hand, advancements in LTLf synthesis [8] have shown that using symbolic solving techniques can significantly speed up synthesis. LTLf synthesis is targeted at software synthesis where the input is a formula. Unfortunately, robotic problems require a planning domain that encodes the possible behaviors of the robot and the human. This planning domain is given as a transition system and not as a formula.

In this work, we propose a compositional method to apply symbolic techniques from LTLf synthesis to reactive synthesis for robots. The main challenge is the incorporation of the planning domain in the symbolic LTLf synthesis framework. Our compositional method converts the domain and the task separately into symbolic representations called binary decision

[1]Dept. of Computer Science, Rice University, Houston, TX 77005, USA. This work has been supported in part by NSF 1830549. This work was supported by a NASA Space Technology Research Fellowship. keliang.h@gmail.com, andrew.wells@rice.edu, kavraki@rice.edu, vardi@cs.rice.edu.

diagrams (BDDs) [9], and combines these BDDs into a game, which is solved for a strategy. Due to the compactness of BDDs, we achieve significant performance improvements.

The main contribution of this paper is a compositional approach to reactive synthesis for finite-horizon tasks. This approach leverages symbolic synthesis techniques and outperforms the previous explicit approach. In Section III, we define the problem of reactive synthesis for finite-horizon tasks. In Section IV, we describe the existing explicit approach as well as a monolithic approach inspired by work from reactive synthesis for infinite-horizon tasks. In Section V, we detail the proposed compositional approach. In Section VI, we use a pick-and-place case study to compare synthesis time between the explicit, monolithic, and compositional approaches and demonstrate the scalability of the compositional approach. We also validate the synthesized strategy via physical execution of the arch construction task on a UR5 robot.

II. Related Work

In reactive synthesis for LTLf formulas, [8] has shown that significant performance gains can be achieved by applying symbolic methods. The symbolic data structures used are called binary decision diagrams (BDDs) [9], which are compact representations of sets of boolean assignments. Using BDDs, the synthesis algorithm is able to operate over sets of similar states in a single operation, as opposed to operating on each state individually in the explicit method (see Section IV-C). However, [8] only takes as input a formula. For robotic applications, the planning domain, which encodes the possible behaviors of the robot and the human, is often given as a transition system and not a formula. In this paper, we focus on incorporating the planning domain into the LTLf synthesis process, leveraging the symbolic representations in LTLf synthesis to achieve better performance.

Existing works [10], [11] in infinite-horizon reactive synthesis have also considered the incorporation of the planning domain into symbolic methods [12], [13]. In these works, the planning domain is encoded as a formula which is combined with the task formula. The combined formula is input to a symbolic synthesis tool to find a strategy. This monolithic approach has been shown to be effective for 2D navigation problems for infinite-horizon tasks. There are differences between the problems investigated in [10], [11] and the problems in this paper. First, the tasks investigated in these previous works are infinite-horizon (e.g., surveillance tasks), while in this paper, we focus on finite-horizon tasks (e.g., assembly tasks), which are computationally less complex. Second, the manipulation case study in this paper is higher-dimensional than many of the 2D navigation domains studied in previous work, and therefore more complex. Nonetheless, we implement this monolithic approach, as described in Section IV-D, and compare it with the proposed approach.

III. Problem Definition

The reactive synthesis problem for finite-horizon tasks considers a planning domain where a robot and a human can take actions, and the robot must achieve a given finite task while the human has a limited number of "moves" to interfere with the task.

A. Problem Inputs

1) Planning Domain: The reactive synthesis problem is given on a planning domain. A reactive planning domain is a transition system $G = (V, v_0, A_e, A_s, \Pi, \rho)$, where

- V is a finite set of states;
- v_0 is an initial state;
- A_e and A_s are human and robot actions respectively. Each action $a \in A_e \cup A_s$ is a partial function from V to V;
- Π is a set of task relevant propositions;
- $\rho : V \to 2^{\Pi}$ is the predicate function that determines the truth value of the propositions from the states.

In this paper, the planning domain will be specified using the planning domain description language (PDDL) [14], a standard language in the field of AI planning. In a PDDL domain, functions over objects determine the state of the transition system. For example, in a block-stacking scenario, a function $at : Blocks \to Locations$ would map the blocks to their current locations. Thus, the instantiation of the function at can determine where objects are, and therefore the state of the transition system. Actions in PDDL have preconditions and effects. The preconditions must be satisfied for the action to be applied. The effects indicate the change to the functions caused by the action.

For reactive planning, we extend PDDL to differentiate human and robot actions. We always allow the human to choose to take no action. The pick-and-place domain is detailed in Section VI.

2) Temporal Task: To express the robot task, we use *linear temporal logic over finite traces* (LTLf) [15]. An LTLf formula ϕ is defined over the set of propositions Π to describe how the truth assignment to Π changes over time. The syntax for an LTLf formula is identical to LTL [16] and is defined recursively as:

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

where $p \in \Pi$ and ψ_1, ψ_2 are also LTLf formulas.

The semantics of LTLf formulas are defined over finite sequences $(w_0, w_1, ..., w_n)$ of truth assignments $w_i \in 2^{\Pi}$, called traces. Unlike LTL where the semantics are defined over infinite traces, reasoning over finite traces allows LTLf to describe finite-horizon tasks such as assembly and delivery. The semantics of LTLf are as follows.

- $w, i \models \top$;
- $w, i \models p$ 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 also include the following shorthands:

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

To describe the robot task, we use an LTLf formula ϕ over the task relevant propositions Π. The task is completed if the trace incurred $(\rho(v_0), \rho(v_1), ..., \rho(v_h))$ satisfied ϕ.

For example, in the arch-building task, the task is

$$\Diamond(\text{Block}_1\text{-at-top}) \bigwedge \Box(\bigwedge_{l_2 \text{ supports } l_1} l_1\text{-occupied} \rightarrow l_2\text{-occupied}). \quad (1)$$

Block_1-at-top indicates whether block_1 is at the top location on the arch. l-occupied indicates if location l is occupied, and is written as a disjunction of Block_i-at-l for all blocks. The formula says that block_1 should be at the top of the arch, but also ensures each object is supported.

3) Human Action Limit: We also assume that the human only has a limited number of moves K. This assumption is placed because for many domains, the human is more capable than the robot, and if we allow the human to keep performing actions, the robot has no way to guarantee task completion.

B. Solution strategy

A strategy for the robot on a reactive domain is a mapping $Str : V^* \rightarrow A_s$, where the robot chooses an action according to the history of domain states. A strategy is winning for the robot if for every infinite execution $(v_0, v_1, ...)$ that starts at the initial state v_0 and follows transitions from Str and at most K actions from A_e, then we can find $h > 0$ such that the trace $(\rho(v_0), \rho(v_1), ..., \rho(v_h))$ satisfies ϕ. In other words, a winning strategy guarantees that at some finite horizon h, the task is satisfied. The goal of reactive synthesis is to find a winning strategy for the robot.

IV. Existing Synthesis Approaches

A. Overview of the Synthesis Algorithm

In previous work [7], we proposed an algorithm for solving the reactive synthesis problem. The explicit, monolithic, and compositional approaches compared in this paper all implement this algorithm, using different encodings and tools.

In this algorithm, the task is first converted to a deterministic finite automaton (DFA). Then, the DFA and the planning domain are combined to construct a game between the robot and the human, and a strategy is found for the game.

A DFA $A = \{Z, z_0, \delta, Z_f\}$ over a set of boolean propositions Π is a transition system where

- Z is a finite set of states;
- z_0 is the initial state;
- Z_f is a set of final states;
- $\delta : Z \times 2^\Pi \rightarrow Z$ is a transition function that determines the next states given the current state and the set of truth assignment to Π.

We say a trace $w = (w_0, w_1, ..., w_n)$ of Π is accepted by A if by starting in z_0 and applying $\delta(z_i, w_i) = z_{i+1}$ for $0 \leq i \leq n$, we terminate in a final state $z_{n+1} \in Z_f$.

We convert the task ϕ into a DFA A_ϕ that accepts exactly the traces that satisfy ϕ [17]. We then take the product of the DFA and the domain G to create a game $P = (S, s_0, S_f, T_s, T_e)$, defined as the following:

- $S = Z \times V \times \mathbb{Z}_K$ is the set of game states. Each state encodes the current DFA state, the domain state, and the number of human actions remaining.
- $s_0 = (z_0, v_0, k)$ is the initial state.
- $S_f = \{(z, v, k) | z \in Z_f\}$ are the final states.
- T_s is a set of partial functions from S to S inherited from the robot actions A_s. If a_s is defined from v, then the corresponding t_s is defined as $t_s((z, v, k)) = (\delta(z, \rho(a_s(v))), a_s(v), k)$.
- T_e is similarly inherited from A_e, but only defined from states where $k \neq 0$. The transition is $t_e((z, v, k)) = (\delta(z, \rho(a_e(v))), a_e(v), k - 1)$.

Intuitively, the game states propagate by first applying the action to find the next domain state, then using the proposition assignment at the new state to advance the DFA state. The count of human actions remaining is reduced if the processed action is a human action. The task is achieved if the execution leads the game from the initial state to a final state.

The game is then solved using a fixed-point approach. We incrementally build a set of game states called the winning set W_i, where the robot can force a win. Initially, we let the winning set $W_0 = S_f$. Then, at each iteration i, we add states to the winning set if all results of a human move are in the winning set and the robot can force the game into the winning set in one move, $W_{i+1} = W_i \cup \{s \in S | (\forall t_e \in T_e, t_e(s) \in W_i) \wedge (\exists e_s \in T_s, t_s(s) \in W_i)\}$. This process terminates when s_0 is added, at which point a strategy is found. If a fixed-point is reached, i.e., no more states can be added, then no strategy exists for the game.

B. An Explicit Approach

In our previous work [7], the game is represented explicitly as states and transitions. Each state is recorded using an assignment to the domain functions and predicates. The winning set maintains an explicit set of states. At each iteration of the fixed-point computation, we check all game states that are not in W_i, and decide if they should be added to W_{i+1}. This causes significant scalability issues because the number of possible game states is exponential in the size of the domain. Thus scaling to larger domains is intractable, as will be shown in Section VI. Empirically, this approach only scaled to pick-and-place domains with 3 objects.

C. Symbolic LTLf Synthesis

To avoid explicitly constructing the states and transitions in the game, we leverage recent work in symbolic LTLf synthesis [8]. LTLf synthesis solves a similar problem to reactive synthesis, but the input is only a LTLf formula ϕ. The propositions of the formula are not generated by a planning domain, but are instead partitioned into inputs $\mathcal{X}$ and outputs $\mathcal{Y}$, directly controlled by the environment (human), and the system (robot), respectively. The goal is to find a strategy that chooses an assignment to $\mathcal{Y}$ that guarantees ϕ is satisfied.

Symbolic LTLf synthesis leverages a data structure called a binary decision diagram (BDD) [9] to achieve fast synthesis. BDDs are compact representations of boolean functions that map a set of boolean variables to a boolean output (true or false). BDDs can also encode a set of boolean assignments by considering the set as a function that maps all assignments in the set to true, and everything else to false. Common operations on functions such as evaluation and composition, as well as set operations such as union, intersection, and complementation have efficient implementations on BDDs.

Symbolic LTLf synthesis [8] finds a strategy by first converting the LTLf formula into a DFA A_ϕ. The traces that satisfy ϕ are exactly those that are accepted by A_ϕ. This DFA is represented compactly using BDDs. The states are numbered and encoded using a set of variables $\mathcal{Z}$ as bit vectors. The set of final states Z_f is written as a function $Z_f = 2^{\mathcal{Z}} \rightarrow \{0, 1\}$. The transition function δ is encoded as a boolean function from $\mathcal{Z} \times \mathcal{X} \times \mathcal{Y}$ to true or false for each bit in $\mathcal{Z}$. Once the DFA is constructed, the synthesis algorithm performs a fixed-point computation and finds a strategy that maps $\mathcal{Z}$ to $\mathcal{Y}$. During this computation, the winning set and strategy are compactly maintained as BDDs, as previous work showed this is more efficient than explicit synthesis for a benchmark of randomly-generated problems [8]. Symbolic synthesis is utilized in both the monolithic approach (IV-D) and our new approach (V).

D. A Monolithic Approach

Instead of using the explicit approach, one way to leverage symbolic LTLf synthesis is to combine the planning domain and the task as a single LTLf formula, and call the existing symbolic LTLf synthesis tool. This is similar to the approach taken in infinite-horizon reactive synthesis [11], [10].

We generate the LTLf formula

$$\phi = \phi_0 \wedge \phi_{trans} \wedge \phi_{pre-s} \wedge ((\phi_h \wedge \phi_{pre-e}) \implies \phi_{goal}),$$

where

- ϕ_0 describes the initial state;
- ϕ_{goal} is the task given by the user;
- ϕ_{pre-e} asserts at all times steps, the human respects its action preconditions;
- ϕ_{pre-s} asserts at all times steps, the robot respects its action preconditions;
- ϕ_h ensures the number of human actions does not exceed the limit;
- ϕ_{tran} describes the relationship between the current and next state given the action chosen. This subformula is written using add-lists, delete-lists and framing axioms as commonly done in SAT-Plan [18].

Essentially, the formula states that the initial condition and the transition function must hold, and the robot must choose legal actions, and if the human performs legal actions, then the task must be achieved.

We then give this formula to an existing symbolic LTLf synthesis tool [8].The result is a strategy that determines the action to take and, given the previous state and the human action chosen, the next state of the planning domain.

In the case study in Section VI, we see that this monolithic approach has a bottleneck in memory usage during the translation from the LTLf formula to the DFA. In practice, this bottleneck prevents the monolithic approach from solving large problems.

V. Proposed Compositional Approach

The monolithic approach fails to scale well because the formula we input to the LTLf synthesis tool is unnecessarily large. Using this insight, we propose the compositional approach, which is the main contribution of this paper. Since the planning domain is defined by functions and sets, we can encode the planning domain directly as BDDs, and pass only the task through the conversion from LTLf to a DFA. The task is often a much smaller formula than the domain, thus we alleviate the LTLf to DFA conversion bottleneck. Once the task is converted to a DFA, the compositional approach directly combines it with the BDDs representing the planning domain to generate a product DFA. The symbolic synthesis tool then computes a strategy on this product DFA. Specifically, we perform the following steps.

We first create the variables Π_d representing the planning domain state and the number of human actions remaining. The state includes π_l (loss) and π_w (win) representing the robot and the human violating the preconditions of the planning domain respectively. We create variables Π_e and Π_s to represent the human and robot actions respectively.

For each variable in Π_d, we generate a BDD that describes whether this variable is true on the next step, given the current state and the actions chosen. These BDDs combine to represent a function $S2S : 2^{\Pi_d \cup \Pi_e \cup \Pi_s} \rightarrow 2^{\Pi_d}$, that describes the transition function of the planning domain.

We then create BDDs that map Π_d to the truth value of the propositions in Π. These BDDs represent a function $S2P : 2^{\Pi_d} \rightarrow 2^{\Pi}$, that describes the predicate function of the planning domain.

We construct the formula $\square(\neg\pi_l) \wedge (\lozenge(\pi_w) \vee \phi)$, essentially saying that the robot should respect action preconditions, and if the human respects action preconditions and number of actions available, then the task should be satisfied. We input this formula to the symbolic LTLf synthesis tool in [8], and generate a DFA represented by its transition function $\delta : 2^{\mathcal{Z}} \times 2^{\Pi} \rightarrow 2^{\mathcal{Z}}$ and its final states $Z_f : 2^{\mathcal{Z}} \rightarrow \{0, 1\}$ encoded as BDDs. For details on translation from LTLf to the BDD-represented DFA, see [8]. Essentially, the states of the DFA, represented by assignments of $\mathcal{Z}$, describe the progress toward completion of the task. The transition function δ describes how the task state progresses according to the output from the planning domain.

We then construct a product DFA A_p, where the state variables are $\mathcal{Z} \cup \Pi_d$, inputs are Π_e, and outputs are Π_s. The transition function is represented as $\delta_p((v, z), a_e \cup a_s) = (S2S(v, a_e \cup a_s), \delta(z, S2P(v))$. In other words, from a state with planning domain state v and task state z, if the human performs a_e and the robot performs a_s, then the next domain state is found using the planning domain transition $S2S(v, a_e \cup a_s)$, and the next task state is found using the

DFA transition with the predicate function $\delta(z, S2P(v))$. This is essentially the game construction from section IV-A.

Finally, we use the synthesis tool from [8] to generate a strategy for this game. This tool performs a fixed-point computation similar to that described in Section IV-A, but uses BDD operations instead of explicit state operations. Specifically, for the explicit approach, the existential and universal quantification operations are linear in the number of product DFA states. Using BDD representations, the universal quantification is worst-case quadratic in the size of the BDD, and the existential quantification is constant time (due to the variable ordering in the BDDs). In the case study in Section VI, we observe that the size of the BDD is much smaller than the number of explicit states, thus we can compute the fixed-point much faster using the compositional approach.

The states of the product DFA are the combined task and domain states, and the output variables represent the robot action, so the resulting strategy is a mapping from the combined task and domain states to actions for the robot. Following this strategy, we ensure that the robot never violates an action precondition and will achieve the task as long as the human does not violate its conditions.

VI. Case Study

To compare the scalability of the explicit, monolithic, and compositional approaches discussed in this paper, we implement each approach and test them on a pick-and-place scenario with varying parameters. In this scenario, a set of objects and a finite set of locations are defined. The human can move any object to any location. The robot can perform grasp/place actions to pickup and drop objects, as well as transfer/transit actions to move the robot arm with/without an object in its gripper. This scenario is designed to simulate possible human-robot co-assembly tasks.

The explicit, monolithic, and compositional approaches are implemented in C++. Translation from LTLf to DFA as well as symbolic game strategy synthesis are performed using Syft [8], which uses MONA [19] internally. BDDs are constructed using the CUDD [20] package. Computation time is measured on a workstation with a quad core 4GHz processor with 32G memory and averaged over 10 runs.

A. Runtime

We vary the number of objects $|O|$, locations $|L|$, and human actions allowed K, in order to test the scalability of each approach. The goal for each task is to rearrange the objects so that each object is at a specific location. Overall, we observe that for the pick-and-place domain, the compositional approach achieves a significant speedup over the explicit and monolithic approaches, and does not experience the memory bottleneck of the monolithic approach.

Figure 2a shows the total runtime for varying numbers of locations. We see that the runtime for the explicit approach grows as a polynomial with the number of locations.

On average, the monolithic approach is 2.7 times faster than the explicit approach. Note that synthesis time for each test case varies drastically. This is because specific parameter

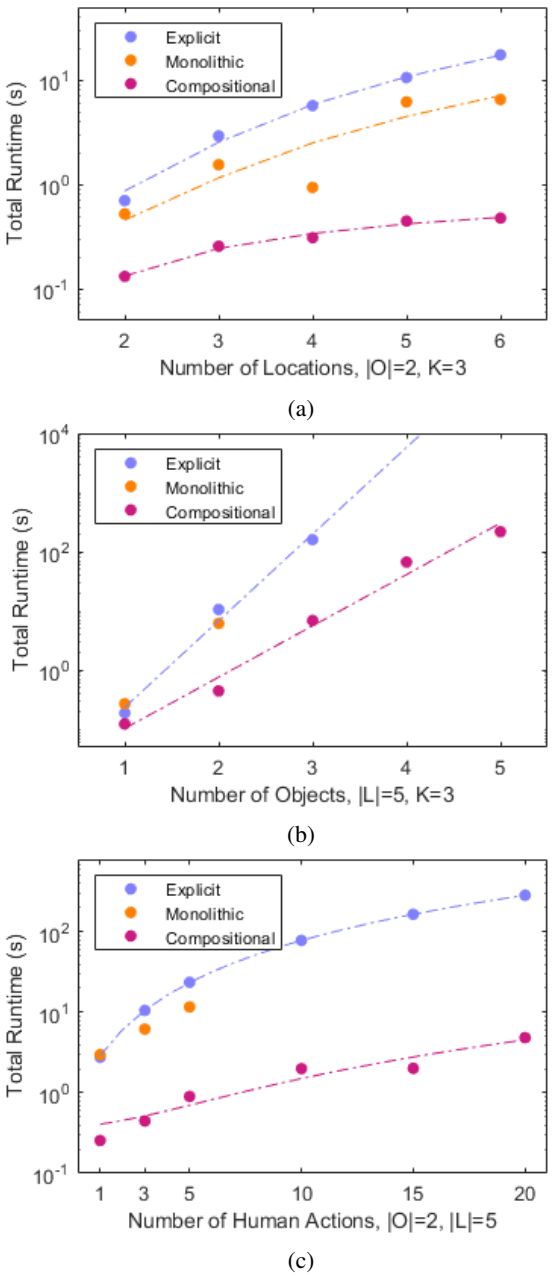


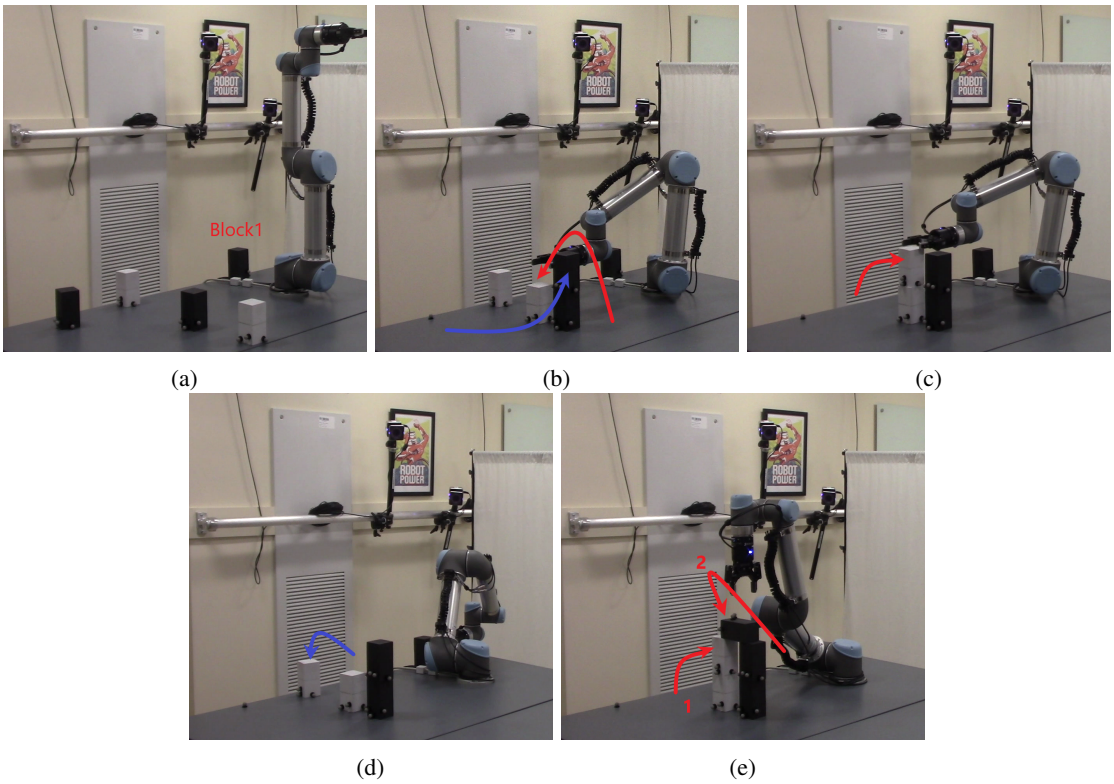
Fig. 2: Average end-to-end runtime for synthesis using different approaches (log-scale). Dashes lines show exponential (b) or quadratic functions (a, c) fit to the corresponding data.

values (in this case, the number of locations) could affect the symmetry of the combined formula, thus reducing the runtime even though the problem itself is larger.

The compositional approach, on the other hand, is over an order of magnitude faster than both the monolithic and explicit approaches. We observe a larger speedup for a larger number of locations, with a 37 times speedup over the explicit approach for 6 locations.

Figure 2b shows the total runtime for varying numbers of objects. The runtime for the explicit approach grows exponentially, and quickly times out at 500 seconds with only 3 objects.

We observe that the monolithic approach runs out of



(a) (b) (c)

(d) (e)

Fig. 3: Execution of the arch construction task. (a) Initial condition; block_1 needs to be placed at top of arch. (b) The robot places the other base (red) while the human helps by placing one of the upper supports (blue). (c) The robot places the other upper support. (d) As the robot is reaching for the top object, the human interferes by removing an upper support. (e) The robot recovers the misplaced support and finishes the task.

memory during the translation from LTLf to DFA starting at 3 objects. In particular, we found that between 40% to 53% of the runtime for the monolithic approach is spent on translating the formula to a DFA prior to synthesis. This bottleneck was also observed in [8]. This prevents the monolithic approach from handling large problems in the pick-and-place domain.

The compositional approach also scales exponentially with the number of objects, but exhibits a much smaller rate of growth on the log plot, indicating an exponential speedup over the explicit approach. Even for a difficult problem with 5 objects, the compositional approach was able to find a strategy using an average of 219 seconds.

Figure 2c shows the runtime with respect to the number of human actions allowed. The monolithic approach runs out of memory beyond 5 actions allowed. The explicit approach demonstrates a quadratic growth in runtime. The compositional approach, however, seems to perform sub-quadratically. This is due to the fact that as the number of human actions grows, the BDD representation of the planning domain does not necessarily increase in size. In this case, $K = 10$ and $K = 15$ both require 4 bits to represent the human actions, thus the runtime increase from $K = 10$ to $K = 15$ is very small.

B. Physical Execution

We validate the correctness and feasibility of the strategy synthesis tool by exporting it as a ROS [21] package and incorporating it on a UR5 robot. Figure 3 shows an example of the robot performing the arch construction task using the strategy found by the compositional approach. The LTLf formulation of the task is Formula (1). This task is more complex temporally than the rearrangement problems in Section VI-A, but we do not allow the human to place objects at the top of the arch.

We see that the robot successfully completes the arch, despite the human interfering twice. The first time the human was helpful, and the robot takes advantage of the help. The second time the human action hinders the task, but the robot is able to recover. This problem was not synthesizable with a 500 seconds timeout for the explicit and monolithic approaches. The compositional approach was able to synthesize a strategy in 24 seconds. Specifically, there are over 2.8 million explicit game states, while the final BDD that represents the strategy only contains 2604 BDD nodes.

VII. Conclusion and Discussion

In this paper, we present a reactive synthesis algorithm for finite-horizon tasks expressed in LTLf. By encoding the planning domain and the task as BDDs and combining them at the BDD level, we achieve orders-of-magnitude speed-up over existing explicit approaches on a pick-and-place domain and break the scalability bottleneck of monolithic approaches.

Whether the runtime experiments extend to other domains, such as navigation domains, remains open for further investigation. Also open for investigation is whether using the compositional approach instead of a monolithic approach

could improve the runtime performance of reactive synthesis for infinite-horizon tasks.

REFERENCES

[1] 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 (ICRA)*. IEEE, 2013, pp. 5025–5032.

[2] 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.

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

[4] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[5] C. I. Vasile and C. Belta, "Reactive sampling-based temporal logic path planning," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 4310–4315.

[6] E. M. Wolff, U. Topcu, and R. M. Murray, "Efficient reactive controller synthesis for a fragment of linear temporal logic," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 5033–5040.

[7] 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)*. Vancouver, BC, Canada: IEEE, 2017, pp. 5326–5332.

[8] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y. Vardi, "Symbolic LTLf synthesis," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. AAAI Press, 2017, pp. 1362–1369.

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

[10] C. Finucane, G. Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, temporal logic and robot control," in *IEEE/RSJ International Conference on Intelligent Robots (IROS)*. IEEE, 2010, pp. 1988–1993.

[11] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with TuLiP: The temporal logic planning toolbox," in *Control Applications (CCA), 2016 IEEE Conference on*. IEEE, 2016, pp. 1030–1041.

[12] A. Pnueli, Y. Sa'ar, and L. D. Zuck, "JTLV: A framework for developing verification algorithms," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 171–174.

[13] R. Ehlers and V. Raman, "Slugs: Extensible GR (1) synthesis," in *International Conference on Computer Aided Verification*. Springer, 2016, pp. 333–339.

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

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

[16] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science, 1977., 18th Annual Symposium on*. IEEE, 1977, pp. 46–57.

[17] G. De Giacomo and M. Y. Vardi, "Synthesis for LTL and LDL on finite traces." in *International Joint Conferences on Artificial Intelligence (IJCAI)*, vol. 15, 2015, pp. 1558–1564.

[18] H. Kautz and B. Selman, "SATPLAN04: Planning as satisfiability," 2006.

[19] J. G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm, "Mona: Monadic second-order logic in practice," in *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 1995, pp. 89–110.

[20] F. Somenzi, "CUDD: CU Decision Diagram Package 3.0.0. Universiy of Colorado at Boulder," 2016.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.