2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Verifiable Learned Behaviors via Motion Primitive Composition: Applications to Scooping of Granular Media

Andrew Benton, Eugen Solowjow, Prithvi Akella[1]

***Abstract*— A robotic behavior model that can reliably generate behaviors from natural language inputs in real time would substantially expedite the adoption of industrial robots due to enhanced system flexibility. To facilitate these efforts, we construct a framework in which learned behaviors, created by a natural language abstractor, are verifiable by construction. Leveraging recent advancements in motion primitives and probabilistic verification, we construct a natural-language behavior abstractor that generates behaviors by synthesizing a directed graph over the provided motion primitives. If these component motion primitives are constructed according to the criteria we specify, the resulting behaviors are probabilistically verifiable. We demonstrate this verifiable behavior generation capacity in both simulation on an exploration task and on hardware with a robot scooping granular media.**

I. Introduction

In recent years, learning from human demonstrations has proven tremendously successful at imitating intricate, human-like motion on robotic systems [1]–[3]. This has allowed for improvements in robotic grasping [4]–[6], assembly [3], [7], [8], and even robotic surgery [9]–[11]. However, these methods often require prohibitive amounts of precisely labeled data [12]. Additionally, these learned behaviors are typically not transferrable to tasks that are similar but not identical, prompting further research into task-transferrable learning [13]–[15]. However, works in this vein exhibit similar, if not heightened, requirements on the amount of data available to the learning procedure.

Despite these challenges, more comprehensive learned models that incorporate streams of multimodal data have shown tremendous success at learning generalized, intricate behaviors. For example, the recently developed Palm-E model has successfully translated natural language user commands to control policies for a 6-DOF arm, realizing the intended tasks even when they were not explicitly learned [16]. Building on the success of Palm-E and other foundational robotic models [17]–[19], recent work also aims to codify effective design principles for these models [20].

Conceptually, however, both the Palm-E model and the other learning paradigms mentioned prior hinge on a notion of composing generalized behavior from a finite set of learned behaviors. Prior work in controls and robotics has shown that generalizing from this initial behavior set, termed motion primitives in the existing literature, yields robust, and more importantly, verifiable generalized behavior provided the primitives and subsequent behaviors are constructed with care [21]–[23]. Consequently, inspired by the previous attempts at codifying design principles for these learned models [20], we posit that by leveraging these prior works in motion primitives and black-box risk-aware verification, we can synthesize verifiable learned behaviors over a provided set of carefully constructed motion primitives.

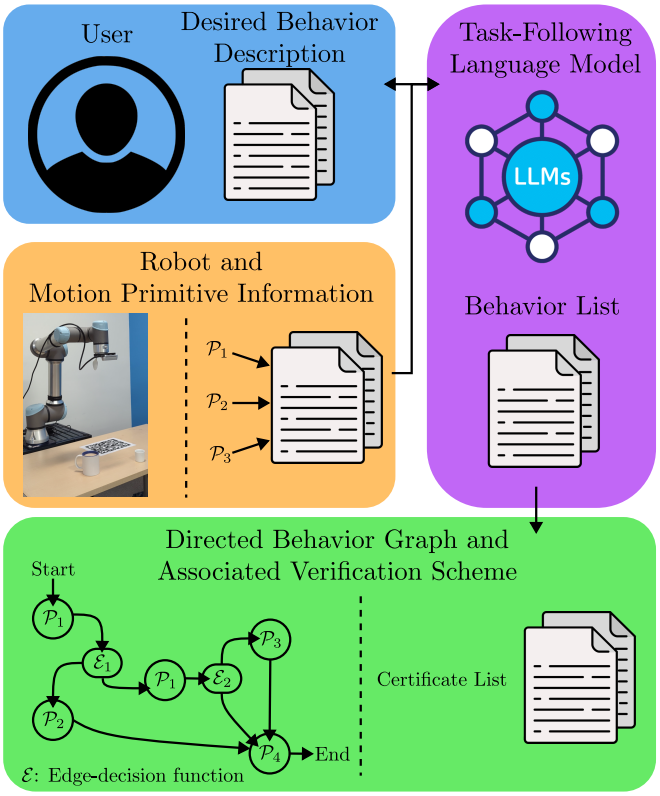

Fig. 1. A graphical representation of our natural-language-based behavior generalizer and verification scheme. By ensuring that the language model only composes behaviors as a directed graphical abstraction over the provided motion primitives, we show that any such generated behavior has an associated certificate list that we can exploit to verify the learned behavior's ability to realize the user's desired task.

Our Contribution: Leveraging recent work in risk-aware verification [24], [25], we take steps towards constructing a framework for verifying learned, generalized behaviors composed from a set of motion primitives. Specifically, if the input/output spaces of the motion primitives satisfy certain conditions that permit its verifiability, and the behavior is constructed as a directed graph over these primitives, then the resulting behavior is similarly verifiable. We showcase this verifiability in both simulation and on hardware, focusing on exploration and reconnaissance for the former and a granular media scooping task for the latter.

Structure: We review black-box risk-aware verification and

[1]All authors are with Siemens Corporation {andrew.benton, prithvi.akella, eugen.solowjow}@siemens.com

motion primitives in Section II before formally stating the problem under study in Section II-C. Section III details our behavior generation scheme and states our main contribution regarding the verifiability of the resulting generated behaviors. Finally, Section IV showcases our behavior generation scheme developing an exploratory behavior - Section IV-A - and a scooping motion for granular media - Section IV-B. Both behaviors are also verified in the same sections according to the provided verification scheme.

II. Terminology and Formal Problem Statement

A. Black-Box Risk-Aware Verification

The information in this section is adapted from [24], [25]. Black-box risk-aware verification assumes the existence of a discrete-time controlled system of the following form, with system state $x \in \mathcal{X}$, control input $u \in \mathcal{U}$, environment state $d \in \mathcal{D}$ and potentially unknown dynamics f:

$$x_{k+1} = f(x_k, u_k, d),\ \forall\ k = 0, 1, 2, \dots. \tag{1}$$

As verification measures the robustness of a controlled system's ability to realize a behavior of interest, work in this vein assumes the existence of a feedback controller $U : \mathcal{X} \times \mathcal{D} \to \mathcal{U}$. The system's evolution when steered by this controller U will be denoted as Σ - a function mapping an initial system and environment state to the system state evolution as prescribed by (1), *i.e.* $\Sigma(x_0, d) = \{x_0, x_1, \dots, x_K\} \in \mathcal{X}^K$ for some $K > 0$. Finally, a robustness measure ρ maps this state evolution $\Sigma(x_0, d)$ and environment state d to the reals, *i.e.* $\rho : \mathcal{X}^K \times \mathcal{D} \to \mathbb{R}$. For context, these robustness measures can be those coming from temporal logic [26] or the minimum value of a control barrier function over a time horizon [27] among other methods. A positive outcome of this robustness measure indicates that the corresponding state evolution realized the desired behavior, *i.e.* $\rho(\Sigma(x_0, d), d) \geq 0$ implies the state evolution $\Sigma(x_0, d)$ realized the behavior of interest.

Black-box risk-aware verification employs this robustness measure to provide a probabilistic statement on the system's ability to realize the desired behavior for all permissible initial conditions and environment states. This will formally be expressed in the following theorem:

Theorem 1. *Let $\{r^i = \rho(\Sigma(x_0^i, d^i), d^i)\}_{i=1}^N$ be a set of N robustness evaluations of trajectories whose initial conditions and environments (x_0^i, d^i) were sampled via π over $\mathcal{X} \times \mathcal{D}$, and let $r^* = \min\{r_1, r_2, \dots, r_N\}$. Then, both the probability of sampling an initial condition and environment evolution pair whose robustness is lower bounded by r^* and the confidence in the associated probability is only a function of the number of samples N and a scalar $\epsilon \in [0, 1]$, i.e.*

$$\mathbb{P}_\pi^N\left[\mathbb{P}_\pi[\rho(\Sigma(x_0, d), d) \geq r^*] \geq 1 - \epsilon\right] \geq 1 - (1 - \epsilon)^N.$$

B. Motion Primitives

Motion primitives are a well-studied field in the controls and robotics literature, though we will provide a slight variant on existing definitions to align with our notation.

Definition 1. A *Motion Primitive* is 4-tuple $\mathcal{P} = (\Xi, A, U, R)$ with the following definitions for the tuple:

- (Ξ) The complete set of parameters for this primitive, *i.e.* $\Xi \subseteq \mathbb{R}^p$ for an appropriate dimension $p \geq 0$.
- (A) A function taking a system and environment state (x, d) as per (1) and outputting the subset of valid parameters P for this pair, *i.e.* $A(x, d) = P \subseteq \Xi$.
- (U) The parameterized controller for this primitive, mapping states, environments, and the parameter to inputs, *i.e.* $U : \mathcal{X} \times \mathcal{D} \times \Xi \to \mathcal{U}$.
- (R) A parameterized function outputting the subset of the state space the system will occupy upon completion of the primitive, *i.e.* for $\xi \in \Xi$ and with environment state d, $R(\xi, d) = X_f \subseteq \mathcal{X}$.

As an example consistent with the simulations to follow then, consider the system as per (1) to be a single-integrator system on the plane required to navigate in a finite-sized grid. A feasible motion primitive $\mathcal{P}$ would be moving the system to an adjacent cell. For simplicity's sake, assume there are no obstacles, and as such, the environment state space $\mathcal{D} = \varnothing$. Then, the complete set of parameters Ξ would be the labels for all the cells in this grid, the accepting function A outputs all adjacent cells to the cell containing the current system state x, U could be a proportional controller sending the system to the appropriate grid, and R would output the subset of the state space encompassed by the cell to which the system was required to move.

C. Problem Statement

Our goal is to develop a framework by which behaviors learned over these primitives can be verified. As such, we define a behavior B as a directed graph of primitives, with edges from a primitive $\mathcal{P}$ indicating the primitive $\mathcal{P}'$ to be run upon completion of $\mathcal{P}$. For examples of such behaviors, see the sketch provided in Figure 1 and the resulting behavior for our simulation example in Figure 3. The formal definition of these behaviors will follow.

Definition 2. A *behavior* B is a directed graph defined as a 4-tuple, *i.e.* $B = (N, E, S, T)$ with the following definitions:

- (N) The finite set of nodes for the graph, where each node is a primitive as per Definition 1, *i.e.* $N = \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_{|N|}\}$.
- (E) The set of directed edges connecting nodes in the graph. Each edge identifies a method to choose parameters for the successive primitive. If multiple edges emanate from a node, then a method exists such that at runtime, only one edge is chosen.
- (S) A start function taking as input the system and environment state (x, d) as per (1) and outputting both the starting primitive and its parameter, *i.e.* $S(x, d) = (\xi, \mathcal{P})$ where $\mathcal{P} \in N$ and $\xi \in A_{\mathcal{P}}(x, d)$.
- (T) The set of terminal nodes, *i.e.* $T \subseteq N$.

Our goals are twofold. First, determine whether we can verify the behaviors generated by Algorithm 1, and second, if the behaviors are verifiable, determine a framework by

Algorithm 1 Natural Language-based Behavior Generalizer

Require: A set of primitives as per Definition 1 and their descriptions $\mathbb{D} = \{(\mathcal{P}_i, \text{description of primitive } i)\}_{i=1}^{M}$, a list of existing behaviors $\mathbb{B} = \{B_1, B_2, \dots\}$ with behaviors B as per Definition 2, and a natural language abstractor A taking as input a string s defining a desired behavior, a string I defining any useful, non-primitive information available for behavior generation, and the primitive list $\mathbb{D}$ and outputting behaviors B.

```
while True do
    c ← desired behavior B
    if c ∉ 𝔹 then
        s ← description of desired behavior
        I ← helpful non-primitive information
        𝔹 ← 𝔹 ⋃ A(s, I)
    end if
end while
```

which we can verify any behavior generated by this method. Phrased formally, the problem statement will follow.

Problem 1. *Determine if the behaviors generated by Algorithm 1 are verifiable, and if they are verifiable, determine a method to verify any such generated behavior.*

III. Verifying Learned Behaviors

We will provide a solution to both aspects of Problem 1 simultaneously, by constructing the framework for verifying any behavior as per Definition 2. To construct this framework, we first note that there exist two outcomes to executing any behavior from any initial system and environment state - it either terminates successfully or it does not. In the event it terminates successfully, we can record the set of all primitives run over the course of the behavior, their corresponding parameters, and the system states upon termination of the corresponding primitive, *i.e.* $\mathbb{D} = \{(\xi_1, \mathcal{P}_1, x_1^f), (\xi_2, \mathcal{P}_2, x_2^f), \dots\}$. If the behavior fails due to reasons such as an intermediary controller failure or an error in the behavior's graph construction leading to a runtime error, we can record the failure.

This permits us to construct a robustness measure for a verification scheme aligned with the method described in Section II-A. First, for each pair in the dataset $\mathbb{D}$ generated by running the behavior, we can define a certificate function checking whether the terminal state laid in the terminal set prescribed by the primitive, parameter, and environment:

$$C\left(\xi, \mathcal{P}, x^f, d\right) = x^f \in R_{\mathcal{P}}(\xi, d). \tag{2}$$

Here, we note that we are implicitly associating boolean outcomes with ± 1. The robustness measure ρ would check the validity of each of these certificates over the run of a behavior and output 1 if all certificates were satisfied and -1 if the system failed or any certificate was not satisfied. Specifically then, let (x_0, d) be the initial system and environment state, let Σ be the trajectory function as described in Section II-A, and let $\mathbb{D}$ be the dataset of tuples collected over the course of a successfully run behavior. Then the robustness measure

$$\rho_B(\Sigma(x_0, d), d) = \begin{cases} \min\limits_{\gamma \in \mathbb{D}} C(\gamma, d) & \text{if behavior finished,} \\ -1 & \text{else.} \end{cases} \tag{3}$$

Here, we have abbreviated the tuples in $\mathbb{D}$ with the variable γ to ease notation. That being said, the robustness measure ρ_B in (3) evaluates to a positive number if and only if the behavior successfully terminated and all component primitives exhibited their component desired behaviors.

Using the robustness measure in (3), we can verify any behavior as per Definition 2. To ease the formal exposition of the results, we will first denote via $\mathcal{B}$ the subset of the system and environment state spaces that have a valid starting point for the behavior B to be verified. This is to ensure that in the verification procedure to follow, we do not sample and evaluate the behavior's performance from initial conditions and environment states that disallow the behavior from the start. Formally then,

$$\mathcal{B} = \{(x, d) \in \mathcal{X} \times \mathcal{D} \mid S_B(x, d) \neq \varnothing\}. \tag{4}$$

With these definitions we have the following theorem identifying a framework to verify behaviors, though to simplify exposition, we will express the assumptions separately:

Assumption 1. Let $\{r^i = \rho_B(\Sigma(x_0^i, d^i), d^i)\}_{i=1}^{N}$ be the behavioral robustness of N attempts at executing behavior B from uniformly sampled initial conditions and states (x_0, d) over the allowable space $\mathcal{B}$ as per (4) with robustness measure ρ as per (3), and let $r^* = \min_i r^i$.

Theorem 2. *Let Assumption 1 hold. If $r^* = 1$, then $\forall\ \epsilon \in [0, 1]$, the behavior B will execute successfully for at least $100(1-\epsilon)\%$ of the initial condition and environment pairs in $\mathcal{B}$ and the confidence in this statement is $1 - (1-\epsilon)^N$.*

Proof: As Assumption 1 satisfies the conditions for Theorem 1, we can employ the same theorem and get the following result $\forall\ \epsilon \in [0, 1]$ and substituting $r^* = 1$:

$$\mathbb{C}1 \triangleq \mathbb{P}_{\mathrm{U}[\mathcal{B}]}[\rho_B(\Sigma(x_0, d), d) \geq 1] \geq 1 - \epsilon,$$
$$\mathbb{C}2 \triangleq \mathbb{P}^N_{\mathrm{U}[\mathcal{B}]}[\mathbb{C}1] \geq 1 - (1-\epsilon)^N.$$

Here, $\mathrm{U}[\mathcal{B}]$ denotes the uniform distribution over $\mathcal{B}$. We will analyze $\mathbb{C}1$ first. Note that in order for $\rho_B(\Sigma(x_0, d), d) \geq 1$, all certificate functions over the dataset $\mathbb{D}$ generated by running behavior B must evaluate to 1 - a consequence of equations (3) and (2). As a result,

$$\rho_B(\Sigma(x_0, d), d) = 1 \iff \text{The behavior executes successfully.}$$

Therefore, we can define a subset of the feasible joint state space, corresponding to initial conditions and environment states where from and in the behavior executes successfully:

$$\mathbb{V} = \{(x, d) \in \mathcal{B} \mid \rho(\Sigma(x, d), d) = 1\}.$$

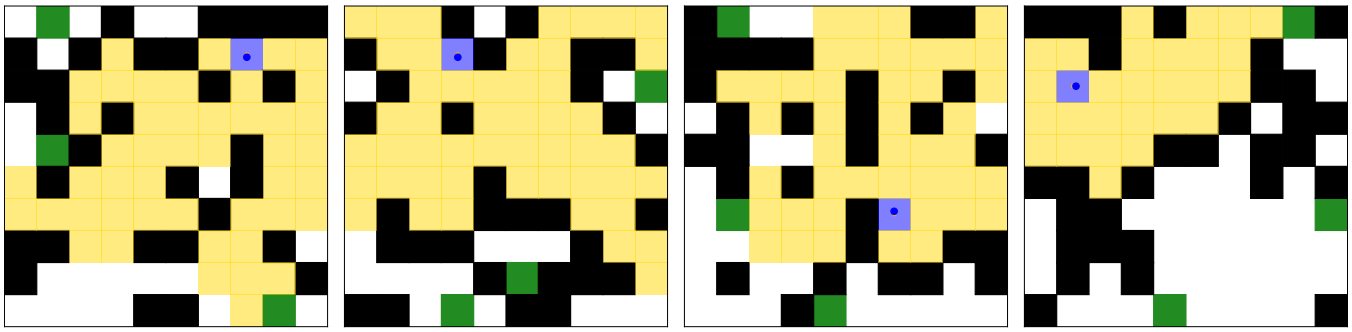

Fig. 2. Examples of the environments considered for the example in Section IV-A. The blue circle represents the agent, the blue square represents the agent's starting cell, the green squares are goals, the black squares are obstacles, and the gold region is the region explored by the learned behavior.

Similarly, we can define a volume fraction function over the allowable joint state space:

$$\mathcal{V}(Q) = \frac{\int_Q 1ds}{\int_{\mathcal{B}} 1ds}.$$

Finally, since the uniform distribution assigns probabilistic weight to a subset of events equivalent to their volume fraction in the sample space, $\mathbb{C}1$ resolves to the following:

$$\mathbb{C}1 \equiv \mathcal{V}(\mathbb{V}) \geq 1 - \epsilon.$$

Substituting this equivalency in $\mathbb{C}2$ completes the proof. ■

A. Extending to Non-Deterministic Behaviors

In the prior sections, we only considered deterministic system evolution and behavior graph resolution. However, it may be the case that either the system evolves or the behavior graph resolves non-deterministically. Our proposed verification framework should account for this non-determinism, and this section details how the prior procedure extends to this case. We will formalize this non-determinism directly in the context of verification. Specifically, we assume that we have a distribution by which we can draw robustness evaluations of system trajectories, *i.e.*

$$\rho(\Sigma(x_0, d), d) \text{ is sampled from } \pi_V$$

Note that this accounts for both cases where the initial system and environment states are potentially sampled randomly via a distribution π_X over the allowable space $\mathcal{B}$ as per (4) and the ensuing trajectories $\Sigma(x_0, d)$ are also randomly sampled from some unknown trajectory-level distribution π_S, arising from the aforementioned non-deterministic system evolution or behavior graph resolution.

As a result, we can follow the same verification method as in Theorem 1, though we cannot identify trajectories via initial conditions as we did in Assumption 1. The following assumption and corollary expresses this notion formally:

Assumption 2. Let ρ_B be the robustness measure for the behavior B as per equation (3), let $\{r^i = \rho_B(\Sigma(x_0^i, d^i), d^i)\}_{i=1}^N$ be the robustnesses of N trajectories sampled via the (unknown) distribution π_V, and let $r^* = \min_i r^i$.

Corollary 1. *Let Assumption 2 hold. If* $r^* = 1$, *then* $\forall\ \epsilon \in [0, 1]$, *the non-deterministic system* Σ *successfully executes the behavior* B *with minimum probability* $1 - \epsilon$ *and confidence* $1 - (1 - \epsilon)^N$, *i.e.:*

$$\mathbb{P}_{\pi_V}^N \left[\mathbb{P}_{\pi_V}[\rho(\Sigma(x_0, d), d) \geq r^*] \geq 1 - \epsilon\right] \geq 1 - (1 - \epsilon)^N.$$

Proof: This is a direct result of Theorem 1. ■

IV. Demonstrations

A. Exploratory Behavior Generation

To illustrate the verifiability of behaviors generated via Algorithm 1, this section will detail our efforts at using a natural language abstractor built on ChatGPT to construct an exploratory behavior.

System and Environment Description: To that end, the simulations to follow feature an agent idealized as a single integrator system on the plane and navigating within a 10×10 grid with obstacles and a few goals. The system state x is its planar position and its labels for each of the cells, *i.e.* $x \in [-5, 5]^2 \times \{\text{empty, obstacle, unexplored, goal}\}^{100} \triangleq \mathcal{X}$. The environment, *i.e.* obstacle and goal cells, is the subset of the overall label space where there exist 30 obstacles and 3 goals with no overlaps, *i.e.* $\mathcal{D} \subset \{\text{empty, obstacle, goal}\}^{100}$. The system dynamics as per (1) are known in this case, with single-integrator dynamics for the planar dimension and label updates when specifically provided by a controller - otherwise, labels remain constant.

Motion Primitives: The system has two primitives upon which the natural-language behavior generalizer can build behaviors. Their descriptions will follow:

$\mathcal{P}_1^s$: A label update function that updates the labels in the state x to match the labels of the cells immediately surrounding the agent, *i.e.* if the agent were in cell $(2, 3)$ the function updates the labels of cells $\{(2, 3), (3, 3), (1, 3), (2, 4), (2, 2)\}$.

Ξ : The set of all cells, *i.e.* $\Xi = \{0, 1, 2, \ldots, 9\}^2$.

A : A function outputting the cell the system currently occupies, *i.e.* if the system's planar position were $[-4.5, -3.5]$, the only valid parameter is cell $(0, 1)$.

U : Updates the state to reflect the environment labels of all adjacent cells.

R : A function outputting the portion of the state space where the labels for the agent's current and adjacent cells align with those of the environment. All other

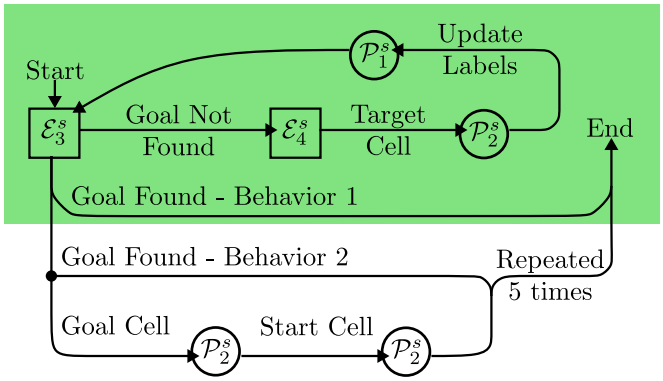

Fig. 3. Depiction of the directed behavior graph generated by Algorithm 1 for the example detailed in Section IV-A. The first behavior's graph is highlighted in green, the second behavior incorporates the first and the extra information is the unhighlighted part of the graph.

cell labels are unconstrained, *i.e.* if the agent's current and adjacent cells were all empty, then $R(\xi, d)$ would output the subset of the state space containing label vectors whose elements for those cells all read "empty" with no constraints on other elements.

$\mathcal{P}_2^s$: A navigation function that steers the agent to a desired cell while avoiding obstacles.

Ξ : The set of all cells, *i.e.* $\Xi = \{0, 1, 2, \ldots, 9\}^2$.

A : A function outputting the portion of the parameter space where the cell is reachable by the agent in the provided environment.

U : A Markov-Decision-based planner tracked by a PD controller that steers the agent to the desired cell while avoiding obstacles.

R : Outputs the portion of the planar state space encompassed by the desired cell, *i.e.* if the agent could reach cell $(2, 2)$, then $R(\xi = (2, 2), d) = [-2, -1]^2$.

Algorithm Information: We desired an exploratory behavior whereby the system searches the grid for a goal and after identifying a goal, oscillates back and forth between the goal and its starting location at least 5 times. As useful information for the task-following algorithm, the inputted information - string I in Algorithm 1 - indicated that the language model could use the following functions when determining edges in the outputted behavior graph:

$\mathcal{E}_1^s$: A function that outputs as a list, all the cells that have been explored by the agent thus far, *i.e.* all cells that have a label other than "unexplored" in the current state.

$\mathcal{E}_2^s$: A function that outputs as a list all cells immediately adjacent to the agent's currently occupied cell.

$\mathcal{E}_3^s$: A function that determines whether a goal has been found and outputs the corresponding cell.

Behavior 1: For the first step, we asked the algorithm to devise a behavior that explored the grid until it identified a goal. Specifically, the inputted behavior string s was as follows: *"Please construct a function that performs the following tasks in sequence. First, it searches over all explored cells that are not obstacles to find the explored cell with the highest number of unexplored neighbors. Let's call this identified cell, cell A. Second, it sends the agent to cell A and identifies the labels of all adjacent cells. Three, it repeats steps one and two until a goal has been found, at which point, it stops."* The part of the graph highlighted in green in Figure 3 shows the generated behavior graph. As part of this generation procedure, it used two of the provided functions $\mathcal{E}_1^s, \mathcal{E}_2^s$ to construct the edge decision function $\mathcal{E}_4^s$ whose description will follow:

$\mathcal{E}_4^s$: A function that searches over all explored cells - the list of explored cells is provided by $\mathcal{E}_1^s$ - and assigns to each cell the number of its adjacent cells that are unexplored - the list of adjacent cells is provided by $\mathcal{E}_2^s$. Reports the first cell in the list with the maximum number of unexplored neighbors.

Behavior 2: We wanted to build on the prior behavior for the latter half of our goal, and as such, informed the LLM of the existence of this prior behavior in the list of existing behaviors denoted as $\mathbb{B}$ in Algorithm 1. Then, as the user, we requested the following from the LLM: *"Please construct a function that performs the following tasks in sequence. First, it finds a goal. Second, it moves between the goal and its starting location 5 times."* The behavior graph for this second behavior is the unhighlighted graph in Figure 3.

Verification Procedure and Remarks: As Behavior 2 utilized Behavior 1, verifying both amounts to verifying the former. Following the results of Theorem 2, we recorded a data set $\mathbb{D}$ of parameters, primitives, and terminal states while running the second behavior. The certificates per equation (2) amount to checking that updated labels matched their true labels after running primitive $\mathcal{P}_1^s$ and checking that the system occupied the desired cell after running primitive $\mathcal{P}_2^s$. The allowable joint state space $\mathcal{B}$ as per (4) was the portion of the joint space where the system starts in a state x such that at least one goal is reachable in the corresponding environment d. Finally, the verification procedure uniformly randomly sampled state pairs $(x, d) \in \mathcal{B}$ and checked the corresponding certificates for each run of the behavior.

After running the second behavior from 100 randomly sampled initial state pairs, the behavior terminated successfully every time. As such, by Theorem 2 we expect that the second behavior will run successfully for 95% of possible state pairs and we are 99.4% confident in this statement - we generated these numbers by substituting $\epsilon = 0.05$ and $N = 100$ for Theorem 2. To validate these statements, we ran the second behavior in 2000 more sampled environments, and it terminated successfully every time. If we were incorrect in our prior statement that the behavior would run successfully for at least 95% of feasible state pairs $(x, d) \in \mathcal{B}$, then we would have been effectively guaranteed to identify such a failure over the 2000 subsequent runs. As we did not, we are confident in our corresponding statement. Furthermore, while the synthesized behaviors seem rudimentary, they suffice to indicate that our behavior synthesis scheme produces effective and verifiable behaviors.

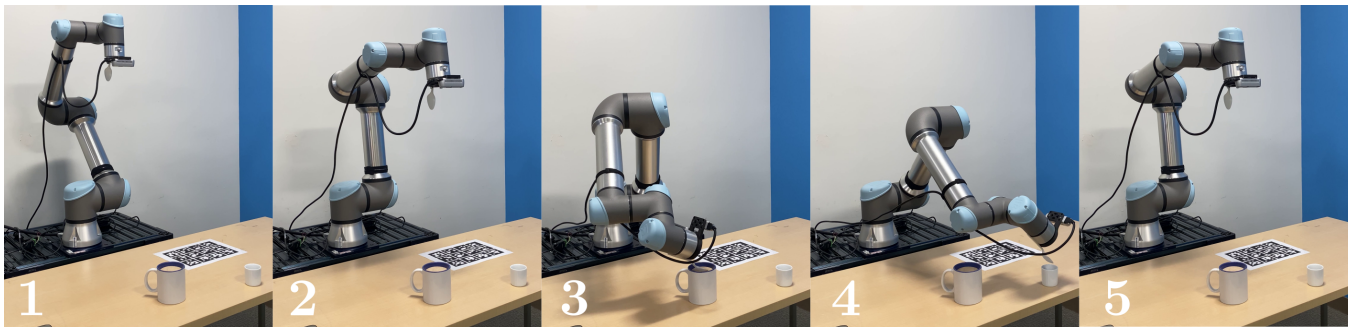

Fig. 4. Depiction of the learned scooping behavior. In this case, the motion was coded previously, but contingent on the arm's ability to sense the cups in its environment. As such, the LLM interface only asked for the end-user to provide that initial positioning (1) wherein the arm had a high likelihood of sending both cups. Then the LLM behavior first moves to the desired sensing position (2), calls the scooping primitive as seen in (3)-(4), and returns to the instructed sensing position in (5) in case any of the cups shifted during the procedure. Then the process repeats.

B. Scooping of Granular Media

Our second demonstration focusing on granular media scooping illustrates the framework's utility in helping end-users set up repetitive, verifiable tasks.

System and Environment Description: The scooping problem consists of picking up material from a donor container and depositing it into a receiver container using a UR5e 6-DOF robot arm with a wrist-mounted RealSense depth camera. While a rudimentary scooping motion has been programmed *apriori*, it does not know the environment in which it will be performing this motion - similar to the situation when a pre-programmed robot has to be initialized for specific use. The robot's state $x \in \mathbb{R}^6$ is the full pose of the end-effector, the control input u corresponds to joint rotations, and the environment d corresponds to the locations and orientations of the donor and receiver containers and the level and distribution of sand in the donor container.

Motion Primitives: In this case, the system only has one primitive, the scooping primitive, described as follows:

$\mathcal{P}^r$: A primitive performing a scooping motion from a donor container to a receiver container.

- Ξ : The space of feasible end-effector poses where a parameter $\xi \in \Xi$ denotes the pose in which the robot will sense all objects in the environment to start the scooping motion.
- A : A function outputting the space of end-effector poses from which all containers are in view of the onboard vision system.
- U : A controller that performs the scooping motion.
- R : A function that outputs a ball around the provided parameter within which the end-effector's pose will lie upon the termination of the scooping motion.

That being said, the acceptance function A is implicitly defined and impossible to know *apriori*. Here, we intend for the algorithm to assist the end-user in selecting a parameter ξ whose validity, *i.e.* existence in $A(x, d)\ \forall\ (x, d) \in \mathcal{X} \times \mathcal{D}$, can be checked through the ensuing verification procedure.

Algorithm Information: To assist the user in picking such a parameter ξ, the algorithm was provided an information string I describing a helper function $\mathcal{E}_1^r$ that translated and rotated the end-effector a desired amount. This string also included several examples of natural language translations to inputs for this function $\mathcal{E}_1^r$. Additionally, the string included another function $\mathcal{E}_2^r$ that saved the end-effector pose for future reference, and the LLM was told to call this function if the user deemed the current end-effector pose satisfactory.

Behavior Generation and Verification: The task-model repeatedly queried the user for end-effector translations and rotations and as to whether or not the user deemed the current pose sufficient for sensing any placement of containers. As such, there was no singular behavior prompt s. However, as the resulting behavior repetitively executes the scooping primitive with the user-provided sensing pose parameter ξ, this behavior can be verified by the results of Corollary 1. To do so, before every scooping motion, we placed the containers at a computer-generated randomly chosen distance from a pre-determined set-point. As we are manually placing containers at the pre-determined locations, there will be noise affecting this placement, though we assume this noise is independent for successive placements. We will denote this distribution of container placements via π. As there is no need to sample over initial robot states - the system always starts and ends at the parameterized sensing pose ξ every iteration - we can draw independent environments - container placements - via our distribution π and record the robot's ability to perform its scooping motion in each placement. Doing so for 59 sampled environments with successful trials each time indicates according to Corollary 1 that if we continued to sample environments and test the system accordingly, the system would succeed at least 95% of the time and we are at least 95% confident in that statement.

V. Conclusion

We propose a framework by which a natural language abstractor can synthesize verifiable behaviors as a directed graph over provided motion primitives. To showcase the increased flexibility and verifiability of the synthesized behaviors, we instructed the task-following model to construct an exploratory behavior for a simulated planar agent and a scooping behavior for a robotic arm. In both cases, the generated behavior was verifiable via the aforementioned method, and we were able to validate our probabilistic verification statements in simulation.

REFERENCES

[1] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97. Citeseer, 1997, pp. 12–20.

[2] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual review of control, robotics, and autonomous systems*, vol. 3, pp. 297–330, 2020.

[3] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7, no. 2, p. 17, 2018.

[4] Y. Lin, S. Ren, M. Clevenger, and Y. Sun, "Learning grasping force from demonstration," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 1526–1531.

[5] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 763–768.

[6] E. Misimi, A. Olofsson, A. Eilertsen, E. R. Øye, and J. R. Mathiassen, "Robotic handling of compliant food objects by robust learning from demonstration," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6972–6979.

[7] O. M. Manyar, Z. McNulty, S. Nikolaidis, and S. K. Gupta, "Inverse reinforcement learning framework for transferring task sequencing policies from humans to robots in manufacturing applications," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 849–856.

[8] Y. Wang, R. Xiong, L. Shen, K. Sun, J. Zhang, and L. Qi, "Towards learning from demonstration system for parts assembly: A graph based representation for knowledge," in *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent*. IEEE, 2014, pp. 174–179.

[9] B. Keller, M. Draelos, K. Zhou, R. Qian, A. N. Kuo, G. Konidaris, K. Hauser, and J. A. Izatt, "Optical coherence tomography-guided robotic ophthalmic microsurgery via reinforcement learning from demonstration," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1207–1218, 2020.

[10] T. Osa, K. Harada, N. Sugita, and M. Mitsuishi, "Trajectory planning under different initial conditions for surgical task automation by learning from demonstration," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6507–6513.

[11] J. W. Kim, C. He, M. Urias, P. Gehlbach, G. D. Hager, I. Iordachita, and M. Kobilarov, "Autonomously navigating a surgical tool inside the eye by learning from demonstration," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7351–7357.

[12] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, no. 2, apr 2017. [Online]. Available: https://doi.org/10.1145/3054912

[13] C. Chao, M. Cakmak, and A. L. Thomaz, "Towards grounding concepts for transfer in goal learning from demonstration," in *2011 IEEE International Conference on Development and Learning (ICDL)*, vol. 2. IEEE, 2011, pp. 1–6.

[14] K. Hausman, Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim, "Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets," *Advances in neural information processing systems*, vol. 30, 2017.

[15] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-shot visual imitation learning via meta-learning," in *Conference on robot learning*. PMLR, 2017, pp. 357–368.

[16] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.

[17] Y. Cui, S. Niekum, A. Gupta, V. Kumar, and A. Rajeswaran, "Can foundation models perform zero-shot task specification for robot manipulation?" in *Learning for Dynamics and Control Conference*. PMLR, 2022, pp. 893–905.

[18] D. Shah, A. Sridhar, N. Dashora, K. Stachowicz, K. Black, N. Hirose, and S. Levine, "Vint: A large-scale, multi-task visual navigation backbone with cross-robot generalization," in *7th Annual Conference on Robot Learning*, 2023.

[19] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[20] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *Microsoft Auton. Syst. Robot. Res*, vol. 2, p. 20, 2023.

[21] F. Stulp, E. Theodorou, M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning motion primitive goals for robust manipulation," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 325–331.

[22] W. Ubellacker and A. D. Ames, "Robust locomotion on legged robots through planning on motion primitive graphs," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 12 142–12 148.

[23] P. Akella, W. Ubellacker, and A. D. Ames, "Probabilistic guarantees for nonlinear safety-critical optimal control," *arXiv preprint arXiv:2303.06258*, 2023.

[24] P. Akella, A. Dixit, M. Ahmadi, J. W. Burdick, and A. D. Ames, "Sample-based bounds for coherent risk measures: Applications to policy synthesis and verification," *arXiv preprint arXiv:2204.09833*, 2022.

[25] P. Akella, M. Ahmadi, and A. D. Ames, "A scenario approach to risk-aware safety-critical system verification," *arXiv preprint arXiv:2203.02595*, 2022.

[26] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.

[27] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.