Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

Learning to Plan with Logical Automata

Brandon Araki[1, *], Kiran Vodrahalli[2, *], Thomas Leech[1, 3], Cristian-Ioan Vasile[1], Mark Donahue[3], Daniela Rus[1]
[1]MIT CSAIL, Cambridge, MA 02139, [2]Columbia University, New York City, NY 10027
[3]MIT Lincoln Laboratory, Lexington, MA 02421, [*]Authors contributed equally

***Abstract*—This paper introduces the *Logic-based Value Iteration Network* (LVIN) framework, which combines imitation learning and logical automata to enable agents to learn complex behaviors from demonstrations. We address two problems with learning from expert knowledge: (1) how to generalize learned policies for a task to larger classes of tasks, and (2) how to account for erroneous demonstrations. Our LVIN model solves finite gridworld environments by instantiating a recurrent, convolutional neural network as a value iteration procedure over a learned Markov Decision Process (MDP) that factors into two MDPs: a small finite state automaton (FSA) corresponding to logical rules, and a larger MDP corresponding to motions in the environment. The parameters of LVIN (value function, reward map, FSA transitions, large MDP transitions) are approximately learned from expert trajectories. Since the model represents the learned rules as an FSA, the model is *interpretable*; since the FSA is integrated into planning, the behavior of the agent can be *manipulated* by modifying the FSA transitions. We demonstrate these abilities in several domains of interest, including a lunchbox-packing manipulation task and a driving domain.**

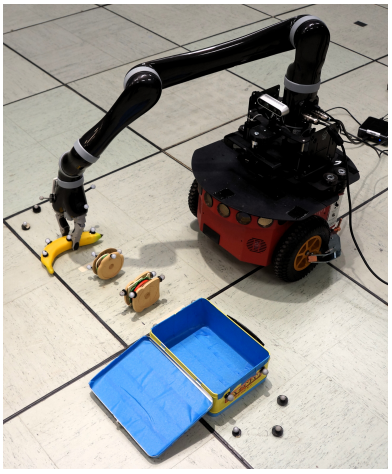


Fig. 1: The Jaco mobile robotic arm platform used in the experiments for lunchbox packing.

I. Introduction

In the imitation learning (IL) problem, desired behaviors are learned by imitating expert demonstrations [1, 11, 35]. IL has had success in tackling tasks as diverse as camera control, speech imitation, and self-driving for cars [42, 10, 19, 46]. However, an IL model trained to imitate a specific task must be re-trained on new expert data to learn a new task. Additionally, in order for a robot to correctly learn a task, the expert demonstrations must be of high quality: most imitation learning methods assume that experts do not make mistakes. Therefore, we ask

1) *How can expert demonstrations for a single task generalize to much larger classes of tasks?*
2) *What if the experts are unreliable and err?*

This paper provides answers to these questions by applying elements of formal logic to the learning setting. We require our policies to be derived from learned Markov Decision Processes (MDP), a standard model for sequential decision making and planning [5, 38]. We assume these MDPs can be factored into a large MDP that describes the motion of the robot in the physical environment, and, more importantly, a small finite state automaton (FSA) that corresponds to the rules the agent follows. After learning the transition and reward functions of the MDP and FSA, it is possible to manually change the FSA transitions to make the agent perform new tasks and to correct expert errors. Additionally, the FSA provides a compact symbolic representation of the policy.

For example, imagine the robotic arm in Fig. 1 packing first a sandwich and then a banana into a lunchbox. The physical environment and the motions of the robotic arm can be described by a "low-level" MDP. The rules the robot follows are described using FSAs. In the FSA, transitions are dependent on logical truth statements called *propositions*. In this environment there are three propositions – "robot has grasped sandwich", "robot has grasped banana", and "robot has dropped whatever it is holding into the lunchbox". The truth values of these propositions control transitions between the FSA's states, which we also refer to as logic states. For example, when "robot has grasped sandwich" is true, the FSA transitions from being in an initial state to being in a state in which "the robot has grasped the sandwich." When it is in this new state and "robot has dropped whatever it is holding into the lunchbox" is true, it transitions to the next state, "the robot has placed the sandwich into the lunchbox." We assume that the propositions correspond to locations in 2D space (e.g., we assume that the manipulator has a pre-programmed behavior to grasp a banana when it is in the vicinity of the banana and "robot has grasped banana" becomes true). This assumption enables us to factor the unknown MDP as the product of the high-level FSA and the low-level MDP. A simpler example of a product MDP is illustrated in Fig. 2.

The agent then learns approximate transitions and rewards associated with this product MDP, and generates a policy by running a planning algorithm. This approach has two benefits: 1) the learned policy is *interpretable* in the sense of learned FSA representations of rules, and 2) the behavior of the agent is *manipulable* because the rules that the agent follows can be changed in a predictable way by modifying the FSA's transitions. These benefits address the questions posed before: performing new tasks without re-learning and correcting faulty behaviour.

A. Outline of Our Approach

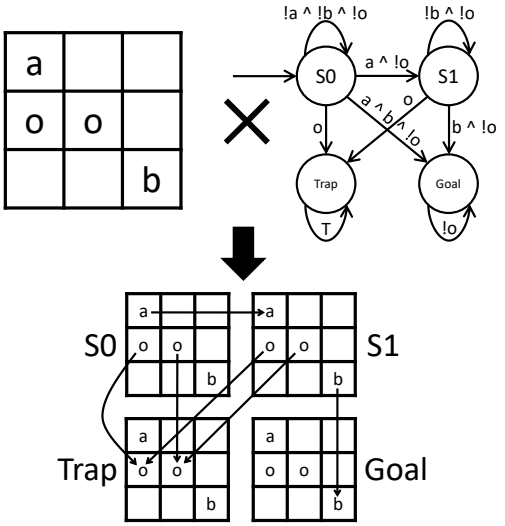


Fig. 2: An illustration of how an MDP and an FSA create a product MDP. The MDP is a 2D gridworld with propositions a, b, and o. The FSA describes the rules "go to a, then b, and avoid o. The resulting product MDP represents how these rules interface with the 2D gridworld.

Planning over Approximate MDPs Our model assumes that the MDP of the robot's behavior, also called a product automaton (PA), factors into a small, high-level FSA and a large, low-level MDP. When the FSA and MDP are known, one can find the optimal policy over the PA with standard planning methods [4, 26, 16, 40]. The model in this paper extends this approach to the IL setting where the transition and reward functions of the PA are unknown by learning an approximate MDP and then planning over the resulting MDP model. Exclusively during training we assume a *logic oracle* that the agent can query to learn its current FSA state. This assumption is more plausible and efficient to simulate compared to related works, which require knowledge of the full FSA [32, 45]. Our model, the Logic-based Value Iteration Network (LVIN), learns the relevant part of the transition matrix (TM) describing the FSA and directly integrates it into a differentiable recursive planning algorithm generalizing the Value Iteration Network (VIN) proposed in [41]. The key idea is to add a VIN module at each state of the FSA and link them together appropriately.

Logic Formalism In the robotics and control community, temporal logic languages such as Linear Temporal Logic (LTL) are used to unambiguously specify complex tasks, and a large and versatile class of these specifications can be directly translated into FSAs [22, 43, 6, 24, 25, 44, 30, 36]. In this paper, we assume that the high-level FSA is generated by an unknown LTL specification, although our methodology generalizes to any formal grammar that specifies FSAs. In fact, our model does not require that the expert demonstrations be generated from an FSA — it simply finds the best explanation that can be expressed as a product of an FSA and an MDP.

Experiments In the experiments, we demonstrate that for several gridworld tasks of varying complexity and a robot picking task (Fig. 1), our methodology allows us to *efficiently understand and modify* a robot's behavior. Our approach also solves tasks requiring *long sequences* of accurate actions, where we demonstrate standard learning approaches often fail. As another application, we *fix expert mistakes* without re-training on new data.

B. Contributions

1) We improve learning by attaching a logic oracle to the environment during training in the form of an FSA. Our logic oracle is required to produce less information than previous work (only FSA state, not the full transition matrix) and is thus more feasible to implement.
2) We introduce a differentiable planning model called the Logic-based Value Iteration Network (LVIN) which integrates an FSA into the recursive planning step of VIN. Using an imitation learning objective, we report considerable improvements over baselines in four different domains, including a robot picking task with real-world experiments.
3) We show that our framework can learn the transition matrix between FSA states, thus allowing us to *interpret* the logic rules that the model has learned.
4) We show how the learned transition matrix can be modified to *manipulate* the behavior of the agent to reliably perform other desired tasks without further learning. As a result, we can generalize to new tasks and fix the mistakes of unreliable experts without additional expert demonstrations.

II. Related Work

Logic-based Approaches Some recent work uses *logical structure* to make imitation learning and reinforcement learning (RL) problems problems easier. [32] uses LTL to define constraints on a Monte Carlo Tree Search. [28] and [18] use the product of an LTL-derived FSA with an MDP to make learning more efficient. In [21] the authors use LTL to design a sub-task extraction procedure as part of a more standard deep reinforcement learning setup. However, these methods assume the LTL specifications are already known, and [32, 33, 21, 18] do not allow for a model that is easy to interpret and manipulate. By contrast, our model only requires the current FSA state and the location of logic propositions in the environment.

Multi-task and Meta Learning We can also look at our method through the lens of multi-task and meta learning, methods which solve classes of tasks assuming a distribution over tasks [7, 2, 12, 14, 15]. LVIN is a model-based approach which can be viewed as sharing the structure of the low-level large MDP across tasks, while the high-level small FSA governs the task parameters. If we separated the learning of the FSA and the low-level MDP, we could learn the MDP across multiple tasks. Importantly, the FSA is human interpretable and manipulable, allowing us to change the task being solved with no new data (zero-shot), in contrast with one-shot methods like MAML [14] or [20] which require more data to adapt to new tasks from the task distribution.

Faulty Experts Other works tackle the problem of unreliable experts in imitation learning. [29] interpolates between imitation and intention learning with an entirely different approach based on inverse reinforcement learning, where transition dynamics are

known. [17] uses reinforcement learning with expert demonstrations, while our approach only requires easy and direct modification of an interpretable policy.

Hierarchical Learning LVIN is an instance of hierarchical learning: We can view the FSA as a high-level description of the tasks the agent must accomplish. The first instance of hierarchical learning was introduced in [31]. Related is the options framework of [39]. The idea is to temporally abstract out the kinds of actions you must take in a sequence, and to use sequences of these actions to specify policies. [2] applies the options framework and uses policy sketches, sequential strings of sub-policies from a sub-policy alphabet, to build an overall policy. Here, only actions have hierarchical structure while our method simplifies the entire MDP by providing a high-level view via the logic FSA.

In more recent work, [27] combines high-level imitation policies with low-level reinforcement learned policies to train more quickly. [23] builds in a hierarchy of planning with models of objects in the world rather than only considering low-level states, similar to the propositions in our model. Both models lack an interpretable transition matrix which can be easily modified to change policy behavior.

III. Problem Statement

Our goal is for agents to find *interpretable* and *manipulable* solutions to 2D gridworld tasks in the imitation learning setting. We define the finite gridworld state space $\mathcal{S}$ to consist of (row, column) pairs (r, c). The action space $\mathcal{A}$ consists of the 8 cardinal movement directions. We assume the agent is provided with expert demonstrations of a task (e.g., packing a lunchbox with many different configurations of initial conditions, including the locations of the food and the lunchbox). Additionally, the demonstrated task is assumed to be expressible as an LTL specification (see Sec. V-A), where *propositions* correspond to objects with semantic meaning in the environment, e.g., banana, lunchbox. We denote the set of propositions by $\mathcal{L}$ and include $\emptyset \in \mathcal{L}$ to denote "no proposition." $\mathcal{F}$ corresponds to states in an FSA (see Sec. IV) describing the rules of the environment. The propositions are assumed to have stationary locations in $\mathcal{S}$, determined by a known *proposition map* $\mathcal{M} : \mathcal{S} \to \mathcal{L}$. These locations can be obtained from sensors such as location sensors or pre-trained object detectors. We also assume access (exclusively during training) to the state $f \in \mathcal{F}$ of the FSA via a *logic oracle*. The agent must learn a policy $\pi : \mathcal{S} \to \text{dist}(\mathcal{A})$ (a distribution over $\mathcal{A}$) minimizing the cross-entropy loss $H(\pi_{\text{agent}}, \pi_{\text{expert}}) := \mathbb{E}_{a\sim\pi_{\text{agent}}}[\log \pi_{\text{expert}}(a)]$ with expert trajectories $\{\{(s_t, f_t, \ell_t, a_t)^{(n)}\}_{t=1}^{T}\}_{n=1}^{N}$ while being interpretable and manipulable, so a human can efficiently tweak the policy to achieve a different goal in the same state-action space (zero-shot learning). Here, $s_t \in \mathcal{S}, a_t \in \mathcal{A}, f_t \in \mathcal{F}, \ell_t \in \mathcal{L}$, T is the number of iterations per trajectory, and N is the number of trajectories. We reiterate that *interpretable* in this paper means that we learn an FSA (together with a policy) from expert trajectories of the unknown product automaton MDP. *Manipulable* means that we can obtain new policies without re-learning by modifying the learned specification FSA.

Feasibility of Assumptions Recognizing propositions in environments is a feasible assumption given recent advances in computer vision. In a driving environment, one could train a convolutional deep network to recognize objects of interest (traffic lights, other cars) independently of the driving task on potentially much larger datasets of images than those obtained while actually driving. Requiring only the current FSA state during training makes the oracle feasible to implement in practice *beyond simulation settings*, as the TM is often a difficult feature of the logic specification to design. One can often implement the FSA state oracle with a simple deterministic function of the environment. For instance, a self-driving car can identify geographic entities like `home` and `school` via GPS lookup.

IV. The Logic-based Value Iteration Network

As described in Section I, our method, the Logic-based Value Iteration Network (LVIN), (1) uses expert trajectories to learn an approximate MDP and then (2) plans on top of the learned MDP using value iteration, a standard dynamic programming method which solves known MDPs [5]. An MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$ corresponding to states, actions, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \text{dist}(\mathcal{S})$ (a distribution over state space), and a discount factor $\gamma \in (0, 1)$. LVIN learns the MDP parameters $\mathcal{R}$, $\mathcal{T}$, and γ, and the final policy is a lookup table commonly known as a Q-function [38]: each state s and action a are assigned a value $Q(s, a)$. The policy is $\pi(s) := \max_a Q(s, a)$.

We assume the state space of the approximate MDP we learn can be factored into two components: $\mathcal{S}$, the gridworld state space, and $\mathcal{F}$, the FSA states (defined by formulae of propositions). We thus learn a tuple $(\mathcal{S} \times \mathcal{F}, \mathcal{A}, \overline{\mathcal{T}}, \overline{\mathcal{R}}, \overline{\gamma})$. The transitions $\overline{\mathcal{T}}$ can be factored into the components $\overline{\mathcal{P}} : \mathcal{S} \times \mathcal{A} \to \text{dist}(\mathcal{S})$ and $\overline{TM} : \mathcal{F} \times \mathcal{L} \to \text{dist}(\mathcal{F})$ assuming that unique propositions define the transitions between states in $\mathcal{F}$, i.e., the FSA is deterministic. The FSA MDP is given by $(\mathcal{F}, \mathcal{L}, \emptyset, \overline{TM}, 1)$. Note $\overline{TM}$ depends on $\mathcal{S}$ through $\mathcal{M}(\mathcal{S}) = \mathcal{L}$.

Value-Iteration Network The Value Iteration Network (VIN) [41] is a neural network architecture used to find policies for MDPs. The key idea is to learn a neural net policy given value features generated by a value iteration procedure on an MDP learned from expert data that approximates the true MDP of the environment. The main insight is that the standard value iteration algorithm can be expressed in the form of a convolutional neural network. For known MDP $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, the value iteration updates are given by:

$$Q^{t+1}(s, a) \leftarrow \mathcal{R}(s, a) + \sum_{s' \in \mathcal{S}} [\gamma \mathcal{T}(s'|s, a)]\, V^t(s')$$
$$V^{t+1}(s) \leftarrow \max_a Q^{t+1}(s, a)$$

We interpret $[\gamma \mathcal{T}(s'|s, a)]$ as a convolution filter (the grid world dynamics are sparse), and the maximization over actions as a max-pooling step. By iterating these steps k times, we get a k-layer neural network. Since this procedure is differentiable, we can simultaneously learn a reward map $\overline{\mathcal{R}}$, a transition matrix $\gamma\overline{\mathcal{T}}$, the resulting value function $\overline{V}$, and a neural net policy over $\overline{V}$ simply by stacking each operation end-to-end and then backpropagating through the policy loss function (an imitation loss if there are experts, or an environment reward in the more general reinforcement learning setting) [41].

LVIN: Augmenting VIN with Logic We generalize VIN by assuming the factorization of the approximate MDP we learn into low-level and high-level components corresponding to $\mathcal{S}$ and $\mathcal{F}$, respectively. The constraints on proposition map $\mathcal{M}$ identify propositions with the actions of the FSA MDP (Fig. 3). We think of the resulting factorization as creating a separate VIN for each FSA state (Fig. 4). We learn reward map $\overline{\mathcal{R}}$ and transitions $\overline{\mathcal{T}}$ describing an MDP by backpropagating the imitation loss from the Q-value policy through the following network. For input (s, f, a), the output of the j^{th} layer of the LVIN network is:

$$\texttt{LVIN}_{j+1}(\gamma\bar{\mathcal{T}}, \bar{\mathcal{R}}, \bar{V}^j) := \langle \overline{Q}(s,f,a), \widehat{V}(s,f), \overline{V}(s,f)\rangle^{j+1} \tag{1}$$

$$\overline{Q}^{j+1}(s,f,a) := \overline{\mathcal{R}}(s,f,a) + \gamma \sum_{s' \in \mathcal{S}} \overline{\mathcal{P}}(s'|s,a)\overline{V}^j(s',f) \tag{2}$$

$$\widehat{V}^{j+1}(s,f) := \max_a \overline{Q}^{j+1}(s,f,a) \tag{3}$$

$$\overline{V}^{j+1}(s,f) := \mathbb{E}_{f' \sim \overline{\texttt{TM}}(\cdot|f,\mathcal{M}(s))}\left[\widehat{V}^j(s,f')\right] \tag{4}$$

$$\texttt{LVIN}(\gamma\bar{\mathcal{T}}, \bar{\mathcal{R}}, \{\bar{V}^j\}_{j=0}^k) := \texttt{LVIN}_k(\cdot,\cdot,\texttt{LVIN}_{k-1}(\cdot,\cdot,\cdots)) \tag{5}$$

Alg. 1 gives the full training algorithm. We augment the training of $\overline{\mathcal{T}}$ with an additional predictive task: given the current state, FSA state and an action, predict the next FSA state. This objective is meant to provide supervision to the task of learning the FSA transition matrix factor of the full transition dynamics. There are therefore two training losses: (1) a cross-entropy loss on next FSA state prediction, for learning the unknown FSA transition matrix, and (2) a cross-entropy loss on the action prediction. Cross-entropy between distributions p and q is denoted $H(p, q)$.

Algorithm 1 LVIN Multi-Trajectory Training

procedure LVIN-TRAINING
  Training Inputs: $\{\{(s_t, f_t, \mathcal{M}(s_t), a_t)^{(n)}\}_{t=1}^T\}_{n=1}^N$
  To learn:
    Transition matrix $\overline{\texttt{TM}} \in \mathbb{R}^{\mathcal{F}\times\mathcal{F}\times\mathcal{L}}$
    Low-level action kernels $\overline{\mathcal{P}}(\cdot|\cdot, a)$
    Value and Q functions $\overline{V}, \overline{Q}$
  Build the model $\texttt{LVIN}(\gamma\overline{\mathcal{T}}, \overline{\mathcal{R}}, \{\overline{V}^j\}_{j=0}^k)$:
    Normalize $\overline{\texttt{TM}}$ so that it is row-stochastic.
    for all $s \in \mathcal{S}$: $\mathcal{M}(s) \neq \emptyset$, all $f \in \mathcal{F}$ **do**
      $\overline{V}^{i+1}(s,f) := \mathbb{E}_{f' \sim \overline{\texttt{TM}}(\cdot|f,\mathcal{M}(s))}\left[\widehat{V}^i(s,f')\right]$
    end for
  for all $(s_t, f_t, \mathcal{M}(s_t), a_t)^{(n)}$ in data **do**
    Gradient update on $\overline{\texttt{TM}}$:
      loss $= H(f_{t+1}, \overline{\texttt{TM}}(\cdot|f_t, \mathcal{M}(s_t)))$
    Backpropagate the imitation loss through LVIN:
      loss $= H(a_t, \sigma_{\text{softmax}}(Q_{\texttt{LVIN}}(s_t, f_t)))$
  end for
end procedure

Avoiding System Identification Since we learn the MDP with an imitation loss and a predictive loss, we avoid the sample-inefficient system identification problem (learning the true MDP exactly as in model-based IL): we only care about approximate MDPs which result in good policies after planning (a similar property holds for VIN). Our approach therefore lies inbetween model-based and model-free IL and benefits from properties of both settings. Thus we expect to see some errors in $\overline{\mathcal{T}}$ and $\overline{\mathcal{R}}$ that barely affect the LVIN policy, leaving interpretability and manipulability intact.

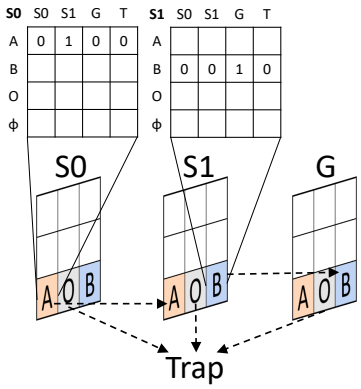


Fig. 3: The FSA transition matrix (learned by predicting the next FSA state) connects the value maps across FSA states (S0, S1, G, T) (see Eq. (4)). Each proposition (A: first goal, O: obstacle, B: second goal) is associated with a row of the learned TM for each FSA state based on $\mathcal{M}$.

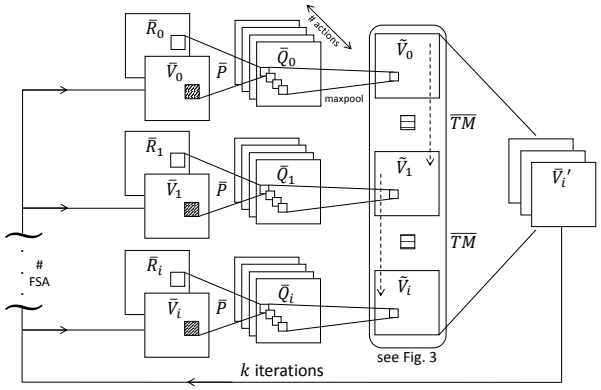


Fig. 4: LVIN forward pass: Each FSA state has a value map. $\overline{\mathcal{P}}(\cdot|\cdot, a)$ are shared across FSA states (can relax), and applied to produce Q-maps. Max-pooling yields updated value maps for each FSA state. The third layer output is depicted in detail in Fig. 3. The process is looped k times, see Eqs. (1)-(5).

V. Experiments and Results

We test LVIN against 2-3 baselines on 4 domains. As in [41, 3, 8], we consider gridworlds, which can express many complex tasks. (LVIN extends to any discrete environment (including dynamic ones) with more compute.) Each domain illustrates a key claim. The kitchen domain lends itself to in-depth analysis due to its simplicity. In the longterm domain, the agent must collect four keys and pass through four doors in sequence in order to reach the goal. Its complexity shows how the learned TM can be interpreted to understand the rules governing the agent's behavior even in cases where the actual FSA is difficult to understand,

as discussed in Sec. V-D. We show how to manipulate a TM in the pickworld domain, Sec. V-E. Lastly, the rules of the driving domain show how the TM can be modified to fix policies learned from demonstrations by faulty experts, Sec. V-F.

A. Generating Expert Data

Linear Temporal Logic We use linear temporal logic (LTL) to formally specify tasks [9]. Formulae ϕ constructed in LTL have the syntax grammar

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U} \phi_2 \tag{6}$$

where p is a *proposition* (a boolean-valued truth statement that can correspond to objects or goals in the world), $\neg$ is negation, $\vee$ is disjunction, $\bigcirc$ is "next", and $\mathcal{U}$ is "until". The derived rules are conjunction ($\wedge$), implication ($\implies$), equivalence ($\leftrightarrow$), "eventually" ($\Diamond\phi \equiv \texttt{True}\,\mathcal{U}\,\phi$) and "always" ($\Box\phi \equiv \neg\Diamond\neg\phi$), see [4] for details. Intuitively, $\phi_1 \mathcal{U} \phi_2$ means that ϕ_1 is true until ϕ_2 is true, $\Diamond\phi$ means that there is a state where ϕ is true and $\Box\phi$ means that ϕ is always true.

Generating Data We use the software packages SPOT [13] and Lomap [43] to convert LTL formulae into FSAs. Every FSA that we consider has a goal state, referred to in figures as G, which is the desired final state of the agent, as well as a trap state, referred to in figures as T, which is an undesired terminal state. For each domain, we generate a set of environments in which obstacles and other propositions are randomly placed. Given the FSA and an environment, we run Dijkstra's shortest path algorithm to create expert trajectories that we use as data for imitation learning.

B. Baselines

VIN: We compare the performance of LVIN to VIN. VIN cannot predict the next FSA state, nor can it learn a TM.

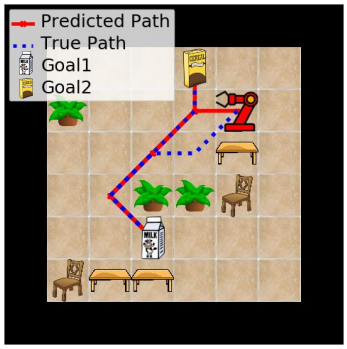
Hard-coded LVIN: It is not necessary to learn the TM from data – if the FSA is known, then the TM corresponding to the FSA can be used. We compare the performance of LVIN to LVIN with a hard-coded TM to see if learning the TM degrades performance.

CNN: We formulate a less constrained version of LVIN that uses a 3D CNN instead of a TM to transfer values between FSA states. A TM is also learned, but it is learned independently of the action predictions and is not used in planning. The CNN operation acts on a concatenation of the proposition matrix and the value function, returning the next iteration of the value function. The CNN has $|\mathcal{F}|$ input and $|\mathcal{F}|$ output channels. The kernel size is $(|\mathcal{L}| + |\mathcal{F}|, 1, 1)$, so the convolution operates on one cell (r, c) of the domain at a time, linearly combining the propositions and logic state values.

C. Environments

Kitchen Domain We first examine the kitchen domain (shown in Fig. 5a), an 8×8 gridworld with determnistic actions that enable movement to adjacent cells. The domain has three propositions: o for obstacle, a for milk, and b for cereal. The specification is $\Diamond(a \wedge \Diamond b) \wedge \Box\neg o$ – first fill the bowl with milk (visit a) and then put in the cereal (visit b) while avoiding randomly placed obstacles (chairs, tables, and plants).

The first two rows of Fig. 5b show that LVIN, hard-coded LVIN, and the CNN baseline all learn how to plan. The VIN baseline, however, has poor action prediction. Examining the VIN's failure modes reveals that since it has no record of which goal it has visited, the VIN model goes to the cereal without going to the milk almost as frequently as it visits both goals in the correct order. It also often visits only the milk without ever advancing to the cereal. VIN's performance highlights the importance of the FSA as a form of memory for tasks with sequential steps. We also note that the CNN baseline performs just as well as LVIN, indicating that explicitly integrating a TM into the planning step is not necessary for good performance.



(a) The 8×8 kitchen domain.

Kitchen Domain

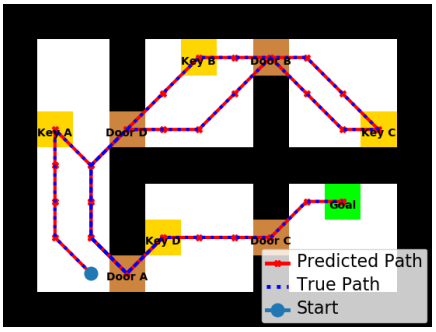
	LVIN	Hard-coded LVIN	CNN	VIN
Action Accuracy	98.85%	99.07%	98.29%	68.05%
FSA Accuracy	99.71%	99.73%	99.73%	N/A
Performance over 5000 rollouts				
Both Goals, Correct Order	99.84%	99.76%	99.20%	38.92%
Only Milk	0.02%	0.06%	0.00%	19.88%
Only Cereal	0.02%	0.00%	0.00%	34.10%
Both Goals, Wrong Order	0.02%	0.00%	0.74%	5.28%
No Goal	0.10%	0.18%	0.06%	1.82%

(b) Performance of LVIN and baselines in the kitchen domain.

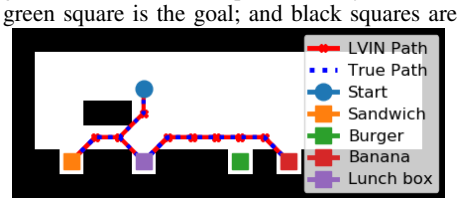
Fig. 5

Longterm Domain The longterm domain is a 12×9 gridworld and shows LVIN's ability to learn complex sequential specifications. In this environment (Fig. 6a) there are 10 propositions: keys ka, kb, kc, kd that unlock doors da, db, dc, and dd, respectively; and g for the goal and o for obstacles. To progress to the goal, the agent must follow the specification $\Diamond g \wedge \Box\neg o \wedge (\neg da\ \mathcal{U}\ ka) \wedge (\neg db\ \mathcal{U}\ kb) \wedge (\neg dc\ \mathcal{U}\ kc) \wedge (\neg dd\ \mathcal{U}\ kd)$ that involves learning a "longterm" plan – it must first pick up Key A, then go get Key D, then Key B, then Key C, before it can access the room in which the goal is located. The results in Table Ia show that while LVIN and the CNN baseline have good performance, VIN cannot complete a single rollout successfully, which is unsurprising given the lengthy sequential nature of this domain.

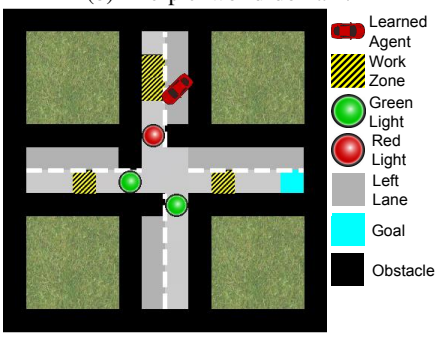
Pickworld Domain The pickworld domain is an 18×7 gridworld where the agent must first pick up either a sandwich a or a burger b and put it in a lunchbox d, and then pick up a banana c and put it in the lunchbox d. The specification is $\Diamond((a \vee b) \wedge \Diamond(d \wedge \Diamond(c \wedge \Diamond d))) \wedge \Box\neg o$. As shown in Table Ib, VIN



(a) The longterm domain. Yellow squares are keys; brown squares are doors; the green square is the goal; and black squares are obstacles.



(b) The pickworld domain.



(c) The driving domain.

Fig. 6

cannot complete a single successful rollout due to the sequential nature of the domain.

Driving Domain The driving domain (Fig. 6c) is a 14×14 gridworld with the goal of showcasing LVIN's ability to learn and encode rules, in this case three "rules of the road." The model must learn three rules of the road – reach the goal (g) and avoid obstacles (o): $\Diamond g \wedge \Box \neg o$; prefer the right lane over the left lane (l): $\Box\Diamond\neg l$; and stop at red lights (r) until they turn green (e): $\Box(r \Rightarrow (r \ \mathcal{U} \ e))$. Interestingly, the VIN baseline slightly outperforms the other baselines (see Table Ic). This is because there is only one "sequential" goal, which is to reach the goal g. Otherwise, the rules of the road can be encoded into the VIN's reward function. However, since the VIN does not encode the TM, it can only learn the rules implicitly, whereas the LVIN and CNN baselines can learn the rules explicitly. This allows a user to check that safe rules have been learned, as discussed in Sec. V-F.

D. Interpretability

One benefit of learning the TM, whose rows and columns correspond to specific FSA states and propositions, is that its values can be assigned meaningful interpretations. Consider the longterm domain. The specification has 10 propositions and translates to an FSA with 33 states. With no knowledge of

Longterm

	LVIN	CNN	VIN
Action Accuracy	99.49%	98.82%	66.16%
FSA Accuracy	100.00%	99.40%	N/A
Performance over 1000 rollouts			
Success Rate	100.00%	82.80%	0.00%

(a) Performance of LVIN and baselines on the longterm domain

Pickworld

	LVIN	Hard-coded LVIN	CNN	VIN
Action Accuracy	99.67%	99.46%	99.06%	59.68
FSA Accuracy	100.00%	100.00%	100.00%	N/A
Performance over 1000 rollouts				
Success Rate	83.20%	83.20%	71.90%	0.00%

(b) Performance of LVIN and baselines on pickworld

Driving

	LVIN	Hard-coded LVIN	CNN	VIN
Action Accuracy	99.35%	99.38%	99.10%	99.39%
FSA Accuracy	100.00%	99.93%	87.94%	N/A
Performance over 1000 rollouts				
Success Rate	99.60%	98.40%	98.60%	99.90%

(c) Performance of LVIN and baselines on driveworld

TABLE I

S0	da	db	dc	dd	ka	kb	kc	kd	g	o	ϕ
S0									1		1
S1					1						
S2			1								
S3											
S4											
G											
T	1	1		1		1	1	1		1	

(a) In the initial state, Door A is not allowed, and Key A leads to state $S1$

S1	dd	kd
S0		
S1		
S2		1
S3		
S4		
G		
T	1	

(b) In $S1$, Key D leads to $S2$.

S2	db	kb
S0	1	
S1		
S2		
S3		1
S4		
G		
T		

(c) In $S2$, Key B leads to $S3$.

S3	dc	kc
S0		
S1		
S4		
S3		
S2		1
G		
T	1	

(d) In $S3$, Key C leads to $S4$.

S4	g
S0	
S1	
S2	
S3	
S4	
G	1
T	

(e) $S4$ leads to goal.

TABLE II: The learned transition matrix of the longterm domain. Cells of interest are highlighted in yellow; unexpected values are highlighted in red. Propositions in a given state that are never encountered are shaded in gray.

the specification it would be difficult to tell what is going on. However, the structure of the domain forces the agent to pick up the keys in a specific order. Therefore the agent visits only 6 of the 33 states (7 including the trap state). Since LVIN learns the TM of this reduced FSA, after training we can examine the TM to reconstruct the FSA and learn the rules of the system. Table II contains parts of the TM of the longterm domain learned by

LVIN. Each table is associated with one FSA state and specifies how each proposition maps to the next FSA state.

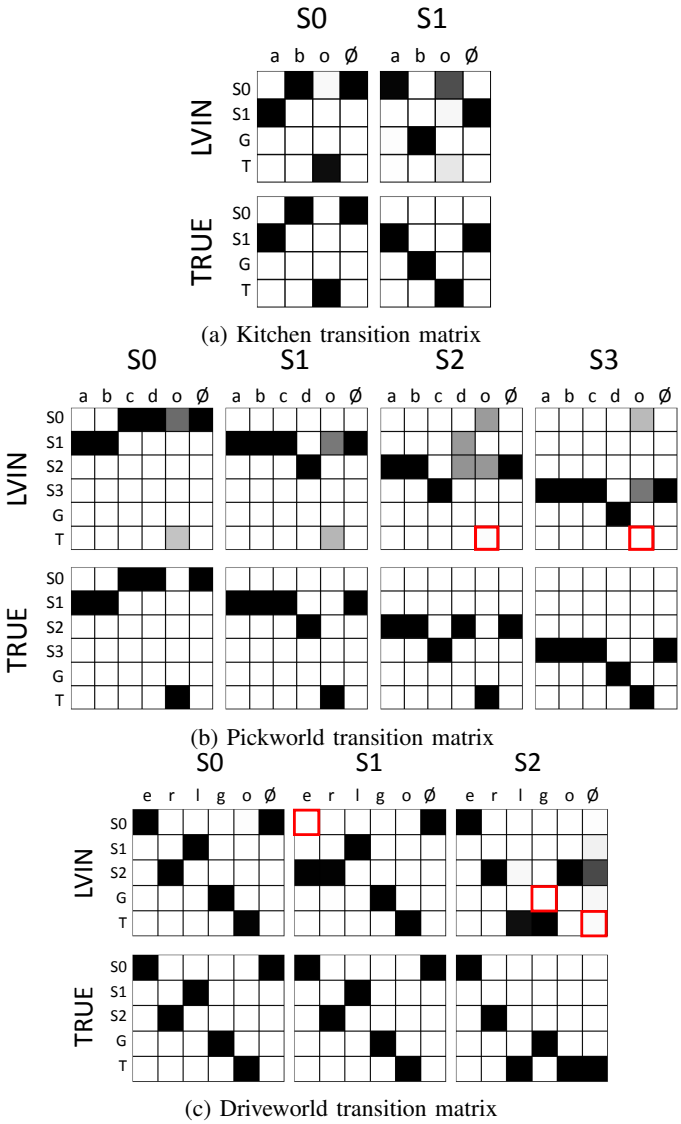
In the TM of the initial state $S0$, Table IIa, we see that all doors and keys map to the trap state, except for Key A, which maps to state $S1$. In other words, the model has learned that when in $S0$, the agent is not permitted to travel through doors and that it must pick up Key A before any other key. Partial TMs for the other states show that the model has learned a sequence of keys to pick up, and that it cannot pass through the door associated with its key until it has picked up the key. Unexpected transitions are highlighted in red. In every case, unexpected transitions occur where the model has not actually observed a transition (columns highlighted in grey) but rather had to infer the value. This is understandable because in these cases the model picks a value that helps it reduce its loss, which may not coincide with the true TM.

The learned and true TMs of the other domains are shown in Fig. 7, and they can be analyzed in a similar way to the longterm domain's TM. Note that the trap and goal states are not shown because they are trivial. For the kitchen and pickworld domains, the states are ordered sequentially – for the kitchen domain (Fig. 7a), the goal in $S0$ is to reach a, and the goal in $S1$ is to reach b. In the pickworld domain (Fig. 7b), the goal in $S0$ is to reach either a or b; the goal in $S1$ is to reach d; the goal in $S2$ is to reach c; and the goal in $S3$ is to reach d again. For the driveworld domain (Fig. 7c), $S0$ corresponds to when the car is in the right lane, driving towards goal g. $S1$ corresponds to when the car is in the left-hand lane, and $S2$ corresponds to when the car is at a red light.

E. Manipulability

Since we can interpret the TM layer, we can also modify it to change the behavior of the agent in a predictable way. To demonstrate, we manipulate the TM that was learned in the pickworld domain. The learned TM tells the robot to pick up either the sandwich or the burger, put it in the lunchbox, and then pick up the banana and put it in the lunchbox; $\phi_{p1} = \Diamond((a \vee b) \wedge \Diamond(d \wedge \Diamond(c \wedge \Diamond d))) \wedge \Box \neg o$. However, whoever is having their lunch packed may have a preference between sandwich and burger. Let "only sandwich, then banana" be ϕ_{p2} and "only burger, then banana" be ϕ_{p3}. As another example, a user may prefer the banana to be packed before the sandwich/burger. Let "banana, then sandwich/burger" be ϕ_{p4}.

The modifications to the TM are shown in Fig. 8. To make the agent pick up only the sandwich, we modify the TM's initial state $S0$ so that b (the burger) maps back to $S0$ instead of the next state $S1$ (Fig. 8a). Similarly, to pick up only the burger, we modify $S0$ so that a (the sandwich) maps back to $S0$, as shown in Fig. 8b, and the sandwich is ignored. Lastly, to modify the TM to pick up the banana first (Fig. 8c), we first modify $S0$ so that a and b are ignored and c (the banana) leads to $S1$. In $S1$, the agent's goal is to drop its payload into the lunchbox. We then modify the next state, $S2$, so that a and b are the goals of $S2$ and c, the banana, is ignored. These modifications result in the robot picking up the banana in $S0$ and picking up the sandwich or burger in $S2$. The results of these changes can be seen in Table III. The first two columns show that LVIN and the CNN



(a) Kitchen transition matrix

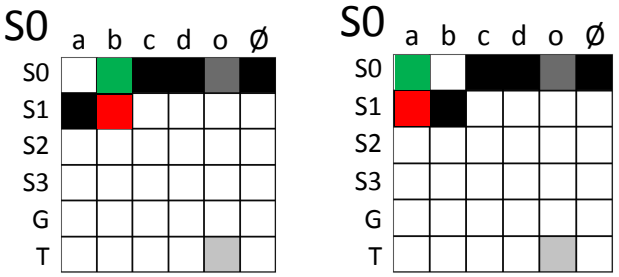
(b) Pickworld transition matrix

(c) Driveworld transition matrix

Fig. 7: Learned vs. true transition matrices. The tiles are shaded in grayscale to denote the probability of a transition (black corresponds to 1 and white to 0). Red outlines denote what LVIN should have learned if it learned an incorrect transition.

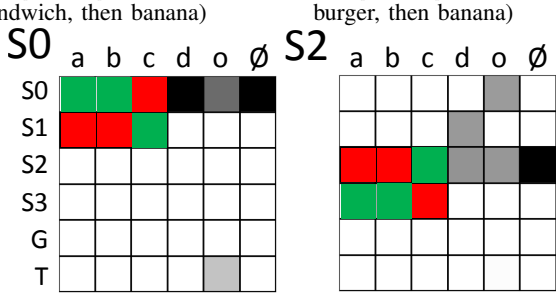
baseline trained directly on these new specifications successfully learn them. The second pair of columns shows the performance of LVIN and the CNN baseline when they are initially trained on the original specification, ϕ_{p1}, and then have their TMs modified to represent the new specifications. Modified LVIN has equivalent performance to the LVIN model trained directly on the new specifications. By contrast, the modified CNN model performs poorly because the TM is not integrated into the planning step for the CNN. Instead, the CNN baseline attempts to perform the old specification. The tests highlight an important shortcoming of the CNN model: Since the TM is not directly incorporated into the planning step, modifying the TM does not change the behavior of the agent.

Manipulability Experiments on Jaco Arm To show how LVIN can be applied to the real world, we implemented the algorithm



(a) $\phi_{p1} \rightarrow \phi_{p2}$ (pick up only sandwich, then banana)

(b) $\phi_{p1} \rightarrow \phi_{p3}$ (pick up only burger, then banana)



(c) $\phi_{p1} \rightarrow \phi_{p4}$ (pick up banana, then sandwich/burger)

Fig. 8: Modifications to the learned transition matrix of the pickworld domain so that the agent follows a new specification. Deleted values are marked in red, and added values in green.

		LVIN	CNN	Mod. LVIN	Mod. CNN
Performance over 1000 rollouts					
ϕ_{p2}	Sandwich-to-Burger Ratio	1.0	1.0	1.0	0.58
ϕ_{p3}	Burger-to-Sandwich Ratio	1.0	1.0	1.0	0.43
ϕ_{p4}	Success Rate	89.80%	90.60%	95.00%	1.10%

TABLE III: The performance of models using the modified transition matrices for ϕ_{p2}, ϕ_{p3}, and ϕ_{p4}.

	ϕ_{p1}	ϕ_{p2}	ϕ_{p3}	ϕ_{p4}
Success Rate	18/20	19/20	18/20	20/20
Failure Modes	2/20 banana slipped out of hand	1/20 banana slipped out of hand	1/20 banana slipped out of hand 1/20 bad path	N/A

TABLE IV: Performance of Jaco arm over 20 trials on each pickworld specification.

on a Jaco arm, shown in Fig. 1. The Jaco arm is a 6-DOF arm with a 3-fingered hand and a mobile base. An Optitrack motion capture system was used to track the hand and the four objects of interest (a plastic banana, sandwich, burger, and lunchbox). The system was implemented using ROS [34]; the Open Motion Planning Library (OMPL) [37] was used for motion planning for the arm. The motion capture system was used to translate the positions of the hand and the lunch items into a 2D grid corresponding to Fig. 6b, and an LVIN model trained on simulated data was used to generate a path satisfying the specification. The LVIN model trained on ϕ_{p1} was then "manipulated" as discussed to follow specifications ϕ_{p2}, ϕ_{p3}, and ϕ_{p4}. 20 trials were run for each specification with the Jaco arm, and the results are shown in Table IV. The arm was largely successful in carrying out the planned trajectories – 4 of the 5 failures were caused by the banana falling out of the hand, and only 1 of the 80 trials failed because of LVIN generating a bad path.

F. Fixing Expert Errors

Our interpretable and manipulable model can also be used to fix the mistakes of faulty experts. Suppose the real-world driving data contains bad behavior from drivers breaking the rules. We model this scenario in Table V, where the Unsafe TM shows a scenario in which the model has learned to run a red light 10% of the time. This result can be observed directly from the TM, since the probability of entering the initial state given that the agent is on a red light is 10%, meaning it will ignore the red light, while the probability of recognizing that it is in a "red light" state is 90%. We correct the TM by setting the initial state entry to 0 and the red light state entry to 1. We perform 1000 rollouts using each of these TMs. The Unsafe TM results in the agent running 9.88% of red lights while the Safe TM prevents the agent from running any red lights.

Unsafe TM

Initial State	red light
Initial	0.1
Left Lane	0.0
Goal	0.0
Red Light	0.9
Trap	0.0

Safe TM

Initial State	red light
Initial	0.0
Left Lane	0.0
Goal	0.0
Red Light	1.0
Trap	0.0

Rollout Performance

Unsafe TM	9.88%
Safe TM	0.00%

TABLE V

VI. Conclusion

Developing interpretable and manipulable models that learn to plan is an ongoing goal in deep policy learning. By learning an FSA transition matrix *in conjunction* with a planning module, we were able to build a model that a human can *control* intuitively. As a result, the model immediately generalizes to new task specifications, can be fixed in the presence of expert errors, and can also solve long-term planning tasks requiring higher-level state transition information than the environment provides. Future work will focus on improving the poor scaling of computation with state and action space size, continuous state space versions of LVIN, and removing the logic oracle assumption during training time.

Acknowledgments

NSF grant 1723943, ONR grant N000141812830, and the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001 supported this project. K.V. is supported by an NSF GRFP fellowship. Any opinions, findings, conclusions or recommendations in the paper do not necessarily reflect the views of the funding agencies.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. *ICML '04 Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[2] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *ArXiv e-prints*, 2016.

[3] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. 2018.

[4] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

[5] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996.

[6] A. Bhatia, L.E. Kavraki, and M.Y. Vardi. Sampling-based motion planning with temporal goals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2689–2696. IEEE, 2010.

[7] Rich Caruana. Learning many related tasks at the same time with backpropagation. *Advances in Neural Information Processing Systems*, pages 657–664, 1995.

[8] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *arXiv preprint arXiv:1810.08272*, 2018.

[9] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2001. ISBN 978-0-262-03270-4.

[10] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[11] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Journal of Machine Learning*, 75:297–325, 2009.

[12] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *ArXiv e-prints*, 2016.

[13] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016. doi: 10.1007/978-3-319-46520-3_8.

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[15] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[16] Jie Fu and Ufuk Topcu. Probably approximately correct mdp learning and control with temporal logic constraints. *ArXiv e-prints*, 2014.

[17] Yang Gao, Huazhe (Harry) Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[18] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-correct reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[19] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[20] De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Fei-Fei Li, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *ArXiv e-prints*, 2018.

[21] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*, 2018.

[22] S. Karaman and E. Frazzoli. Sampling-based Motion Planning with Deterministic μ-Calculus Specifications. In *IEEE Conference on Decision and Control (CDC)*, Shanghai, China, December 2009.

[23] Ramtin Keramati, Jay Whang, Patrick Cho, and Emma Brunskill. Strategic object oriented reinforcement learning. 2018.

[24] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[25] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's Waldo? Sensor-based temporal logic motion planning. In *IEEE International Conference on Robotics and Automation*, pages 3116–3121, 2007.

[26] Morteza Lahijanian, Sean B Andersson, and Calin Belta. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409, 2012.

[27] Hoang M Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. *ArXiv e-prints*, 2018.

[28] Xiao Li, Yao Ma, and Calin Belta. Automata guided hierarchical reinforcement learning for zero-shot skill composition. *ArXiv e-prints*, 2017.

[29] James MacGlashan and Michael L. Littman. Between imitation and intention learning. *IJCAI'15 Proceedings of the 24th International Conference on Artificial Intelligence*, pages 3692–3698, 2015.

[30] MR Maly, M Lahijanian, Lydia E. Kavraki, H. Kress-Gazit, and Moshe Y. Vardi. Iterative Temporal Motion Planning for Hybrid Systems in Partially Unknown Environments. In *International Conference on Hybrid Systems: Computation*

and Control (HSCC), pages 353–362, Philadelphia, PA, USA, March 2013. ACM.

[31] Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049, 1998.

[32] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. *ArXiv e-prints*, 2017.

[33] Chris Paxton, Yotam Barnoy, Kapil Katyal, Raman Arora, and Gregory D. Hager. Visual robot task planning. *ArXiv e-prints*, 2018.

[34] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[35] S. Ross, G. Gordon, and J. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 15:627–635, 2011.

[36] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The International Journal of Robotics Research*, 2017.

[37] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. http://ompl.kavrakilab.org.

[38] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[39] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: Learning, planning, and representing knowledge at multiple temporal scales. *Journal of Artificial Intelligence Research*, 1:1–39, 1998.

[40] M. Svorenova, I. Cerna, and C. Belta. Optimal control of mdps with temporal logic constraints. In *IEEE 52nd Annual Conference on Decision and Control (CDC)*, pages 3938–3943, 2013.

[41] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems 29*, pages 2154–2162, 2016.

[42] Sarah Taylor, Taehwan Kim, Yisong Yue, Moshe Mahler, James Krahe, Anastasio Garcia Rodriguez, Jessica Hodgins, and Iain Matthews. A deep learning approach for generalized speech animation. *ACM Transactions on Graphics (TOG)*, 36(4):93, 2017.

[43] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, Calin Belta, and Daniela Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8):889–911, 2013.

[44] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding Horizon Temporal Logic Planning for Dynamical Systems. In *Conference on Decision and Control (CDC) 2009*, pages 5997–6004, 2009. doi: 10.1109/CDC.2009.5399536.

[45] Saining Xie, Alexandre Galashov, Siqi Liu, Shaobo Hou, Razvan Pascanu, Nicolas Heess, and Yee Whye Teh. Transferring task goals via hierarchical reinforcement learning. 2018.

[46] Yisong Yue and Hoang Le. Imitation learning tutorial. Tutorial at ICML 2018, 2018. URL https://sites.google.com/view/icml2018-imitation-learning/home.