Programming by Demonstration for Locally k-Testable tasks

Saurabh Arora and Herbert G. Tanner

***Abstract*— This paper proposes PbD (Programming by Demonstration) of the tasks representable as Locally k-Testable (or LT_k) subclass of regular languages, in a system modeled as a Markov Decision Process. The method uses predicate abstraction and language identification in the limit to infer a symbolic task. The learned task is implemented by constraining the evolution of MDP. Using relatively small training sample, the approach generalizes to all possible ways of accomplishing a learned task, unlike inverse reinforcement learning method. A Grid-world case study validates the results.**

***Index Terms*— Learning by demonstration; imitation learning; grammatical inference**

I. INTRODUCTION

In the process of PbD (Programming by Demonstration), an expert user provides examples to teach a task to a machine with a learning algorithm (learner). Latter is expected to generalize the inferred task to all unseen contexts. A challenge is how to assure that the learner will infer all possible ways to accomplish the task. A potential solution is reported here for tasks exhibiting recognizable patterns. Typical pattern-based tasks taught using PbD include dancing, or solving Towers of Hanoi [1]; modeling activity at a convenience store [2]; and surgical pattern-cutting [3]. In this paper, a task is assumed to be encoded as a formal language (i.e., a string set formed by a predefined alphabet [4]) belonging to the class of Locally k-Testable (LT_k) languages [5], which is a subclass of regular formal languages.

The words (strings) in a LT_k language exhibit distinguishing patterns in how alphabet symbols are arranged within each word. When an LT_k language encodes a task, these patterns can capture the logic specification that "a task finishes when learner does $x,\ldots,y$ avoiding $x',\ldots,y'$, accomplishing g in a finite unspecified time" (e.g. *Pick brown and black objects, place them on shelves avoiding black objects from going on brown shelf, and brown ones from going on the black shelf, putting more brown than black* is a LT_2 task) or "a task finishes when learner does x with $x',\ldots,$ y with y', accomplishing g in a finite unspecified time". Using *identification in the limit from positive data* [6], the reported method (called 'PbD-LT_k') infers such a LT_k task using a finite set of words belonging to the language.

How to guarantee complete task generalization is an open question for the existing PbD methods. The answer is still elusive in typical PbD methods such as inverse reinforcement learning (IRL [7]) [8]. A complete generalization in IRL is possible only with a large training sample (especially for large state space) and an accurate estimate of feature values (feature counts in IRL). Both the conditions are emphasized in prevalent literature as difficult to satisfy [9], [10].

If one, however, assumes that the discrete-event dynamics of some physical system can be adequately abstracted in the form of a formal language, and the desired task can be encoded as an identifiable language, then grammatical inference methods can ensure complete learning in the limit, assuming representative data has been provided (Theorem 5 and Lemma 2, [11]). The task is guaranteed to be completely learned, and consequently complete generalized. Therefore, learner does not loose any option for accomplishing the task.

Methods like IRL require a task representation to have an approximate form e.g. a linear combination of features [12], with few exceptions [13]. No particular restrictive assumptions need to be placed on the learning target in grammatical inference methods, other than specifying the language subclass where it needs to belong to. This claim is defendable in the sense that such an assumption may be at most as conservative as requiring task descriptions to take the form of some linear parameterization (combination) of features. While which type of assumption is more stringent can be debated, it is also true that the expert employed in IRL will still require the designer to engineer a broad set of relevant features to enable an acquisition of a task [8], [13]. Such features are usually associated with the structures of task model and learning algorithm employed. The features that may emerge as the part of a symbolic abstraction process that sets the stage for the application of a grammatical inference algorithm, can potentially be used for other PbD methods. In other words, symbolic abstraction of physical processes combined with grammatical inference does fall within the general framework of PbD.

Whenever a physical process is modeled in the form of a discrete computation model—be that an automaton, discrete-event system, Petri-net, or MDP—there is always the issue of computational complexity associated with size and resolution. The more refined a model is, the more complex it becomes. Just as in existing PbD methods computing a global solution (a Markov policy) becomes increasingly more challenging if size of state space increases [10], PbD-LT_k is not impervious to the *curse of dimensionality*.

Section II outlines a gap in existing literature. Section III introduces the problem, while Section IV describes the method PbD-LT_k claimed to fill gap. Section V offers comparison with IRL for validation, explaining limitations of solution. Section VI concludes the paper.

The authors are with the Department of Mechanical Engineering, University of Delaware, Newark, DE, USA {sarora,btanner}@udel.edu

II. Background

The learning method in this paper is introduced in the context of PbD [14]. A set of system trajectories is understood here as a behavior. A task refers to particular subset of a behavior, and this subset is taken as satisfying a particular specification. A complete generalization of learned task is very challenging [8]–[10], even with a large training set and a function approximation in the task (linear in feature space). Explicit user intervention have been devised [8], [10] to minimize the task solutions missed by the learning algorithm. Necessitating the continuous feedback may impose a condition (continuous availability of user) impracticable for many applications, e.g. assistive robotics for elderly or subjects with special needs. This paper attempts to use the task models with a structure that can be exploited to ensure generalization without linear approximation, through established grammatical inference methods [11].

III. Problem Statement

A. Technical Preliminaries

An alphabet Σ is a finite set of symbols [4]. Σ^* refers to set of all strings of any finite length. A string is any finite length sequence of symbols, $w = \sigma_1\sigma_2\ldots\sigma_n \in \Sigma^*,\ \sigma_i \in \Sigma$, $i \in \mathbb{N}$. A formal language $L \subset \Sigma^*$ is a set of strings. The grammar G is a set of rules for constitution of strings in L, s.t. $\mathrm{L}(G) = L$. Symbols $\rtimes$ and $\ltimes$ mark beginning and end of a string respectively. $|C|$ is the cardinality of a finite set C. $u \cdot v$ or uv denotes the concatenation of strings u and v. The concatenation of two string sets C_1 and C_2 is $C_1 \cdot C_2 = \{uv | u \in C_1, v \in C_2\}$. A DFA (Deterministic Finite-state Automaton) over alphabet Σ is a tuple $A \triangleq (Q, \mathbf{\Sigma}, T, I, F)$ comprising a set Q of states, an extended alphabet $\mathbf{\Sigma} = \Sigma \cup \{\rtimes, \ltimes\}$, a transition function $T : Q \times \mathbf{\Sigma} \to Q$ which can be expanded recursively (i.e. $T(q, \sigma w) = T(T(q, \sigma), w), \sigma \in \Sigma, w \in \Sigma^*$), initial states I, and accepting states F. The symbols $\{\rtimes, \ltimes\}$ mark the beginning and end of strings accepted by A. The language accepted by A is $\mathrm{L}(A) = \{ \rtimes w \ltimes \in \{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\} \mid T(q, w') \in F, q \in I, I = \{\rtimes\}, w' = w \ltimes \}$. It is known that such a language is regular [4].

A k-factor is a k length contiguous subsequence of a string $\rtimes w \ltimes$ [5]. A string extension function $\mathfrak{f}^{\mathsf{se}}_k : \Sigma^* \to \mathrm{Fin}(\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\})$[1] is a function mapping a string w to the k-factors of $\rtimes w \ltimes$, e.g. for $w = acb$, $\mathfrak{f}^{\mathsf{se}}_2(w) = \{\rtimes a, ac, cb, b \ltimes\}$ (If $|w| < k$, then $\mathfrak{f}^{\mathsf{se}}_k(w) = w$) [11]. A sub-regular language is a regular (string-extension) language, with characteristic string patterns, for which the membership of a string can be decided by using a set of k-factors (grammar).

LT_k is a class of sub-regular languages [5]. Language L is LT_k if there is some $k \in \mathbb{N}$ such that for all $w_1, w_2 \in \Sigma^*$, if $\mathfrak{f}^{\mathsf{se}}_k(w_1) = \mathfrak{f}^{\mathsf{se}}_k(w_2)$ then $w_1 \in L \iff w_2 \in L$ [5]. $G_{k\mathrm{LT}} = \{\{\mathfrak{f}^{\mathsf{se}}_k(w)\} \mid w \in L\}$ denotes a (string-extension) grammar comprising the sub-grammars made of k-factors, generating LT_k language $L = \mathrm{L}(G_{k\mathrm{LT}})$ (cf. [11]). The k-factors of a LT_k string are exactly similar to those in one of the sub-grammars for the language. An example of LT2 grammar is $G_{2\mathrm{LT}} = \{\{\rtimes a, ac, c \ltimes\}, \{\rtimes b, bc, cb, c \ltimes\}, \{\rtimes a, ac, cb, b \ltimes\}\}$ with 3 sub grammars. The strings comprising 2-factors in these sub grammars belong to $\mathrm{L}(G_{2\mathrm{LT}})$, e.g. $\rtimes ac \ltimes$, $\rtimes bcbc \ltimes$, and $\rtimes acb \ltimes$.

1) Identification from Text: presentation of language L is a function $\mathrm{pr} : \mathbb{N} \to L \subseteq \Sigma^*$ that indexes all elements of L. The process creates the *text* for L - $\mathrm{txt_L}$, a sequence of words from L separated by the terminal symbols $\ltimes, \rtimes$. $\mathrm{txt_L}(i)$ denotes ith string in the sequence. The initial segment of the text that includes strings $\mathrm{txt_L}(0)$ to $\mathrm{txt_L}(j)$ is called a *demonstration* (of length $j+1$) and is denoted $\mathrm{txt_L}[j]$, e.g. $\mathrm{txt_L}[5] = ac \ltimes \rtimes accb \ltimes \rtimes abbc \ltimes \rtimes ac \ltimes \rtimes accb \ltimes$. We consider only *positive* text here, i.e., strings $\mathrm{txt_L}(i)$ belong to L (some variants use strings that are not from the language). An inductive inference machine (IIM) ϕ accepts increasingly large initial segments of a text ($\mathrm{txt_L}[i]$) and outputs a hypothesis after each input, identifying target language in the limit (cf. [15]).

B. Symbolic abstraction of MDP behavior

An atomic proposition is a logic predicate evaluated on a system state. It can be either action based ⟨a specific motion primitive is executed⟩, or sensor based ⟨sensed information (e.g. size) satisfies a requirement⟩, or location based ⟨learner is in a specific region⟩. The set of atomic propositions is AP. A *literal* is an atomic proposition or its negation, and *sentences* are conjunctions of literals where each atomic proposition appears at most once, (e.g. $\alpha = (r_1 \wedge r_2 \wedge (\neg r_3)), r_i \in AP$) [16]. S_{AP} denotes the set of sentences. The reported method uses the sentences evaluated on system states, as language symbols, to abstract the information about system dynamics.

An Markov Decision Process (MDP) $M = (S, \mathscr{A}, T_M, \{P_M\},\ \gamma, D(I_S), L_M, R)$ can be either continuing (evolution does not terminate) or episodic (evolution stops on reaching terminal states which completes an episode, and then re-initializes) [17]. comprising states S, actions $\mathscr{A}$, a transition function $T_M : (S \times \mathscr{A}) \mapsto 2^S$, transition probabilities P_M, a discount factor γ, a distribution $D(I_S)$ over initial states I_S, labeling function $L_M : S \to S_{AP}$, and a reward $R : S \times A \to \mathbb{R}$. $\mathscr{H}(s) = \{a \in \mathscr{A} |\ T_M(s,a) \in S, s \in S\}$. L_M enables the symbolic abstraction by mapping states to corresponding sentences. A control policy $\pi : S \to A$ governs the evolution of $M \backslash R$. A continuing MDP w/o reward represents the interaction between learner and environment, $M \backslash R = (S, \mathscr{A}, T_M, \{P_M\}, \gamma, D(I_S), L_M)$.

C. Problem Statement

A demonstrator trains a learner on a task expressible as a LT_k language, $L = \mathrm{L}(G_{k\mathrm{LT}})$ with state labels as alphabet (i.e. $\Sigma = S_{AP}$). We assume that the learner has this information about the task.

The problem is stated as follows: From the trajectories $\{(s^i_0, a^i_0), (s^i_1 \ldots (s^i_j, a^i_j)\}^e_{i=1}\ , s_j \in S, a_j \in \mathscr{A}, i, j, e \in \mathbb{N}$ given by demonstrator, learner perceives state labels $\{L_M(s_j) = \alpha_j \in \Sigma = S_{AP},\ s_j \in S\}$, creating the strings (symbolic trajectories of labels) $\mathrm{txt_L}(i) = \alpha_0\alpha_1\alpha_2 \ldots \alpha_y,\ i, y \in \mathbb{N}, \alpha_y \in \Sigma$.

[1] $\mathrm{Fin}(B)$ denotes the set of finite subsets of set B. $|w|$ is the count of concatenated symbols that make w.

TABLE I

COMPONENTS OF A TASK, $A = (Q, \mathbf{\Sigma}, T, I, F)$

Q $\{q = \langle w, g\rangle | w = \sigma_1\sigma_2\ldots\sigma_t \in \Sigma^*\cup\{\rtimes\}\cdot\Sigma^*\cup\{\rtimes\}\cdot\Sigma^*\cdot\{\ltimes\}\cup\Sigma^*\cdot\{\ltimes\}, \sigma_i\in\Sigma, i\in\mathbb{N}, 0\le t\le(k-1), g\in \mathrm{Fin}(\mathfrak{f}_k^{\mathsf{se}}(\{\rtimes\}\cdot\Sigma^*\cdot\{\ltimes\}))\}$
$\mathrm{Fin}(\mathfrak{f}_k^{\mathsf{se}}(\{\rtimes\}\cdot\Sigma^*\cdot\{\ltimes\}))$ denotes all the possible sets of k-factors of the strings in set $\{\rtimes\}\cdot\Sigma^*\cdot\{\ltimes\}$. A state is a tuple of a string w and a set g of k-factors of an arbitrary string.

$\mathbf{\Sigma}$ $\Sigma\cup\{\rtimes,\ltimes\}$

T[2] $\{\langle w,g\rangle\in Q | Sf^1(w)\neq\ltimes\}\times\mathbf{\Sigma}\backslash\{\rtimes\}\to Q$
$T(\langle w,g\rangle,\sigma) = \langle w', g'\rangle, w' = Sf^{(k-2)}(w)\cdot\sigma, g' =$
$\begin{cases}\{Sf^{(k-2)}(w)\cdot\sigma\} & 1\le t<(k-1)\wedge g=\{\mathsf{w}\}, \mathsf{w}\in\{\rtimes\}\cdot\Sigma^{t-1}\\ \{w\cdot\sigma\} & t=(k-1)\wedge g=\{\mathsf{w}\}, \mathsf{w}\in\{\rtimes\}\cdot\Sigma^{t-1}\\ g\cup\{w\cdot\sigma\} & t=(k-1)\wedge g=\{\mathsf{w}\}\cup g'', \mathsf{w}\in\{\rtimes\}\cdot\Sigma^{t-1}\\ & g''\in\mathrm{Fin}(\Sigma^k)\cup\emptyset\end{cases}$
A LT_k string $\rtimes w\ltimes, w=\sigma_1\sigma_2..\sigma_k...\sigma_m\in L$ has a prefix k-factor $\rtimes\sigma_1...\sigma_{k-1}$ along with other k-factors (interiors and suffix) in it. Three kinds of transitions construct a run in which the DFA recognizes the parts of a string $\rtimes w\ltimes$: type 1 transitions till the end of the prefix part (applicable if $k>2$), type 2 transition ends the prefix (if $k\ge 2$), and type 3 transitions for the rest of k-factors (k arbitrary). This run ends in accepting states, F.

I $\begin{cases}\langle\lambda,\{\rtimes\}\rangle & k=1\\ \langle\rtimes,\{\rtimes\}\rangle & k>1\end{cases}$

F $\{\langle w,g\rangle\in Q | g\in G_{k\mathrm{LT}}\}$
An accepting state is a state with a sub-grammar of $G_{k\mathrm{LT}}$ in it

Objective of learning is that the trajectory followed by learner should satisfy the constraints encoded in $\mathrm{L}(G_{k\mathrm{LT}})$/ DFA A s.t. $L(A)=\mathrm{L}(G_{k\mathrm{LT}})$. Formulate a PbD method using which the learner generalizes the task over whole state space of the system $M\backslash R$. The method is expected to control the system evolution, completing task A.

For IIM ϕ created for the problem above, let the hypothesis about $\mathrm{L}(G_{k\mathrm{LT}})$ after nth demonstration $\mathrm{txt_L}[n]$ (sequence of first (n+1) strings) be the grammar $\phi(\mathrm{txt_L}[n]), n\in\mathbb{N}$. The true hypothesis is $G_{k\mathrm{LT}}$. A learning algorithm using ϕ completely identifies the language $\mathrm{L}(G_{k\mathrm{LT}})=L$, iff for a demonstration $\mathrm{txt_L}[n], n\in\mathbb{N}$ in presentation pr, it converges to $\phi(\mathrm{txt_L}[n])$, such that $\phi(\mathrm{txt_L}[n])=G_{k\mathrm{LT}}$.

IV. PBD-LT_k

A. LT_k Task

A LT_k task $\mathrm{L}(G_{k\mathrm{LT}})$ is represented as DFA $A=(Q,\mathbf{\Sigma},T,I,F)$ (Table I). The symbols $\{\sigma\in\Sigma\ =S_{AP}$ represent task specific constraints (sentences) satisfied by states. Therefore, a sub-grammar of $G_{k\mathrm{LT}}$ (in states F) represents a set of constraints. Each sub-grammar embodies an option to meet the desired specification because a solution (LT_k string) can be constituted of the k-factors from any sub-grammar.

[2] $\mathrm{Sf}^n(w)$ is length n suffix of w. Σ^{t-1} refers to the set of all strings of length $t-1$.

TABLE II

COMPONENTS OF M_p

S_p $\begin{cases}\{s_p=(s,q) | L_M(s)=Sf^1(w)\in\Sigma, q=\langle w,g\rangle\} & k>1\\ \{s_p=(s,q) | L_M(s)\in g\} & k=1\end{cases}$

L_p $L_p(s_p)=L_M(s)\in S_{AP}$

T_p $S_p\times\mathscr{A}\to 2^{S_p}$
$T_p(s_{p_1},a)=\{s_{p_2}=(s_2,q_2) | s_2=T_M(s_1,a), q_2=T(q_1,\sigma),\sigma\in\Sigma\}, \forall s_{p_1}=(s_1,q_1)\in S_p, a\in\mathscr{H}(s_1)$
Transition from current product state s_{p_1} on input action a of $M\backslash R$ leads to the product state s_{p_2} which has both its components reachable from respective components of s_{p_1}.

P_p $P_p(s_{p_2}|s_{p_1},a)=P_M(s_2|s_1,a)$

I_p $\{(s,q)\in S_p | s\in I_S\}$

F_p $\{(s,q)\in S_p | T(q,\ltimes)\in F\}$
A terminal state is a product state s_p which has its DFA component with a transition defined to an accepting state.

B. Learning

A string extension learner (SEL) learns a language as the corresponding grammar, by iteratively combining the sets of k-factors mapped by a (language-specific) string extension function, completely converging to the target in the limit (Theorem 5 in [11]).

The SEL $\phi_{k\mathrm{LT}}^{\mathsf{se}}$ is an IIM identifying a LT_k language, $\phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[i]) =$
$\begin{cases}\phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[i-1]) \\ \quad\cup\{\mathfrak{f}_k^{\mathsf{se}}(\mathrm{txt_L}(i))\}, & \{\mathfrak{f}_k^{\mathsf{se}}(\mathrm{txt_L}(i))\}\not\subseteq\phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[i-1])\\ \phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[i-1]), & \text{otherwise}\end{cases}$
for $i\in\mathbb{N}$; with convention $\phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[-1])=\emptyset$.

An extension of $\phi_{k\mathrm{LT}}^{\mathsf{se}}$ is created to identify the language's DFA A instead of only its grammar. The learning starts with a construction of a 'graph without accepting states'. With each input $\mathrm{txt_L}[i]$, it marks the accepting states $\langle w,g\rangle\in Q$ which has its set g of k-factors same as k-factors of the string ($\{\mathfrak{f}_k^{\mathsf{se}}(\mathrm{txt_L}(i))\}$). The marking process results in F on the graph, completing the inference of A. This *learner of task* $\mathrm{L}(G_{k\mathrm{LT}})=L(A)$ is implemented as the algorithm of Figure 5 (Appendix) followed by an extended $\phi_{k\mathrm{LT}}^{\mathsf{se}}$ in Figure 6.

C. Constraining the System Behavior

A product enforces the constraints encoded in A on the behavior of $M\backslash R$, resulting in a new MDP expressed as $M_p = M\backslash R\bigotimes A = (S_p,\mathscr{A},T_p,\{P_p\},\gamma,D(I_p),L_p,F_p)$ (see Table II). The process introduces terminal states to the dynamics of $M\backslash R$. It is inspired by synchronous products in the field of model checking [18]. The desired specification is the membership in task (language $\mathrm{L}(G_{k\mathrm{LT}})$ as A). F in A and F_p in M_p include the sub-grammars symbolizing the desired specification. Thus, a trajectory from I_p to F_p will completely meet it. In summary, specification (membership in L) was a reachability objective in A, and is consequently expressed as the reachability in M_p after product.

1) Task Implementation: After inferring A, the learner computes M_p using algorithm in Figure 7. Figure 1 shows

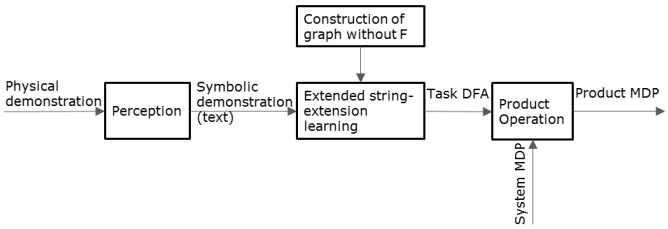

Fig. 1. PbD-LT$_k$ or PbD of a LT$_k$ task

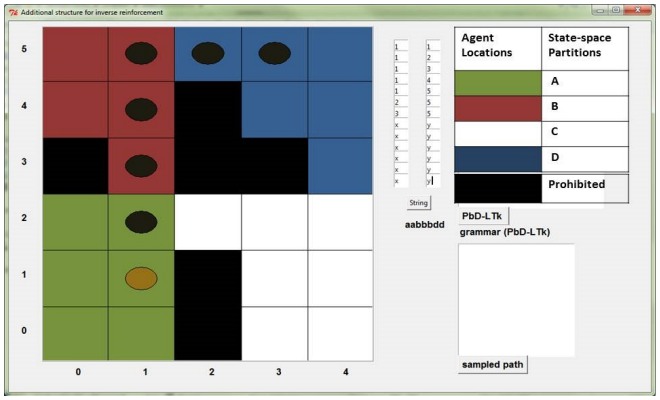

Fig. 2. The agent (circle) moves in cells of four colors. A trajectory is shown as a succession of cell locations (starting with a light-colored circle). A trainer inputs a physical demonstration in a vertical array of boxes. The demonstration shown here begins at $(1,1) \in S$ following partition sequence A→B→D, and is perceived as the string $\text{txt}_{\text{L}}(i)$='aabbbdd' of labels. The bottom right text box is meant for indicating the output of PbD-LT$_k$.

the steps for PbD-LT$_k$. After product, a solution can be found either using a simple graph search on M_p or using dynamic programming with a fixed negative reward.

V. Comparative Study of Generalization

This section compares the extent of task generalization by IRL with that by PbD-LT$_k$ when the available training sample is not large.

Example (Navigating a Grid-world): Consider a Grid-world with a simulated navigating agent inside it. A state $s \in S$ is a grid of 4×5 cells with the agent inside a specific cell (inspired from Grid-world in [12]). s is expressed using Cartesian position (x,y) of that cell. Some of the states are not navigable. S has 4 partitions $\mathfrak{B} = \{A,B,C,D\}$. $\mathscr{A} = \{l \text{ (left)}, r \text{ (right)}, u\text{(up)}, d \text{ (down)}\}$. $\forall s \in S, a \in \mathscr{H}(s)$,

$$T_M(s,a) = \begin{cases} s' & \text{with probability } P_M(s'|s,a) = 0.8 \\ s & \text{with probability } 1 - P_M(s'|s,a) \end{cases}$$

We take $\gamma = 0.95$ and assume $D(\cdot)$ to be a uniform distribution over $I_S = A \cup B \cup C \cup D$.

The learner (navigating agent) in Grid-world is required to be programmed by demonstration for the task *from any initial configuration $s \in I \subset S$, visit partition D avoiding transitions from A to C and from C to A, stopping at D*. Both IRL and PbD-LT$_k$ are implemented for the purpose, using a Python based GUI (figure 2). The trainer provides same demonstrations in each of the cases. Assuming a worst case scenario (a sufficiently abundant and diverse training is unachievable), she demonstrates a small set of physical trajectories $\{(s_0^i, a_0^i), (s_1^i, a_1^i), \ldots\}_{i=1}^{e}, e = 5$ ending in a particular subset of the states, $\{(3,4),(4,4)\} \subset D \subset S$. Each one of these trajectories starts from I_S and ends up in $\{(3,4),(4,4)\}$. A → B→ D, B→ D, C→ D are the sequences of partitions that are visited in these demonstrations.

A. IRL

The learner in IRL converges to a task (expressed as a policy $\tilde{\pi}$) by imitating the policy π_E of demonstrator [12]. In IRL framework, $M \backslash R$ includes features $\psi : (S \times \mathscr{A}) \mapsto [0,1]^b$ and a bounded reward $R(s,a) = \omega^T \psi(s,a)$, $\omega \in \mathbb{R}^b$ ($\|\omega\| \leq 1$). We make the value of π_E (feature counts [12]) represent the task above. $R(s,a) = \omega^T \psi(s,a) < 0$. Features are binary predicates $\psi : (S \times \mathscr{A}) \mapsto \{0,1\}^{16}$ ($b = 16$). A feature for pair $(j_1, j_2) \in \mathfrak{B} \times \mathfrak{B}$ is a logic predicate: ⟨after taking an action a in a state s of space partition j_1, system transitions to a state in partition j_2⟩. The component ω_{j_1,j_2} denotes the weight corresponding to $(j_1, j_2) \in \mathfrak{B} \times \mathfrak{B}$. $\forall (j_1, j_2) \in \mathfrak{B} \times \mathfrak{B} \backslash \{(A,C),(C,A)\}$, $\omega_{j_1,j_2} = \omega_c = -0.5$. $\omega_{A,C} = \omega_{C,A} = -2 < \omega_c$. Consequently, the reward for prohibited transitions, *A to C and C to A*, is lower as compared to that for others.

The learner acquires the terminal states of system MDP from demonstrations, making the MDP episodic after training. $M \backslash R$ is a continuing MDP before training. The agent learns the terminal states $F_S = \{(3,4),(4,4)\}$ during training by observing the last state of each demonstration, thereby making $M \backslash R$ episodic after learning the task (figure 3). The learning algorithm converges to policy $\tilde{\pi}$. Starting in $s_0 \in I_S = A \cup B \cup C$, following $\tilde{\pi}$, the navigating agent reaches $F_S \subset D$ while meeting the task specification.

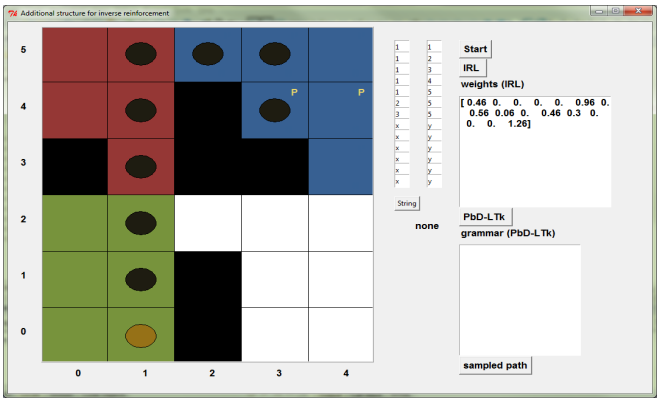

Fig. 3. Top right text box shows output from IRL. Cells with 'P' show the terminal states of system MDP marked by IRL ($F_S = \{(3,4),(4,4)\}$). Trajectory sampled from the learned policy $\tilde{\pi}$ starts from $(1,0) \in I_S$ and ends in $(3,4) \in F_S$.

B. PbD-LT$_k$

Grid-world state labels are interpreted by making each *sentence* as a single *literal* (location-based proposition), $\alpha_j = r_j \in S_{AP}$, $j \in \mathfrak{B}$ (e.g. if $s = (0,0)$, $L_M(s) = \alpha_A = r_A$, i.e. the agent is in the cell corresponding to partition A). When the task is modeled as language $\text{L}(G_{2\text{LT}})$, it comprises the strings expressing symbolic form of trajectories. They follow LT$_2$ patterns while avoiding the 2-factors corresponding to the prohibited behavior. The task grammar is $G_{2\text{LT}} = \{\{\rtimes\alpha_A, \alpha_A\alpha_A, \alpha_A\alpha_B, \alpha_B\alpha_B, \alpha_B\alpha_D, \alpha_D\alpha_D, \alpha_D\ltimes\}, \{\rtimes\alpha_B,$

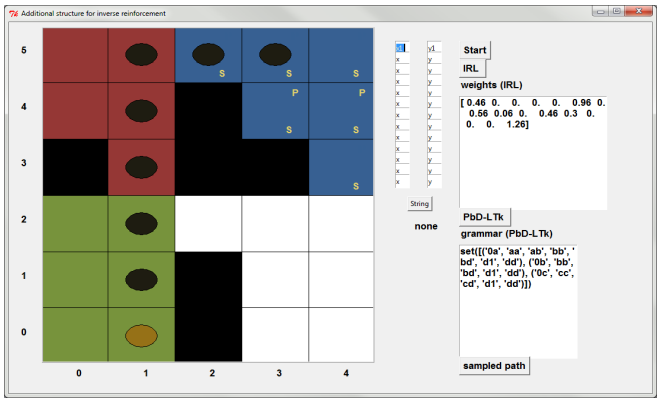

Fig. 4. Complete generalization. Cells with 'S' show the terminal states marked by PbD-LT$_k$. The task grammar $G_{2\text{LT}}$ ($\rtimes$ is 0 and $\ltimes$ is 1) and a path computed from M_P are shown.

$\alpha_B\alpha_B$, $\alpha_B\alpha_D$, $\alpha_D\alpha_D, \alpha_D\ltimes\}$, $\{\rtimes\alpha_C$, $\alpha_C\alpha_C$, $\alpha_C\alpha_D, \alpha_D\alpha_D$, $\alpha_D\ltimes\}\}$ over the extended alphabet $\mathbf{\Sigma} = \{\alpha_A, \alpha_B, \alpha_C, \alpha_D\} \cup \{\rtimes, \ltimes\}$ [3]. During PbD, a training demonstration is perceived as corresponding string $\text{txt}_\text{L}(i)$ of labels (Figures 1 and 2)[4]. PbD of $\text{L}(G_{2\text{LT}})$ results in M_p with terminal states $F_p = \{(s,q) \in S_p | s \in D\}$ (corresponding cells are marked 'S' in Figure 4).

C. Discussion

In a PbD problem, it is troublesome for a trainer to give the demonstrations abundant enough to ensure the generalization to all possible options for accomplishing (or terminating) the task being taught. The condition of providing a sufficiently 'diverse and extensive training' gets impracticable with an increase in the size of underlying state space. A worst case situation was replicated in previous sub-sections for an example with a small state space, where a trainer can demonstrate five trajectories ending in a particular subset of possible terminal states F_S. The situation is conceptually extrapolatable to a problem with large underlying state space. If the terminal states for satisfying the desired specification are not completely observed in demonstration, IRL will learn a limited options to terminate while meeting the task specification. As evident from figure 3, IRL converges to a solution path unique to F_S. Consider the terminal states IRL could not converge to, $D \backslash F_S = \{(2,5),(3,5),\ (4,4),(4,3)\}$ (see Figure 4 for cells in partition D without mark 'P'). These states will not contribute to the computation of the learned solution ($\tilde{\pi}$), thereby compromising its extent of application, especially regarding task termination.

PbD-LT$_k$ can generalize completely. It happens due to the structure introduced (to the learning framework) by a task model with a guaranteed convergence of learning and that introduced by the symbolic abstraction. The labels (sentences) divide the state space into groups of equivalent states satisfying them. For each such group of states, the information learned about the task is extrapolated to the whole group. In Gridworld example, the labels comprise of location based atomic propositions only (but the outlined approach applies to other propositions as well). Therefore, the groups of equivalent states are the partitions themselves. The sequences of partitions that accomplish the task are A $\rightarrow$ B$\rightarrow$ D, B$\rightarrow$ D, and C$\rightarrow$ D. If demonstrations has at least one physical trajectory from each of these representative sequences, the symbolic counterpart (text, Figure 1) maps to the sets of k-factors that are enough to converge to complete task language (Lemma 2 & Theorem 5 in [11]), and thereby DFA with all its accepting states. In product operation, the computation of terminal states F_p in M_p uses the complete state spaces S, Q, and F. The procedure includes the states the learner could not observe in training trajectories. Therefore, PbD-LT$_k$ does not lose any option for finishing the task (all cells marked 'S' in figure 4). The sampled path ends in the state $(3,5) \in S$ (as $s_p = ((3,5), \langle \alpha_A, (\rtimes\alpha_A, \alpha_A\alpha_A, \alpha_A\alpha_B, \alpha_B\alpha_B, \alpha_B\alpha_D, \alpha_D\alpha_D, \alpha_D\ltimes)\rangle) \in F_p$) which is not an option for solution by IRL (figure 4). PbD-LT$_k$ achieves a complete generalization to all the ways to accomplish learned task, which is difficult for IRL.

The reward (expressing π_E) in IRL is linear to make PbD problem solvable by linear programming [7], which makes the problem designer choose and prioritize the features. Similarly, PbD-LT$_k$ necessitates the selection of atomic propositions to create labels suitable for task. The choice of compensation depends on the nature of the PbD target.

A reason for the difference in the results above for PbD-LT$_k$ and IRL is that the desired specification in former is membership in a language, whereas that in latter is a maximum expected cumulative state value [12]. Consequently, there is a difference in the nature of respective tasks - a language comprising symbolically abstracted trajectories and a control law minimizing a cost. The characteristic qualities of set-theoretic grammatical inference can be ported to former but not latter.

D. Limitations of PbD-LT$_k$

The time complexity of PbD of LT$_k$ task is exponential, $O(|Q|+|F|+|Q|\times|S|)$, where $|Q| = x\cdot\ (2^{\left|\tilde{\text{I}}_k^{\text{se}}(\{\rtimes\}\cdot\Sigma^*\cdot\{\ltimes\})\right|} - 1)$ $= x(2^{(y+2z)} - 1)$, $x = (|\Sigma|+2)^{k-1}$, $y = |\Sigma|^k$, $z = |\Sigma|^{k-1}$, and $|F| = |G_{k\text{LT}}|$. Fortunately, the LT class of languages follow hierarchy: $\text{L}(G_{k\text{LT}}) \subset \text{L}(G_{(k-1)\text{LT}}) \ldots \subset \text{L}(G_{2\text{LT}})$, i.e. LT$_2$ includes each LT$_k$ with $k > 2$ (Theorem 98 in [19]). Therefore, the learner can use the demonstration for any task to learn it in LT$_2$ ($k = 2$) form, which bounds $|Q|$. PbD-LT$_k$ assumes prior knowledge. The existing methods have also used similar assumptions to improve learning [20]. In active learning, a privilege for asking on-line queries helps in reducing the sample complexity [10]. Reported method can bridge the gaps in prevalent PbD methods (lack of guaranteed generalization and need for linear reward). On these grounds, the level of restriction imposed in PbD-LT$_k$ is defendable.

[3]The task patterns are $\rtimes\alpha_A(\alpha_A)^l(\alpha_B)^m(\alpha_D)^n\alpha_D\ltimes$, $\rtimes\alpha_B(\alpha_B)^m(\alpha_D)^n\alpha_D\ltimes$, $\rtimes\alpha_C(\alpha_C)^m(\alpha_D)^n\alpha_D\ltimes$ $(l,m,n > 0)$

[4]Symbols shown in the string in Gridworld figure correspond to sentences, $\alpha_A = a, \alpha_B = b$, $\alpha_C = c, \alpha_D = d \in S_{AP}$. As no sub-grammar has α_A and α_C together, no $\text{txt}_\text{L}(i)$ contains the prohibited sequences "$\alpha_A\alpha_C$" or "$\alpha_C\alpha_A$" (A to C, C to A).

VI. CONCLUSIONS

Some tasks can be modeled as LT_k languages (e.g. *Pick large and small cups, place them on shelves avoiding large cups from going on small shelf, and small ones from going on the large shelf, eventually putting more large than small* is a LT_2 task). The method of PbD-LT_k learns such a task exactly correctly without losing potential solutions (completely generalized). This paper is a step toward application of grammatical inference in PbD. Potential future directions include using the languages capable of modeling everyday tasks and are identifiable in the limit (context-free language classes [1], [21]), or using a noisy demonstration [1].

VII. APPENDIX

function GRAPH-WITHOUT-$F(\Sigma)$
  $q^{'}$ =NEXT-STATE(q,σ,k)
  $Q = Q \cup q^{'}$
end function
function NEXT-STATE$(\langle w,g\rangle,\sigma,k)$
  $w^{'} = Sf^{(k-2)}(w)\cdot\sigma$
  if $|g| == 1$ **then**
    if $1 \le t < (k-1)$ **then**
      $g^{'} = \{Sf^{(k-2)}(w)\cdot\sigma\}$
    else
      $g^{'} = \{w\cdot\sigma\}$
    end if
  else
    $g^{'} = g \cup \{w\cdot\sigma\}$
  end if
  return $\langle w^{'}, g^{'}\rangle$
end function

Fig. 5. Learning (part 1): Computes Q for constructing graph w/o F

Given $(Q,\mathbf{\Sigma},T,I,\emptyset)$ **Output** $A = (Q,\mathbf{\Sigma},T,I,F)$, n
function EXTENDED-SEL($\mathrm{txt_L}(n)$)
  $n \leftarrow n+1$
  if $(\mathsf{f}_k^{\mathsf{se}})^{\diamond}(\mathrm{txt_L}(n)) \notin \phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[n-1])$ **then**
    $\phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[n]) \leftarrow \phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[n-1]) \cup (\mathsf{f}_k^{\mathsf{se}})^{\diamond}(\mathrm{txt_L}(n))$
    for all $q \in Q$, **do**
      if $g = (\mathsf{f}_k^{\mathsf{se}})^{\diamond}(\mathrm{txt_L}(n))$ **then**
        $F = F \cup \{q\}$
      end if
    end for
  else
    $\phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[n]) \leftarrow \phi_{k\mathrm{LT}}^{\mathsf{se}}(\mathrm{txt_L}[n-1])$
  end if
end function

Fig. 6. Learning (part 2): Hypothesizing task (DFA) by marking F

function PRODUCT$(A, M\backslash R)$
  for all $s \in S, q = \langle w,g\rangle \in Q$ **do**
    if $L_M(s) = Sf^1(w)$ **then** ▷ for $k > 1$
      $S_p = S_p \cup \{(s,q)\}$
      if $T(q,\ltimes) \in F$ **then**
        $F_p = F_p \cup s_p$
      end if
    end if
  end for
  for all $s_{p_1} = (s_1,q_1) \in S_p, a \in \mathscr{H}(s_1)$ **do**
    if $\exists s_2 = T_M(s_1,a), q_2 = T(q_1,\sigma), \sigma \in \Sigma$ **then**
      $s_{p_2} = (s_2,q_2)$
      if $s_{p_2} \in S_p$ **then**
        $T_p(s_{p_1},a) = T_p(s_{p_1},a) \cup s_{p_2}$
      end if
    end if
  end for
end function

Fig. 7. Computing product M_p

REFERENCES

[1] K. Lee, Y. Su, T. Kim, and Y. Demiris, "A syntactic approach to robot imitation learning using probabilistic activity grammars," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1323–1334, 2013.

[2] K. M. Kitani, Y. Sato, and A. Sugimoto, "Recovering the basic structure of human activities from noisy video-based symbol strings," *IJPRAI*, vol. 22, no. 8, pp. 1621–1646, 2008.

[3] A. Murali, S. Sen, B. Kehoe, A. Garg, S. McFarland, S. Patil, W. D. Boyd, S. Lim, P. Abbeel, and K. Y. Goldberg, "Learning by observation for surgical subtasks: Multilateral cutting of 3d viscoelastic and 2d orthotropic tissue phantoms," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 1202–1209.

[4] J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1969.

[5] J. Rogers and G. K. Pullum, "Aural pattern recognition experiments and the subregular hierarchy," *Journal of Logic, Language and Information*, vol. 20, no. 3, pp. 329–342, 2011.

[6] E. M. Gold, "Language identification in the limit," *Information and Control*, vol. 10, no. 5, pp. 447–474, 1967.

[7] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, 2000, pp. 663–670.

[8] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama, "Robot programming by demonstration with interactive action visualizations," in *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*, 2014.

[9] A. Boularias and B. Chaib-draa, "Apprenticeship learning with few examples," *Neurocomputing*, vol. 104, pp. 83–96, 2013.

[10] M. Lopes, F. S. Melo, and L. Montesano, "Active learning for reward estimation in inverse reinforcement learning," in *European Conference Machine Learning and Knowledge Discovery in Databases, ECML PKDD, Slovenia, September, Proceedings, Part II*, 2009, pp. 31–46.

[11] J. Heinz, "String extension learning," in *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*, 2010, pp. 897–906.

[12] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-first International Conference on Machine Learning*, 2004.

[13] N. D. Ratliff, D. M. Bradley, J. A. Bagnell, and J. E. Chestnutt, "Boosting structured prediction for imitation learning," in *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, British Columbia, Canada, December*, 2006, pp. 1153–1160.

[14] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer Handbook of Robotics*, 2008, pp. 1371–1394.

[15] T. Zeugmann, "From learning in the limit to stochastic finite learning," *Theoretical Computer Science*, vol. 364, no. 1, pp. 77–97, 2006.

[16] J. Fu and H. G. Tanner, "Bottom-up symbolic control: Attractor-based planning and behavior synthesis," *IEEE Transactions on Automatic Control*, vol. 58, no. 12, pp. 3142–3155, 2013.

[17] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT Press, 1998.

[18] A. Classen, P. Heymans, P. Schobbens, A. Legay, and J. Raskin, "Model checking lots of systems: efficient verification of temporal properties in software product lines," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, Cape Town, South Africa, 1-8 May*, 2010, pp. 335–344.

[19] J. Rogers, "Formal description of syntax," Lecture Notes from ESSLLI, ESSLLI, 07, 2007. [Online]. Available: www.cs.earlham.edu/ jrogers/files/reader.pdf

[20] J. Saunders, C. L. Nehaniv, and K. Dautenhahn, "Teaching robots by moulding behavior and scaffolding the environment," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*, 2006, pp. 118–125.

[21] A. Clark and R. Eyraud, "Polynomial identification in the limit of substitutable context-free languages," *Journal of Machine Learning Research*, vol. 8, pp. 1725–1745, 2007.