Efficient Symbolic Approaches for Quantitative Reactive Synthesis with Finite Tasks

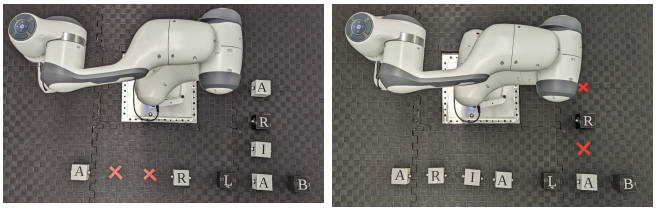
Karan Muvvala and Morteza Lahijanian

***Abstract*— This work introduces efficient symbolic algorithms for quantitative reactive synthesis. We consider resource-constrained robotic manipulators that need to interact with a human to achieve a complex task expressed in linear temporal logic. Our framework generates reactive strategies that not only guarantee task completion but also seek cooperation with the human when possible. We model the interaction as a two-player game and consider regret-minimizing strategies to encourage cooperation. We use symbolic representation of the game to enable scalability. For synthesis, we first introduce value iteration algorithms for such games with min-max objectives. Then, we extend our method to the regret-minimizing objectives. Our benchmarks reveal that our symbolic framework not only significantly improves computation time (up to an order of magnitude) but also can scale up to much larger instances of manipulation problems with up to 2× number of objects and locations than the state of the art.**

I. Introduction

Robots are experiencing a boom in their deployment in the real world. Examples include warehouse, assembly, delivery, home assistive, and planetary exploration robots. As they transition from robot-centric workspaces to the unstructured environments, robots must be able to achieve their tasks through interactions with other agents while dealing with finite resources. This is a computationally challenging problem because planning a reaction for every possible action of other agents, known as *reactive synthesis* problem, in face of resource constraints is expensive. The computational challenge is exacerbated in robotic manipulators where tasks are often complex and robots have high degrees of freedom. More specifically, when faced with humans, the problem becomes even more difficult because we expect robots to have meaningful interactions with us while still trying to complete their tasks without exceeding their resource budget. This work focuses on this challenge and aims to develop efficient quantitative reactive synthesis algorithms for manipulation problems with high-level tasks and limited resource budget in the presence of a human.

Our approach is based on modeling the robot-human interaction with resource constraints as a quantitative two-player game. We use *Linear Temporal Logic over finite trace* (LTLf) [1] for precise specification of tasks that can be accomplished in finite time. Using the game-theoretic approach allows us to base our method within the reactive synthesis framework to generate correct-by-construction strategies that guarantee to satisfy the task while enabling the robot to have meaningful interaction with the human. Specifically, we build on the results of [2], which show resource-minimizing strategies emit "unfriendly" interactions because the human is viewed as an adversarial agent, and hence, these strategies attempt to avoid the human by all means. Instead, *regret-minimizing* strategies, which evaluate the cost of actions in the hindsight relative to a resource budget, are "friendly" because they seek cooperation with the human up until resources are endangered. The challenge with regret-minimizing strategies is that they are finite-memory, requiring the knowledge of history to evaluate regret, leading to a computational below-up (both in time and space) on an already-difficult problem.



(a) Adversarial Behavior (b) Regret Behavior

Fig. 1: "ARIA LAB" construction. (a) resource-minimizing strategy via [3]. (b) regret-minimizing strategy via [2].

To mitigate the computational bottleneck, we propose symbolic algorithms for quantitative games. We represent the manipulation game symbolically, which significantly reduces memory usage. Further, we introduce value iteration algorithms for quantitative symbolic games with min-max objectives. Finally, we extend our symbolic approach to the regret framework in [2]. Case studies and benchmarks reveal that our symbolic framework not only significantly improves computation time (up to an order of magnitude) but also can scale up to way larger instances of manipulation problems with up to 2× number of movable objects and locations over the state of the art.

In short, the contributions of this work are fourfold: (i) the *first* symbolic value iteration algorithm using the compositional encoding for two-player quantitative games, to the best of our knowledge, (ii) an efficient symbolic synthesis algorithm for regret-minimizing strategies, (iii) an open-source tool for quantitative symbolic reactive synthesis with Planning Domain Definition Language (PDDL) plug-in, and (iv) a series of benchmarks and illustrative manipulation examples that show the efficacy of the proposed symbolic framework in scaling up to real-world problems.

Related Work Reactive synthesis has recently been studied for both mobile robots [4], [5] and manipulators [6], [7]. The approaches to mobile robots are based on

Authors are with the Department of Aerospace Engineering Sciences at the University of Colorado Boulder, CO, USA {firstname.lastname}@colorado.edu

qualitative games. Work [3], [8] in robotic manipulation use quantitative turn-based game abstraction to model the interaction between the human and the robot. By assuming the human is adversarial, winning strategies are synthesized for the robot that guarantee task completion. While this approach has nice task completion guarantees with resource constraints, the adversarial assumption is conservative and results in one-sided, uncooperative interactions. Recently, [9] relaxed this assumption by modeling the human as a probabilistic agent. They use data gathered from past interactions to model the interaction as a Markov Decision Process. This approach results in better interactions, but the synthesized strategies reason about the human behavior in expectation, rather than as a strategic agent. In the most recent work [2], the notion of *regret* is employed to incorporate cooperation-seeking behaviors for the robot while still preserving the task completion guarantees. Using this approach produces "meaningful" interactions, but the robot needs to keep track of past human actions, leading to major computational issues.

To enable scalability of reactive synthesis, recent work [10] focuses on symbolic approaches. They use Binary Decision Diagrams (BDDs) to represent sets of states and transitions as boolean functions. With this representation, the robot can efficiently reason about the effects of its actions on the world by manipulating boolean formulas. Work [10] shows how planning with LTLf [11] symbolically allows much better scalability for qualitative manipulation games. In [12], a hybrid approach to symbolic min-max game is formulated. The states are represented explicitly, while the states values are stored symbolically. Work [13] extends this framework to purely symbolic algorithms for specifications defined over the infinite horizon GR(1). However, such infinite tasks are not suitable for robotic manipulation tasks.

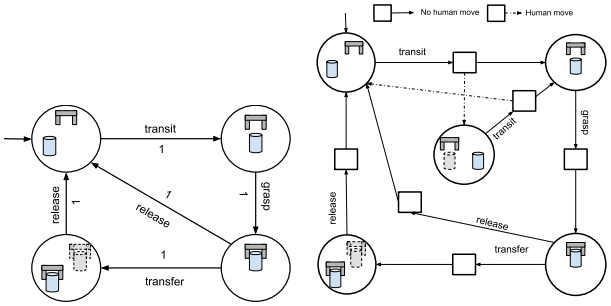
II. Problem Formulation

A. Manipulation Domain Abstraction

The problem of planning for a robot in interaction with a human is naturally continuous with high degrees of freedom. Due to difficulty of reasoning in such spaces, they are commonly abstracted to discrete models [2], [3], [8], [9], known as abstraction. Similar to [2], [3], we use a two-player game abstraction to model the human-robot interaction.

Definition 1 (Two-player Manipulation Domain Game)**.** *The two-player turn-based manipulation domain is defines as a tuple $\mathcal{G} = (V, v_0, A_s, A_e, \delta, F, \Pi, L)$, where*

- *$V = V_s \cup V_e$ is the set of finite states, where V_s and V_e are the sets of the robot and human player states, respectively, such that $V_s \cap V_e = \emptyset$, $v_0 \in V_s$ is the initial state,*
- *A_s and A_e are the sets of finite actions for the robot and human player, respectively,*
- *$\delta : V \times (A_s \cup A_e) \to V$ is the transition function such that, for $i, j \in \{s, e\}$ and $i \neq j$, given state $v \in V_i$ and action $a \in A_i$, $\delta(v, a) \in V_j$ is the successor state,*
- *$F : V \times (A_s \cup A_e) \to \mathbb{R}_{\geq 0}$ is the cost function that, for robot $v \in V_s$ and $a \in A_s$, assigns cost $F(v, a) \geq 0$, and $F(v, a) = 0$ for human $v \in V_e$ and $a \in A_e$,*
- *$\Pi = \{p_0, \ldots, p_n\}$ is set of task-related propositions that can either be true or false, and*
- *$L : V \to 2^{\Pi}$ is the labeling function that maps each state $v \in V$ to a set of atomic propositions $L(s) \subseteq \Pi$ that are true in v.*



(a) Transition System (b) Two-player game

Fig. 2: Manipulation Domain $\mathcal{G}$ for Example 1. States in circle and square belong to set V_s and V_e, respectively.

Intuitively, every state in $\mathcal{G}$ encodes the current configuration of the objects and the status of the robot. For instance, each state v is a tuple that captures the locations of all objects in the workspace and the end effector's status as free or occupied. A successor state encodes the status of the objects and the end effector under human and robot action.

Example 1. *Fig. 2a depicts a sample manipulation domain for a robot. Fig. 2b is its extension to a game abstraction $\mathcal{G}$ that captures the interaction between the robot and a human.*

In this game, each player has a strategy to choose actions. Robot strategy $\sigma : V^* \cdot V_s \to A_s$ and human strategy $\tau : V^* \cdot V_e \to A_e$ pick an action for the respectively player given a finite sequence of states (history). A strategy is called *memoryless* if it only depends on the current state. The set of all strategies for the robot and human players are denoted by $\Sigma^{\mathcal{G}}$ and $\mathrm{T}^{\mathcal{G}}$, respectively.

Given strategies σ and τ, the game unfolds, and a *play* $P(\sigma, \tau) = v_0 v_1 \ldots v_n \in V^*$, which is a finite sequence of visited states, is obtained. Intuitively, a play captures the evolution of the game (movement of the objects in the workspace) for some robot and human actions. Each action has a cost associated with it, e.g., energy. We define the *total payoff* $\mathrm{Val}^{v_0}(P(\sigma, \tau))$ associated with a play to be the sum of the costs of the actions taken by the players, i.e.,

$$\mathrm{Val}^{v_0}(P(\sigma, \tau)) := \textstyle\sum_{i=0}^{n-1} F(v_i, a_i, v_{i+1}), \qquad (1)$$

where $a_i = \sigma(v_0 \ldots v_i)$ if $v_i \in V_s$, otherwise $a_i = \tau(v_0 \ldots v_i)$. Further, a *trace* for play $P(\sigma, \tau)$ is the sequence of labels $\rho = L(v_0)L(v_1)\ldots L(v_n)$ of the states of $P(\sigma, \tau)$. Intuitively, the trace represents the evolution of the manipulation domain relative to a given task.

B. LTL over finite Trace (LTLf)

As most of robotic manipulation tasks are desired to be accomplished in finite time, a good choice to specify them unambiguously is LTLf [1]. LTLf has the same syntax as

Linear Temporal Logic (LTL), but the semantics are defined over finite executions (traces).

Definition 2 (LTLf Syntax [1])**.** *Given a set of atomic propositions* Π*, LTLf formula* φ *is defined recursively as:*

$$\varphi := p \mid \neg\varphi \mid \varphi \ \wedge \ \varphi \mid X \ \varphi \ \mid \varphi U \varphi$$

where $p \in \Pi$ *is an atomic proposition, "*$\neg$*" (negation) and "*$\wedge$*" (and) are Boolean operators, and "*X*" (next) and "*U*" (until) are the temporal operators.*

We define commonly used temporal operators "F" (eventually) and "G" (globally) as $F\varphi := \top \ U \ \varphi$ and $G\varphi := \neg F \neg \varphi$. The semantics of LTLf formulas is defined over finite traces $\left(2^{\Pi}\right)^*$ [1]. Notation $\rho \models \varphi$ denotes that trace ρ satisfies formula φ. We say that play $P(\sigma, \tau)$ accomplishes φ, denoted by $P(\sigma, \tau) \models \varphi$, if its trace $\rho \models \varphi$.

Example 2. *For our manipulation domain, a task to build an arch with two support boxes and one colored box on top can be expressed as,* $\varphi_{arch} = F\left(p_{green,\ top} \wedge p_{block,\ support_1} \wedge p_{block,\ support_2}\right) \wedge G\left(\neg(p_{block,\ support_1} \wedge p_{block,\ support_2}) \rightarrow \neg p_{green,\ top}\right)$.

Our goal is to synthesize a strategy for the robot that can interact with the human to achieve its task φ. Specifically, we seek a strategy that guarantees the completion of φ by using no more than a given cost (energy) budget $\mathcal{B} \in \mathbb{R}_{\geq} 0$ under all possible human moves. We refer to such a strategy as a winning strategy. Formally, Robot strategy σ is called *winning* if, for all $\tau \in \mathrm{T}^{\mathcal{G}}$, $P(\sigma, \tau) \models \varphi$ and $\mathrm{Val}(\sigma, \tau) \leq \mathcal{B}$.

The first problem we consider in this work is an efficient method of synthesizing a winning strategy.

Problem 1 (Min-Max Reactive Synthesis)**.** *Given 2-player manipulation domain game* $\mathcal{G}$ *and LTLf task formula* φ*, efficiently generate a winning strategy* σ^* *for the robot that guarantees the completion of task* φ *and minimizes its required budget* $\mathcal{B}$ *under all possible human strategies, i.e.,*

$$\sigma^* = \arg\min_{\sigma \in \Sigma^{\mathcal{G}}} \max_{\tau \in \mathrm{T}^{\mathcal{G}}} \mathrm{Val}^{v_0}(\sigma, \tau) \ \ \textit{s.t.} \ \ P(\sigma, \tau) \models \varphi \ \forall \tau \in \mathrm{T}^{\mathcal{G}}.$$

Note that in the above formulation, the human is viewed as an adversary, i.e., it assumes human takes actions to maximize the cost budget of the robot. In the real world, however, humans are usually cooperative and willing to help. Hence, the robot strategies obtained by the above formulation are typically "unfriendly" to the human, i.e., they seek to avoid interacting with the human. Rather, we desire strategies that seek cooperation with the human while still considering that the human may not be fully cooperative.

A formulation that enables such interaction is regret-based synthesis as shown in [2]. Intuitively, regret is a measure of how good an action is relative to the best action if the robot knew the human's reaction in advance.

Definition 3 (Regret)**.** *Under robot and human strategies* σ *and* τ*, respectively, regret at state* $v \in V$ *is defined as*

$$\mathrm{reg}^v(\sigma, \tau) = \mathrm{Val}^v(\sigma, \tau) - \min_{\sigma'} \mathrm{Val}^v(\sigma', \tau), \qquad (2)$$

where σ' *is an alternate strategy to* σ *for the robot.*

This regret definition uses best-responses, i.e., $\min_{\sigma'} \mathrm{Val}^v(\sigma', \tau)$, to measure how good an action is for a fixed human strategy τ. Hence, the objective is to synthesize strategies for the robot that minimize its regret.

Problem 2 (Regret-Minimizing Reactive Synthesis)**.** *Given 2-player manipulation domain game* $\mathcal{G}$*, LTLf task formula* φ*, and Budget* $\mathcal{B} \in \mathbb{R}_{\geq 0}$*, efficiently compute a winning strategy* σ^* *for the robot that guarantees not only the completion of task* φ *but also* explores possible collaborations *with the human by minimizing its regret, i.e.,*

$$\sigma^* = \arg\min_{\sigma \in \Sigma^{\mathcal{G}}} \max_{\tau \in \mathrm{T}^{\mathcal{G}}} \mathrm{reg}^{v_0}(\sigma, \tau)$$
$$\textit{s.t.} \quad P(\sigma, \tau) \models \varphi, \quad \mathrm{Val}^{v_0}(\sigma, \tau) \leq \mathcal{B} \qquad \forall \tau \in \mathrm{T}^{\mathcal{G}}.$$

There are generally three major challenges in the above problems. (i) Manipulation domains are notoriously known to suffer from the *state-explosion* problem, i.e., given a set L of locations and set O of objects, the size of the abstraction is $\mathcal{O}(|L|^{|O|})$. Here, this challenge is exacerbated by considering an extension of the domain to human-robot manipulation games as well as high-level complex tasks, requiring composition of the game domain with the task space. (ii) These games are quantitative, requiring to reason not only about the completion of the task but also the needed quantity (budget), which adds numerical computations to the already-expensive qualitative algorithms. (iii) While memoryless strategies are sufficient for Problem 1, regret minimizing strategies (Problem 2) are necessarily finite memory, making the computations even more challenging.

Existing algorithms for Problems 1 and 2 are based on explicit construction of the game venue's graph. They are hence severely limited in their scalability and cannot find solutions to real-world manipulation problems in a reasonable amount of time. In addition, for regret-minimizing strategies, the algorithms require large amount of memory (tens of GB) even for small problem instances (see benchmarks in [2]). To overcome these challenges, rather than relying on explicit construction, we base our approach on symbolic methods. We specifically use Boolean variables and operations to construct symbolic games. We then develop algorithms for these symbolic games to approach Problems 1 and 2.

III. Preliminaries

In this section, we briefly introduce the preliminaries needed for our framework. First, we review the explicit algorithm for quantitative games that solves Problem 1, and then present binary decision diagrams, which are structures used to represent Boolean functions.

A. Overview of Explicit Strategy Synthesis

DFA: The approach proposed in [3] for Problem 1 is to first construct a Deterministic Finite Automata (DFA) for a given task φ. A DFA is defined as a tuple $\mathcal{A}_\varphi = (Z, z_0, 2^{\Pi}, \delta_\varphi, Z_f)$, where Z is a finite set of states, z_0 is the initial state, 2^{Π} is the alphabet, $\delta_\varphi : Z \times 2^{\Pi} \rightarrow Z$ is the

deterministic transition function, and $Z_f \subseteq Z$ is the set of accepting states. A run of $\mathcal{A}_\varphi$ on a trace $\rho = \rho[1]\rho[2]\dots\rho[n]$, where $\rho[i] \in 2^\Pi$, is a sequence of DFA states $z_0 z_1 \dots z_n$, where $z_{i+1} = \delta_\varphi(z_i, \rho[i+1])$ for all $0 \leq i \leq n-1$. If $z_n \in Z_f$, the path is called accepting, and its corresponding trace ρ is accepted by the DFA. The DFA $\mathcal{A}_\varphi$ reasons over the task φ and exactly accepts traces that satisfy φ [1]. Thus, $\mathcal{A}_\varphi$ captures all possible ways of accomplishing φ.

DFA Game: Next, the algorithm composes the game abstraction $\mathcal{G}$ with $\mathcal{A}_\varphi$ to construct a DFA game $\mathcal{P} = \mathcal{G} \times \mathcal{A}_\varphi$. This DFA game is a tuple $\mathcal{P} = (S, S_f, s_0, A_s, A_e, F, \delta_\mathcal{P})$, where A_s, A_e, and F are as in Def. 1, $S = S_s \cup S_e$ is a set of states with $S_s = V_s \times Z$ and $S_e = V_e \times Z$, $S_f = V \times Z_f$ is the set of accepting states, $s_0 = (v_0, \delta_\varphi(z_0, L(v_0)))$ is the initial state, and $\delta_\mathcal{P}$ is a transition function such that $s' = \delta_\mathcal{P}(s, a)$, where $s = (v, z)$, $s' = (v', z')$, if $v' = \delta(v, a)$ a and $z' = \delta_\varphi(z, L(v))$. Intuitively, $\mathcal{P}$ augments each state in $\mathcal{G}$ with the DFA state z as a "memory module" to keep track of the progress made towards achieving task φ. The DFA game evolves by first evolving over the manipulation domain $\mathcal{G}$ and then checking the atomic propositions that are true at the current state and update the DFA state accordingly.

Strategy Synthesis: The classical approach for synthesizing winning strategies σ^* on $\mathcal{P}$ is to play a min-max reachability game [14], [15]. Given a set of accepting states S_f, we define operators, $\mathrm{cPreMax}(s)$ and $\mathrm{cPreMin}(s)$, that update the state values as follows.

$$\mathrm{cPreMax}(s) = \max_{s'}(F(s, a, s') + W(s')) \quad \text{if } s \in S_e, \quad (3)$$

$$\mathrm{cPreMin}(s) = \min_{s'}(F(s, a, s') + W(s')) \quad \text{if } s \in S_s, \quad (4)$$

where W is a vector of state-value pair, and $W(s)$ is the value associated with state s. Then, the algorithm for reachability games with quantitative constraints is to compute the (least) fixed point of the operators (3) and (4) by applying the operator over all the states in $\mathcal{P}$. Thus, playing a min-max reachability game reduces to fixed-point computation and is summarized in Alg. 1. While the value-iteration algorithm for min-max reachability is polynomial [14], [15], fixed-point computation over large graphs ($|S|$ is very large) becomes computationally expensive in terms of time and memory. Thus, to mitigate this issue, Symbolic graphs have been employed where the state and transition function can be represented as boolean functions. In the next section, we discuss how to store and manipulate the boolean functions.

B. Symbolic Data structures: BDDs and ADDs

Symbolic Data structures like Binary Decision Diagrams (BDDs) [16], [17] have been extensively used by the model-checking and verification community due to their ability to perform fixed-point computations over large state spaces efficiently [18]–[21]. BDD is a canonical directed-acyclic graph (DAG) representation of a boolean function $f : 2^X \to \{0, 1\}$ where X is a set of boolean variables. To evaluate f, we start from the root node and recursively evaluate the boolean variable $x_i \in X$ at the current node until we reach the terminal node. The order in which the variables appear along each path is fixed and is called Variable ordering. Due to the fixed variable ordering, BDDs are canonical, i.e., the same BDD always represents the same function. The power of BDDs is that they allow compact representation of functions like transition relations and efficiently perform set operations, e.g., conjunction, disjunction, negation.

Algorithm 1: Explicit Value Iteration

Input : Explicit DFA Game $\mathcal{P}$
Output: Winning strategy σ_{win}, Optimal values W

$W(S) \leftarrow \infty$; $\sigma \leftarrow \emptyset$; $\tau \leftarrow \emptyset$; $W' \leftarrow \emptyset$
$W(S_f) \leftarrow$ initialize all accepting state values as zero
while $W' \neq W$ **do**
  $W' = W$
  $W(s) = \mathtt{max}_{s'}(F(s, s') + W'(s'))$ $\forall s \in S_e$
  $\tau_s = \mathtt{argmax}_{s'}(F(s, s') + W'(s'))$ $\forall s \in S_e$
  $W(s) = \mathtt{min}_{v'}(F(s, s') + W'(s'))$ $\forall s \in S_s$
  $\sigma_s = \mathtt{argmin}_{s'}(F(s, s') + W'(s'))$ $\forall s \in S_s$
return W, σ

Example 3. *Consider the explicit Graph $\mathcal{G}$ in Fig. 3a. Denote each state $v \in V$ by its corresponding boolean formula over boolean variables $X = \{x_0, x_1, x_2\}$. This is shown in Fig. 3b. Fig. 3c shows the BDD corresponding to boolean function representing subset of states $V' = \{v_{s1}, v_{s4}\}$, i.e, $f(x_0, x_1, x_2) = (x_0 \wedge x_1 \wedge x_2) \vee (\neg x_0 \wedge x_1 \wedge \neg x_2)$.*

Algebraic Decision Diagrams (ADDs), also known as Multi-Terminal BDDs, are a generalization of BDDs, where the realization of a boolean formula can evaluate to some set [22]. Mathematically, ADDs represent boolean function $f : 2^X \to C$ where X is a set of boolean variables and $C \subset \mathbb{R}$ is a finite set of constants. Similar to BDDs, ADDs are also built on principles of Boole's Expansion Theorem [23]. Thus, all the binary operations that are applicable to BDDs are translated to ADDs as well.

Example 4. *We assign for states in $V'' = \{v_{s2}, v_{s5}\}$ in Fig. 3a, an integer value 2 and 3, respectively. The corresponding ADD for $f(x_0, x_1, x_2) = (x_0 \wedge \neg x_1 \wedge x_2) \to 2 \quad \vee (\neg x_0 \wedge \neg x_1 \wedge \neg x_2) \to 3$ is shown in Fig. 3d.*

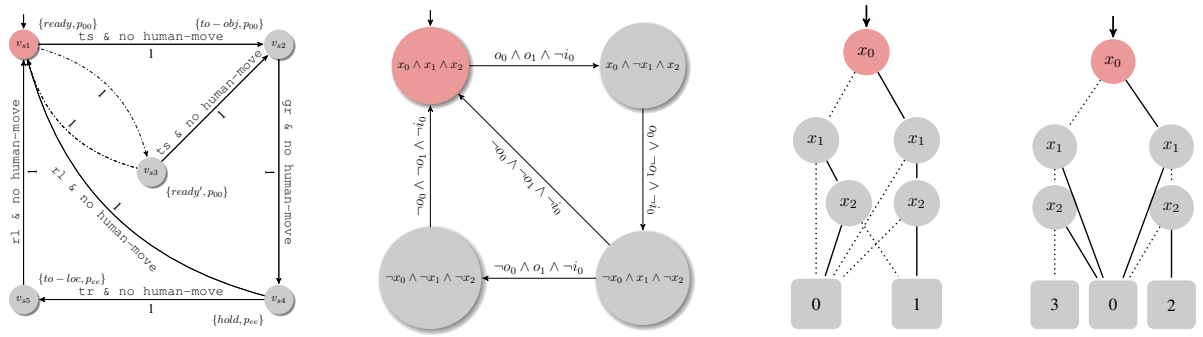
IV. Symbolic Games

To achieve a scalable and efficient synthesis framework, we base our approach on symbolic methods. Here, we introduce symbolic graphs, their construction, and our proposed algorithms to perform symbolic Value Iteration and synthesize regret-minimizing strategies in a symbolic fashion.

A. Symbolic Two-player Games

To avoid explicit construction of all the states and transitions, we use a symbolic representation that lets us reason over sets of states and edges. Usually, using this approach results in a much more concise representation than explicitly representing each state and edge. However, we need to redefine algorithms to accommodate symbolic representations.

Encoding the game graph $\mathcal{G}$ using Boolean formulas enables us to represent the states and actions using only a



(a) Explicit Two-player game abstraction (b) Symbolic Two-player game (c) BDD for set V' (d) ADD for set V''

Fig. 3: (a) and (b) illustrate explicit graph and its equivalent symbolic graph. Robot actions `transit` (ts) is $o_0 \wedge o_1$, `grasp` (gr) is $o_0 \wedge \neg o_1$, `transfer` (tr) is $\neg o_0 \wedge o_1$, and `release` (rl) is $\neg o_0 \wedge \neg o_1$. Human actions `human-move` (dashed) and `no human-move` (solid) are i_0 and $\neg i_0$, respectively. We skip `human-move` edges for simplicity. Variables $x_0, x_1 \in X$ correspond to robot configuration and $x_2 \in X$ for box being grounded (p_{00}) or manipulated (p_{ee}). (c) and (d) illustrate the BDD and ADD for Example 3 and 4.

logarithmic number of boolean variables. Further, we can use BDDs (ADDs) to compactly represent these boolean formulas and perform efficient set-based operations using operations defined over BDDs (ADDs).

Definition 4 (Compositional Symbolic Game $\mathcal{G}_s$ [1])**.** *Given a Two-player game $\mathcal{G}$, its corresponding symbolic representation is defined as tuple $\mathcal{G}_s = (X, I, O, \eta)$ where,*

- *X is the set of boolean variables such that every state $v \in V$ has corresponding boolean formula whose realization is true, i.e., $2^X \to 1$,*
- *I is the set of boolean variables such that every Human action $a_e \in A_e$ has corresponding boolean formula whose realization is true, i.e., $2^I \to 1$,*
- *O is the set of boolean variables such that every Robot action $a_s \in A_s$ has corresponding boolean formula whose realization is true, i.e., $2^O \to 1$, and*
- *transition relation $\eta = \{\eta_{x_0}, \eta_{x_1}, \ldots\}$ is a vector of boolean functions where $\eta_{x_i} : 2^X \times 2^I \times 2^O \to x_i$, i.e., $\eta_{x_i}(X, I, O)$ evaluates to true iff $x_i \in X$ is true in the corresponding successor state.*

Remark 1. *Since our interest is in synthesizing a robot strategy, we do not need to reason over human states explicitly. Hence, we abstract away the states in V_e by modifying the transitions in $\mathcal{G}$ to model the evolution from each robot state to be a function of robot action a_s and human action a_e, i.e., $\delta : V_s \times A_s \times A_e \to V_s$ for all $v \in V_s$. Fig. 3a depicts the modified abstraction for the abstraction in Fig. 2b.*

Example 5. *For each state and action in Fig 3a, we assign a corresponding boolean formula whose realization is set to true using variables $x \in X$, $\iota \in I$, and $o \in O$ as shown in Fig 3b. Further, the transition relation η from v_{s1} and v_{s4} is represented as follows. $\eta_{x_0} = \eta_{x_2} = (x_0 \wedge x_1 \wedge x_2 \wedge o_0 \wedge o_1 \wedge \neg i_0)$. $\eta'_{x_0} = \eta'_{x_1} = \eta'_{x_2} = (\neg x_0 \wedge x_1 \wedge \neg x_2 \wedge \neg o_0 \wedge \neg o_1 \wedge \neg i_0)$. Thus, $\eta_{x_0} = \eta_{x_0} \vee \eta'_{x_0}$, $\eta_{x_1} = \eta_{x_1} \vee \eta'_{x_1}$, $\eta_{x_2} = \eta_{x_2} \vee \eta'_{x_2}$, and $\eta = \{\eta_{x_0}, \eta_{x_1}, \eta_{x_2}\}$. We repeat this for all edges in $\mathcal{G}_s$.*

Definition 5 (Symbolic DFA $\mathcal{A}^s_\varphi$)**.** *Given DFA $\mathcal{A}_\varphi$, its corresponding symbolic representation is defined as tuple $\mathcal{A}^s_\varphi = (Y, X, \zeta, \mathcal{F})$, where*

- *X is as defined for $\mathcal{G}_s$ in Def 4,*
- *Y is the set of boolean variables such that every state $z \in Z$ has corresponding boolean formula whose realization is true, i.e., $2^Y \to 1$,*
- *$\zeta = \{\zeta_{y_0}, \zeta_{y_1}, \ldots\}$ is a vector of boolean functions, where $\zeta_{y_i} : 2^Y \times 2^X \to y_i$, i.e., $\zeta(X, Y)$ evaluates to true iff $y_i \in Y$ is true in the successor state,*
- *$\mathcal{F}$ is a Boolean function whose realization is true for boolean formulas corresponding $z \in Z_f$.*

Given symbolic game $\mathcal{G}_s$ and DFA $\mathcal{A}^s_\varphi$, we perform value iteration symbolically as described below.

B. Symbolic Value Iteration (VI)

Here, we first show a Value Iteration (VI) algorithm for min-max reachability games with uniform edge weights. Then, we extend the method to arbitrary edge weights. Finally, we show how this VI for reachability games can be adapted for DFA games, hence solving Problem 1.

VI with uniform edge weights: In explicit value iteration in Alg. 1, at every iteration of the loop, we need to iterate through every state $s \in S$ and apply the cPreMin and cPreMax operators (Lines 5-8). In fact, these inner loops are performed because we need to reason over a set of states at every (outer-loop) iteration. Indeed, this procedure can be naturally encoded in a symbolic fashion, where we can efficiently perform set-based operations. As value-iteration is essentially a fixed-point computation, where we start from the accepting states and back-propagate the state values, we need to define the predecessor operation to (i) construct the set of predecessors, and (ii) reason over them.

We define the operator that computes the predecessors as the PreImage operator. Alg. 2 outline this predecessor computation using ADDs. Let ω be the Boolean function representing a set of target $\mathcal{F}'$ states in $\mathcal{G}_s$, i.e., $\omega(X) : 2^X \to 1$ for states in the target set $\mathcal{F}'$. Now, to compute the predecessors, we use a technique called *variable substitution*. For each boolean variable $x \in X$ in ω, we substitute the variable x_i with its corresponding transition relation $\omega(\eta_{x_i})$. Variable substitution using ADDs can be implemented using

Algorithm 2: PreImage

Input : vector $\langle \mathsf{ADD}(w_k) \rangle$, Symbolic Game $\mathcal{G}_s$,
Output: vector of pre-images $\mathsf{ADD}(pre)$

1 $pre = \langle \emptyset \rangle$ // vector to store predecessor as 0-1 ADDs
2 **for** $\mathsf{ADD}(B_j) \in \mathsf{ADD}(w_k)$ **do**
3 **for** $\eta_{c_i} \in \eta$ **do**
4 $j' = j + c_i$
5 $\mathsf{ADD}(B_{j'}) =$ $\mathsf{ADD}(B_j).\texttt{vectorCompose}(X, \eta_{c_i})$
6 $pre_{j'} = \mathsf{ADD}(B_{j'})$
7 **return** $\mathsf{ADD}(pre)$

the Compose operator shown on Line 5 in Alg. 2. Substituting $x_i \leftarrow \eta_{x_i}$ in ω captures the value of x_i in the predecessor states for all valid actions in η_{x_i}. We repeat the process for all $x_i \in X$ and store it as pre. Thus, $pre(X, I, O)$ is the new Boolean function such that $2^X \times 2^I \times 2^O \rightarrow 1$ for the set of valid edges from states in pre to a state in ω.

To compute the set of winning states for a reachability game with no quantities, we perform the following procedures. First, we perform universal quantification. This preserves all the states that can *force* a visit to the next state. Then we existentially quantify the boolean function to compute set of states that can force a visit in one-time step. Mathematically, $\omega_1 = \exists O \cdot (\forall I \cdot pre(X, I, O))$ and we repeat this process till we reach a fixed point, i.e., $\omega_{k-1} \equiv \omega_k$.

To compute the winning strategies, we initialize an additional boolean function t as $t(X, O)_0 = \omega_0$. After each fixed-point computation, $t(X, O)_{k+1} = \forall I \cdot pre(X, I, O)_k$. For quantitative reachability games, we replace the universal operation with `max` and the existential operation with `min`. Thus, for a graph with a uniform edge weights, this method suffices. Next, we now discuss how to perform VI over $\mathcal{G}_s$.

VI with arbitrary edge weights: As discussed above, we start with ω_0. But instead of storing it in a monolithic boolean function, we decompose it into smaller boolean functions, each corresponding to the value associated with the states. Mathematically, $\omega_k = \langle B_j \rangle$ where $\langle B_j \rangle$ is a sequence of boolean functions, each representing the set of states with value j. For the initial iteration, $k = 0$, $j = 0$, and $B_j \equiv \omega_0$ is the Boolean function for $\mathcal{F}'$. To compute the predecessor, we first segregate the transition relation η according to their edge weights, i.e., $\eta = \{\eta_{c_1}, \eta_{c_2}, \ldots\}$ where c_1, c_2 are edge weights associated with edges in $\mathcal{G}$.

Definition 6 (Monlithic Transition Relation)**.** *A Monlithic Transition Relation for $\mathcal{G}_s$ is defined as $\eta = \{\eta_{c_i}(X, I, O)\}$ where $\eta_{c_i} = \{\eta_x | x \in X\}$ is a sequence of boolean functions. Each boolean function is segregated based on the set of all edge weights corresponding to A_s. Further, an edge appears only once in η, i.e., $\eta_{c_i}(\eta_{x_i}) \cap \eta_{c_j}(\eta_{x_i}) = \emptyset$ for every $i \neq j$.*

Definition 7 (Partitioned Transition Relation)**.** *A Partitioned Transition Relation for $\mathcal{G}_s$ is defined as $\eta = \{\eta_{c_i}(X, I, O)\}$ where $\eta_{c_i} = \{\eta_x | x \in X\}$ is a sequence of boolean functions. Each boolean function segregated based on valid robot actions $a_s \in A_s$ only, i.e., $|\eta| = |A_s|$.*

Algorithm 3: Symbolic Value Iteration

Input : Symbolic Two-player Game G_s
Output: Winning strategy $\mathsf{ADD}(\sigma)$, Optimal values $\mathsf{ADD}(\omega_k)$

1 $\mathsf{ADD}(\sigma) \leftarrow$ Initialize ∞ ADD
2 $\mathsf{ADD}(\tau) \leftarrow$ Initialize 0 ADD
3 $\mathsf{ADD}(\omega_0), \mathsf{ADD}(\mathcal{F}') \leftarrow$ Init 0-∞ ADD for target states
4 $\mathsf{ADD}(\omega_0) = \mathsf{ADD}(\omega_0).\texttt{min}(\mathsf{ADD}(\mathcal{F}'))$
5 $\mathsf{ADD}(\sigma) = \mathsf{ADD}(\sigma).\texttt{min}(\mathsf{ADD}(\omega_0))$
6 **while** $\mathsf{ADD}(\omega_{k-1}) \not\equiv \mathsf{ADD}(\omega_k)$ **do**
7 $\langle \mathsf{ADD}(B_j) \rangle \leftarrow$ Construct 0-1 ADDs from $\mathsf{ADD}(\omega_k)$
8 $\mathsf{ADD}(pre) \leftarrow \texttt{preImage}(\langle \mathsf{ADD}(B_j) \rangle, G_s)$
9 $\mathsf{ADD}(\mathsf{aPre}) \leftarrow$ Initialize 0 ADD
10 **for** $\mathsf{ADD}(B_j) \in \mathsf{ADD}(pre)$ **do**
11 $\mathsf{ADD}(\mathsf{aPre}) = \mathsf{ADD}(\mathsf{aPre}).\texttt{max}(\mathsf{ADD}(B_j))$
12 **for** $f \in 2^I \rightarrow 1$ **do**
13 $\mathsf{ADD}(\tau) =$ $\mathsf{ADD}(\tau).\texttt{max}(\mathsf{ADD}(\mathsf{aPre}).\texttt{restrict}(f))$
14 $\mathsf{ADD}(\sigma) = \mathsf{ADD}(\sigma).\texttt{min}(\mathsf{ADD}(\tau))$
15 **for** $f \in 2^O \rightarrow 1$ **do**
16 $\mathsf{ADD}(\omega_{k+1}) =$ $\mathsf{ADD}(\omega_k).\texttt{min}(\mathsf{ADD}(\sigma).\texttt{restrict}(f))$
17 **return** $\mathsf{ADD}(\omega_k)$, $\mathsf{ADD}(\sigma)$

Thus, for each η_{c_i} and for each $B_j \in \omega_k$, we compute the PreImage of B_j and store them as $\omega_{k+1} = \langle B_{j'} \rangle$ where $j' = j + c_i$. If $B_{j'}$ already exists, we take the union of the boolean functions. Alg. 2 summarizes the pre-image computation over weighted edges. For strategy synthesis, we repeat the same process as above until $\omega_{k+1} \equiv \omega_k$, i.e., every boolean function $\mathcal{B}_j$ for $k+1$th iteration is $\equiv B_j$ in kth iteration. Alg. 3 gives the pseudocode for symbolic value iteration with quantitative constraints. For efficient manipulation of B_j, we use ADDs. We use the Compose operation for substitution, and `min` and `max` for algebraic computations. To check for convergence, we exploit the canonical nature of ADDs for constant time equivalence checking.

C. Strategy Synthesis for symbolic DFA Games

To synthesize symbolic winning states and strategies for the DFA game $\mathcal{P}$, we compose the PreImage computation as described above to evolve over $\mathcal{G}_s$ and $\mathcal{A}^s_\varphi$ in an asynchronous fashion. Given a Boolean function representing the set of accepting states such that $\omega_0(X, Y) = 2^X \times 2^Y \rightarrow 1$ ($2^X \rightarrow 1 \quad \forall v \in V$ and $2^Y \rightarrow 1 \quad \forall z \in Z_f$), we first compute the predecessors over the states of $\mathcal{A}^s_\varphi$ by evolving over $y \in Y$, i.e., $pre_\varphi(X, Y) \leftarrow$ PreImage of $\omega_0(X, Y)$. This intermediate Boolean function captures only the evolution of the DFA variable, i.e., $2^X \times 2^Y \rightarrow 1$ such that $2^Y \rightarrow 1$ for the valid DFA states in the predecessors. Finally, we evolve over the symbolic game $\mathcal{G}_s$ from the intermediate Boolean function, i.e., $pre(X, Y, I, O) \leftarrow$ PreImage of $pre_\varphi(X, Y)$, to compute the Boolean function representing valid predecessors in $\mathcal{G}_s$. Algorithmically, we modify Alg. 2 with the intermediate pre_φ computation and keep the pre computation consistent for evolution over pre_φ.

We employ the above algorithm to efficiently synthesize symbolic winning strategies for Problem 1. Our evaluations illustrate that this symbolic algorithm has an order of magnitude speedup over the explicit approach. For Problem 2, previous work [2], [24] provide algorithms for synthesizing regret-minimizing strategies for explicit graphs. Next, we discuss our efficient symbolic approach to this problem and synthesis of regret-minimizing strategies.

V. Hybrid Regret synthesis algorithm

The first step in synthesizing regret-minimizing strategies is to construct the Graph of Utility $\mathcal{G}^u$ [2]. Intuitively, this graph captures all possible plays (unrolling) over $\mathcal{P}$ along with the total payoff associated with each path to reach a state in $\mathcal{P}$. In the explicit approach, we first construct the set of payoffs as $[B] = \{0, 1, 2, \ldots, \mathcal{B}\}$ where $\mathcal{B}$ is the user-defined budget. We then take the Cartesian product $\mathcal{P} \times [B]$ to construct states (s, u) of $\mathcal{G}^u$. An edge $(s, u) \rightarrow (s', u')$ in $\mathcal{G}^u$ exists iff (s, a, s') is a valid edge in $\mathcal{P}$ and $u' = u + F(s, a, s')$. As constructing $\mathcal{G}^u$ is Pseduo-polynomial, this computation is memory intensive.

To construct $\mathcal{G}^u$ efficiently, we define $\mathcal{G}^u_s$ as the symbolic Graph of Utility. To construct the Transition Relation, we create an additional set of boolean variables $u \in U$ such that $2^U \rightarrow 1$ for valid utilities in $[B]$. We then create the Transition relation η_u for the utility variables to model their evolution symbolically under each action A_s. Mathematically, $\eta_u = \{\eta_{u_1}, \eta_{u_2}, \ldots, \eta_{u_n}\}$ where $\eta_{u_i} = \{\eta_u | u \in U\} \forall i \in n$ and $|\eta_u| = |\eta|$. Thus, to perform $\mathrm{PreImage}$ operation over $\mathcal{G}^u_s$, we substitute variables $x_i \in X$ and $u_i \in U$ in a synchronous fashion using the $\mathrm{Compose}$ operator.

The second step in regret-based synthesis is to construct the graph of the *best response* $\mathcal{G}^{br}$. This graph captures the best alternate payoff ba from every edge along each path on $\mathcal{G}^u$, i.e., $\mathtt{min}_{\sigma'} \mathrm{Val}(\sigma', \tau)$ in (2). For a fixed strategy σ, we compute $ba = \mathtt{min}_{\sigma' \in \Sigma^{\mathcal{G}} \setminus \sigma} \mathtt{min}_{\tau \in \mathrm{T}^{\mathcal{G}}} \mathrm{Val}^{(s,u)}(\sigma, \tau)$ assuming human to be cooperative. Finally, we repeat the process for every edge in $\mathcal{G}^u_s$ to construct the set of ba.

For symbolic implementation, we first compute cooperative values by first playing min-min reachability game and then "explicitly" loop over every edge in $\mathcal{G}^u_s$ to construct the Symbolic Transition Relation for symbolic graph of best response $\mathcal{G}^{br}_s$. While the computation of the ba set is explicit, the resulting Transition Relation for $\mathcal{G}^{br}_s$ is purely symbolic. Finally, we play a min-max reachability game using method described above to compute regret-minimizing strategies over $\mathcal{G}^{br}_s$. Our empirical results show that using this approach scales to larger manipulation problems with significantly lower memory consumption than the explicit approach.

VI. Experiments

We consider various scenarios for our pick-and-place manipulation domain. We benchmark the explicit approach with the symbolic approaches from Section IV for min-max reachability games in Problem 1 and regret-minimizing strategy synthesis for Problem 2 with varying input parameters, i.e., energy budget $\mathcal{B}$, number of locations $|L|$, and number of boxes $|O|$. The task for the robot in all scenarios is to place boxes in their desired locations.

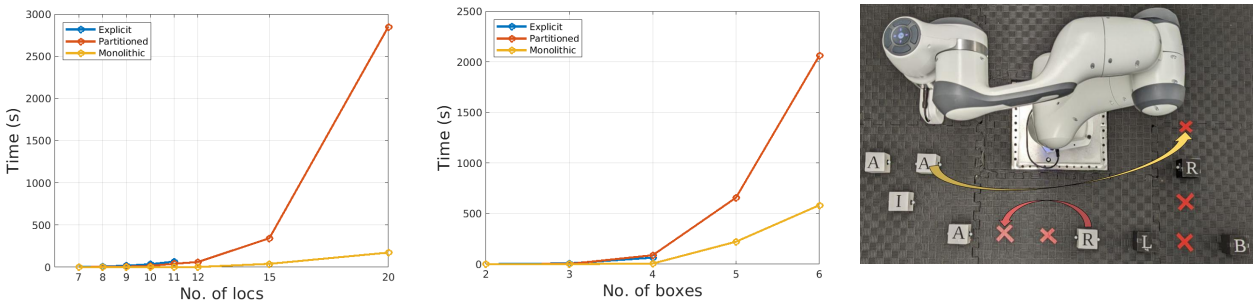
The robot spends 3 units of energy for every action to operate away from the human and one unit of energy when operating near the human. Refer to [2] for the experimental setup details for Problem 2 benchmarks. All the experiments are run single-threaded on an Intel i5 -13th Gen 3.5 GHz CPU with 32 GB RAM. The source code is in Python, and we use a Cython-based wrapper for the CUDD package for ADD operations [25], and MONA for LTLf to DFA translation [26]. The tool is available on GitHub [27]. Finally, we demonstrate our symbolic synthesis algorithm on a 5 objects scenario.

Min-Max with Quantitative Constraints: For problem 1, we benchmark computation time for synthesizing winning strategies using the Explicit, Monolithic, and Partitioned Symbolic approaches. For Scenario 1, we fix $|O| = 3$ and vary $|L|$. In Scenario 2, we fixed $|L| = 8$ and vary $|O|$. Results are shown in Fig. 4. For the Monolithic approach, we see at least an order of magnitude speed-up over the Explicit approach for both scenarios. Further, there is almost an order of magnitude speedup for the Monolithic approach over the Partitioned approach for Scenario 1. We note that the Partitioned approach is slower than the Monolithic approach due to the size of the Monolithic ADD not being large enough to compensate for the computational gain from the $\mathrm{PreImage}$ operation over multiple smaller ADDs. Hence, the cumulative time spent computing predecessors is much more in the Partitioned approach for every fixpoint iteration than in the Monolithic approach. For the Explicit approach, we could not scale beyond 4 boxes and 8 locations due to the memory required to store $\mathcal{G}_s$ explicitly ($>$ 15 GB).

Regret-Minimizing Strategy Synthesis: For problem 2, we consider three scenarios. In Scenario 1, we fix $|O|$ and $|L|$ and vary $\mathcal{B}$. For the Scenario 2, we fix $|O|$ and $\mathcal{B}$ and increase $|L|$. Finally, for Scenario 3, we fix $|L|$ and vary $\mathcal{B}$ and $|O|$. Note, for Scenario 3, the regret budget $\mathcal{B}$ is chosen to be 1.25 times the minimum budget from Problem 1. Results are shown in Fig. 5. For all the scenarios, we observe that the explicit approach runs out of memory for relatively smaller instances. Additionally, we observe that overall the Monolithic approach is faster than the Partitioned approach due to the size of the Transition relation, as mentioned above.

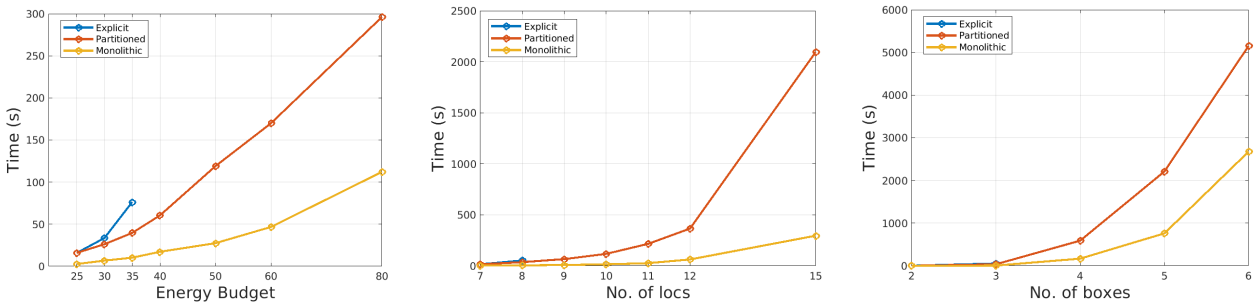
Scalability on the Budget: Unrolling the explicit graph is a computationally intensive procedure as the state space grows linearly with budget $\mathcal{B}$. Thus, we observe in Fig. 5a that the explicit approach runs out of memory for $\mathcal{B} > 35$. There are $62,800$ explicit nodes for $\mathcal{B} = 35$ and $295,496$ explicit nodes for $\mathcal{B} = 80$ in $\mathcal{G}^{br}_s$. We required maximum of 860 MB for the symbolic approaches and more than 9 GB for the explicit approach with $\mathcal{B} = 35$.

Scalability on the number of locations: For this scenario, we observe that the Monolith approach is at least 3 times faster than the Partitioned approach. This is because, as we increase the number of locations, we do not necessarily need more boolean variables. The gap widens as we keep increasing $|L|$. For $|L| = 20$, the graph has 7×10^6 nodes and



(a) $|O| = 3$ and variable $|L|$ (b) $|L| = 8$ and variable $|O|$ (c) Regret-minimizing strategy

Fig. 4: (a) and (b) Benchmark results for min-max synthesis. (c) Regret-minimizing strategy for task in Fig. 1.



(a) $|O| = 3$, $|L| = 7$, variable $\mathcal{B}$ (b) $|O| = 3$, $\mathcal{B} = 25$, variable $|L|$ (c) $|L| = 8$, variable $\mathcal{B}$ & $|O|$

Fig. 5: Benchmark results for regret-minimizing strategy synthesis in Problem 2

55×10^6 edges in $\mathcal{G}_s^{br}$. The maximum memory consumption is 1040 MB as opposed to over 5 GB memory for $|L| = 8$.

Scalability on the number of objects: For Scenario 3, we observe that the explicit approach does not scale beyond 3 objects. The computation times for the symbolic approach also increase exponentially due to the fact that every object added to the problem instance needs an additional set of boolean variables. For $|O| = \{2, 3, 4, 5, 6\}$, we needed 29, 36, 40, 45, and 49 boolean variables with a maximum of 1100 MB consumed. The explicit approach required above 15 GB of memory for $|O| = 4$.

Physical Execution: We demonstrate the regret-minimizing strategy for building "ARIA LAB" in Fig. 1. Figs. 4c and 1 show the initial and final configuration for $|O| = 5$ (white blocks) and $|L| = 7$. Robot and human cannot manipulate black blocks. We note that the human can only move the blocks in its region to its left. The robot initially operates away from the human in the robot region (yellow arrow in Fig. 4c). The human moves "R" (red arrow) cooperatively and allows the robot finish the task in its region (Fig. 1b).

Discussion: We show through our empirical evaluations that symbolic approaches can successfully scale to larger problem instances with significant memory savings. We observe that using a monolithic representation of the transition relation is faster than the Partitioned approach due to the size of ADDs. We note that while this paper focused on boolean representation for our problem, other ADD based optimizations like variable ordering can be utilized for a more compact representation of ADDs which could lead to additional computational speedups. Finally, we note that we compute the set of reachable states on the symbolic graph of utility before Value Iteration. This allows the synthesis algorithm to only reason over the set of reachable states and thus speed up the synthesis process.

VII. Conclusion

Symbolic graphs and algorithms are powerful tools for reasoning over systems with large state spaces. We provide fundamental algorithms for symbolic value iteration using ADDs. Further, we compare two of the most commonly used Transition Relation representations. Finally, we benchmark our implementation for robotic manipulation scenarios and show that we are able to achieve significant speedups. We discuss our findings and summarize which Transition relation representation is more suitable for robotic applications.

References

[1] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Int. Joint Conf. on Artificial Intelligence*, ser. IJCAI '13. AAAI Press, 2013, p. 854–860.

[2] K. Muvvala, P. Amorese, and M. Lahijanian, "Let's collaborate: Regret-based reactive synthesis for robotic manipulation," in *International Conference on Robotics and Automation*, 2022, pp. 4340–4346.

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

[4] 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.

[5] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annual Review of Control, Robot., and Auto. Sys.*, vol. 1, pp. 211–236, 2018.

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

[7] P. Hunter, G. A. Pérez, and J.-F. Raskin, "Reactive synthesis without regret," *Acta informatica*, vol. 54, no. 1, pp. 3–39, 2017.

[8] K. He, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Automated abstraction of manipulation domains for cost-based reactive synthesis," *IEEE Robotics and Automation Letters*, vol. 4, pp. 285–292, 2019.

[9] A. M. Wells, Z. Kingston, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "Finite-horizon synthesis for probabilistic manipulation domains," in *IEEE International Conference on Robotics and Automation*. IEEE, 2021.

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

[11] S. Zhu, L. M. Tabajara, J. Li, G. Pu, and M. Y. Vardi, "Symbolic ltlf synthesis," *arXiv preprint arXiv:1705.08426*, 2017.

[12] K. Chatterjee, M. Randour, and J.-F. Raskin, "Strategy synthesis for multi-dimensional quantitative objectives," *Acta informatica*, vol. 51, no. 3-4, pp. 129–163, 2014.

[13] S. Maoz, O. Pistiner, and J. O. Ringert, "Symbolic bdd and add algorithms for energy games," *arXiv preprint arXiv:1611.07622*, 2016.

[14] H. Gimbert and W. Zielonka, "When can you play positionally?" in *Math. Found. of Comp. Sci.* Springer, 2004, pp. 686–697.

[15] T. Brihaye, G. Geeraerts, A. Haddad, and B. Monmege, "Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games," *Acta Informatica*, vol. 54, pp. 85–125, 2017.

[16] F. Somenzi, "Binary decision diagrams," *NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES*, vol. 173, pp. 303–368, 1999.

[17] R. E. Bryant, "Binary decision diagrams and beyond: Enabling technologies for formal verification," in *Int. Conf. on Computer Aided Design*. IEEE, 1995, pp. 236–243.

[18] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Information and computation*, vol. 98, no. 2, pp. 142–170, 1992.

[19] R. Bloem, K. Chatterjee, and B. Jobstmann, "Graph games and reactive synthesis," in *Handbook of Model Checking*. Springer, 2018, pp. 921–962.

[20] C. Baier and J.-P. Katoen, *Principles of model checking*, 2008.

[21] S. Chaki and A. Gurfinkel, "Bdd-based symbolic model checking," *Handbook of Model Checking*, pp. 219–245, 2018.

[22] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebric decision diagrams and their applications," *Formal methods in Sys. design*, vol. 10, 1997.

[23] G. Boole, *The mathematical analysis of logic*. Philosophical Library, 1847.

[24] E. Filiot, T. Le Gall, and J.-F. Raskin, "Iterated regret minimization in game graphs," in *International Symposium on Mathematical Foundations of Computer Science*. Springer, 2010, pp. 342–354.

[25] F. Somenzi, "Cudd: Cu decision diagram package 3.1.0," *University of Colorado at Boulder*, 2020.

[26] J. G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm, "Mona: Monadic second-order logic in practice," in *Tools and Alg. for the Const. and Anal. of Sys.* Springer, 1995, pp. 89–110.

[27] K. Muvvala, "Symbolic reactive synthesis with quantitative objectives," https://github.com/MuvvalaKaran/symbolic_synthesis.