Teaching and Executing Verb Phrases

Daniel Hewlett, Thomas J. Walsh, Paul Cohen
Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA
{dhewlett,twalsh,cohen}@cs.arizona.edu

***Abstract*—This paper describes a framework for an agent to learn models of verb-phrase meanings from human teachers and combine these models with environmental dynamics to enact verb commands. The framework extends prior work in apprenticeship learning and leverages recent advancements in modeling activities and planning in domains with multiple objects. We show how to both learn a verb model as a relational finite state machine and how to turn this model into reward and heuristic functions that can then be composed with an MDP model of an environment. The resulting "combined model" can then be efficiently searched by a planner to enact a verb command in this environment. Our experiments in simulated robot domains show this framework can be used to quickly teach verb commands and improves over the current state of the art method.**

I. Introduction

Humans use verb phrases in a variety of ways, including giving commands ('jump over the hurdle') and relaying descriptions of events to others ('he leapt over it'). These simple verb phrases can actually convey a significant amount of content, including descriptions of events and conditions that should not occur, such as 'avoid the puddle.' Humans have a qualitative understanding of verbs that supports executing the same verb in multiple contexts, and with different arguments. Humans can also fluidly combine verbs, either as part of novel commands such as 'cross the street and avoid the puddle', or to define new verbs ('fetch' is a series of simpler verbs). In this work we show how a robot can learn (through interaction with a human teacher) executable models of verb phrases that possess these important verb properties.

In this work, we present a framework for learning a novel representation of verb phrases that supports both recognition and execution, as well as the specification of constraints. This verb model, called the *Verb Finite State Machine* or VFSM, represents sequences of qualitative states, so it supports the natural "stages" of verb completion, the composition of verb meanings, and application to different physical (or even metaphorical) environments. Specifying an agent's objective as a verb phrase with this representation allows us to learn and enact behaviors that would be difficult to capture using traditional AI encodings such as goals (traditional planning) or reward functions based on the state of an environment (reinforcement learning). For instance, the meaning of a verb phrase like 'walk over the hill' is not found in feedback from the environment, which has no notion of the verb command, or in a goal location, which does not specify *how* an agent got there.

We begin by describing the verb learning problem in more detail, and then describe the VFSM representation in Section III. Section IV describes our teaching protocol and a learning algorithm for constructing the VFSM from teacher demonstrations and feedback. We then explain how to derive a reward and heuristic function from the VFSM, and how to combine these with a model of the dynamics of the environment to plan a policy to enact the verb, in Section V. Together, these components provide an end-to-end framework for learning verb meanings from human teachers and executing verb commands. We demonstrate the success of our system for several verbs in simulated robot domains, including an example that bootstraps a new verb meaning through composition, in Section VI. On the verbs we tested, our approach substantially outperforms a recent method for learning and executing verb-phrases [1].

II. Framework Overview

Formally, we consider a set of environments E where each $e \in E$ has a set of objects $\mathcal{O}$ and a starting configuration (or distribution over start states) for the low-level properties (e.g x and y coordinates) of the objects and the agent. In addition, we assume that the agent is equipped with a qualitative understanding of the environment, specifically *relations* between objects using a set of *relational fluents* F where every grounded fluent is either true or false at a given timestep. Together, these attributes and fluents make up the *state* of the environment. This two-level representation (object attributes and relations) provides a qualitative description of the environment, without assuming that the dynamics can be captured at a purely relational level. In the physical environments most relevant for robotic agents, these relations typically describe spatial relationships such as *LeftOf*(X,Y), roughly corresponding to prepositional phrases.

Objectives for an agent will be communicated (by a teacher as described below) via verb phrases such as 'go around the block', which we will represent formally as *go-around*(Agent, Block). We consider a set of available verb phrases V such that $v \in V$ has a meaning based on a series of configurations of fluents in F that are either true or false on each timestep. For instance, Figure 1 illustrates (as a pair of finite state machines) the verb *fetch*, where different stages of the verb are triggered by changes in the truth values of these fluents.

Learning in our system involves incremental refinement of verb semantics through interactions with a human teacher, a process that extends work from the apprenticeship learning literature [2]. Formally, the protocol for each episode is as follows.

1) The teacher can choose an environment $e \in E$, and a verb $v \in V$ and ask the agent to execute v in e.

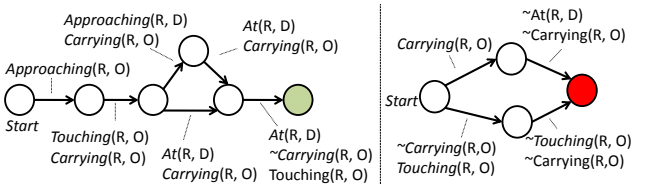

Fig. 1. An example VFSM (shown for clarity as two machines, one for acceptance and one for rejection) for *deliver*(Robot,Object,Destination). The left machine accepts "successful" examples while the right one accepts "violations".

2) The agent then uses its verb model along with a planner to produce and execute a policy that enacts this verb phrase. The human teacher can then label this execution as *successful*, *incomplete* (for partial performance of the verb), or *violation* (of the verb's meaning).
3) If the agent's behavior was judged by the teacher to be incomplete or a violation, the teacher can provide a *demonstration* of the verb in the same environment. Both the agent's and the teacher's labeled trajectories can be used by the agent to refine its verb model.

The overall goal of the teacher is to teach the verb-phrase semantics to the agent in a complete enough form that the agent's chance of succeeding during subsequent teaching episodes is maximized.

III. Verb Representation

Our representation for verb meanings is a finite-state machine we will refer to as a Verb FSM (VFSM) (Figure 1). Conceptually, this VFSM is a combination of two simpler FSMs, M_v^+ representing correct verb behavior, and M_v^-, encoding behavior that violates the semantics of the verb. The VFSM differs from standard FSMs in that each edge is labeled with a set of propositions (relational fluents as described earlier), rather than a single symbol. Also, each intermediate (non-start and non-terminal) state contains a loop back to itself (omitted from the diagram for clarity), allowing parts of the verb (or constraints) to take varying amounts of time. The utility of using a similar FSM for recognition of activities has been previously established [3], showing that the FSM transitions capture the natural stages and choices involved in verb completion.

Importantly, the VFSM is *not* a finite-state controller (FSC), where each state is mapped directly to an action. Rather, the VFSM accepts qualitative traces that match the verb semantics. Additional steps needed to execute the verb described by the VFSM are presented in Section V. The VFSM is a qualitative representation in that it represents only propositional information, and only information about the ordering of states rather than their duration. It is also underspecified because each transition specifies only a subset of the propositions comprising the environment state. In addition, all of the propositions in the VFSM are expressed as relations over the arguments of the verb rather than specific objects (e.g., *At*(R,D) rather than *At*(robot1,dest7)). The full verb representation is the lexical form of the verb and its argument structure (e.g., *pick-up*(Agent,Object)), together with the VFSM.

Formally, the VFSM can be defined by a tuple $\langle S, \Sigma, s_0, \delta, A_p, A_n \rangle$, where S is the set of propositional states, s_0 is the start state, and A_p and A_n are the sets of positive and negative accepting states. Σ, the "alphabet" is theoretically all the possible sets of true fluents, though we do not have to enumerate all of these to define the transition function. Structurally, the VFSM is a DFA, meaning there are no epsilon transitions and each transition out of a state bears a unique label. However, since transitions are labeled with sets rather than individual symbols, a given set of fluents may be compatible with more than one outgoing transition, meaning that the VFSM transition function operates non-deterministically. The current state of the VFSM is thus given by a set of FSM states, and the transition function δ maps each set of states and set of fluents to a set of possible next states. δ is defined as follows: If the machine is in the set of states S_i and the set of fluents F is observed, the new state of the machine is a set containing every state s_j such that some $s_i \in S_i$ contains a transition labeled with a set of fluents σ such that $\sigma \subseteq F$.

Intuitively, this non-determinism allows multiple ways to perform the verb (and multiple constraints) to be "active" at one time. For instance, if an agent is trying to achieve the verb 'jump over' but has started to turn away from the target object and is not jumping, multiple violations need to be watched for. The agent's trajectory can be tracked through the VFSM until it reaches a positive accepting state, indicating that the verb was successfully completed, or a negative accepting state, indicating that a violation has occurred. If no accepting state is reached, the trajectory is deemed *incomplete*.

IV. Learning Verb Meanings from Teachers

Our algorithm for updating the VFSM based on a labeled trace of behavior (as collected in the protocol from Section II) is shown in Algorithm 1. The algorithm manages the storage of examples with each label, eliminating traces that are subsumed by others, and uses a conservative FSM construction from only the instances labeled as "success" or "violation". This cautious approach was taken based on results in apprenticeship learning showing that acting based on the most specific hypothesis allows the teacher to safely drive the generalization process [2].

The trimming of examples (lines 6 through 13) is a form of FSM inference that relies on the notion of *core paths*. A core path is a sequence of sets of relations that lead from the initial state to a terminal state. Intuitively, each element of $\mathcal{I}_v^+$ and $\mathcal{I}_v^-$, containing positive and negative examples, respectively, represents a specific hypothesis for one of the core paths. Initially, these core paths may be longer than needed. For instance, if the verb "jump over" was demonstrated by an agent getting a running start, jumping over the object, and running more afterwards, the initial core path would contain all the relation sets from this sequence, when really all that is needed are the relation sets from the middle portion.

Each agent or teacher trace yields a sequence of states, each composed of ground attributes and relational fluents. For

Algorithm 1 UpdateVerb(v, τ, l)

1: **Initially**: $\mathcal{I}_v^+$ and $\mathcal{I}_v^{\text{inc}}$and $\mathcal{I}_v^- = \emptyset$
2: **Input**: A verb v, execution trace τ, and label l.
3: Let $\mathcal{I} = \mathcal{I}_v^l$ (dataset for label l)
4: **if** $\exists c \in \mathcal{I}$ where c is a subsequence of τ **then**
5: Return //The instance is already explained.
6: **else if** $\mathcal{I} = \mathcal{I}_v^+$ or $\mathcal{I} = \mathcal{I}_v^-$ **then**
7: Let $C = \{c \in \mathcal{I}$ where τ is a subsequence of $c\}$
8: **if** $C = \emptyset$ **then**
9: Add τ to $\mathcal{I}$ as a new core path
10: **else**
11: $\mathcal{I} = (\mathcal{I} \setminus C) \cup \{\tau\}$
12: **end if**
13: **end if**
14: Rebuild M_v^+ and M_v^- from $\mathcal{I}_v^+$ and $\mathcal{I}_v^-$, respectively, by building a trie from all the examples, and then minimizing. Merge these machines to create the final VFSM.

learning the VFSM, only the relational part of each state is considered, so each trace is represented as a sequence of sets of relations. When a new trace τ arrives, if one of the core paths already explains it, (line 4) nothing is done. If τ is not a subsequence of any of the current core paths, τ is added into $\mathcal{I}$ as indicated by its label (line 9). The final case (line 11) is that that τ is a subsequence of some subset of the core paths $C \subseteq \mathcal{I}$, meaning that there is a contiguous sequence of relation sets $F_{i...i+|\tau|}(c)$ in each $c \in C$ such that $F_j(\tau) \subseteq F_j(c)$ for $j = i...i+|\tau|$ and F_j is the set of relational fluents true at time j. In such a case, each matching core path c is replaced by the smaller core path τ. In this way, vestigial components such as the run-up and run-after parts of "jump over" are excised from the core paths.

To build the actual VFSMs, we construct a trie (line 14) out of all the examples in $\mathcal{I}_v^+$ (or $\mathcal{I}_v^-$ for M_v^-), ignoring the highly uninformative examples in $\mathcal{I}_v^{\text{inc}}$. This conservative construction means the agent will be tied to the most specific hypothesis of a verb meaning supported by the current examples in $\mathcal{I}^+$ and $\mathcal{I}^-$. DFA minimization can be performed on this trie [4] to reduce the size of the state space without changing the set of sequences it accepts. Minimization of a trie also ensures that the resulting DFA has precisely one accepting state, since the algorithm merges states with the same set of outgoing transitions. This property will be desirable for combining verbs.

A. Learning Verbs by Definition

In addition to learning by demonstration, our verb representation supports bootstrapping complex verbs out of simpler ones by composition. Effectively, this allows the teacher to define a new verb by reference to existing verbs. Because the VFSM is a finite state machine, VFSMs also support standard FSM operations, including alternation ('go around it or go over it') and concatenation ('go get it, then bring it to me'). For example, since each VFSM has one start state and one positive accepting state, concatenation of two VFSMs machines is achieved by merging the A_p from the first VFSM with s_0 of the second VFSM. We explore a concatenation example in Section VI-B.

V. Executing Verbs

In this section we describe how the VFSM can be combined with a representation of the robot's environment and modern planning techniques to execute the verb. We begin by describing a model from the reinforcement-learning literature that captures both relational information and low-level robot dynamics. Our algorithm for planning and executing of a verb phrase is given in Algorithm 2.

A. Object-Oriented Markov Decision Processes

A standard Markov Decision Process (MDP) [5] model $M = \langle S, A, T, R, \gamma \rangle$ is comprised of states, actions, a transition function, rewards, and a discount factor. The long-term *value* of a state can be described by the optimal value function: $V^*(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')V(s')$. In order to leverage the VFSM model, a robot needs a relational view of its environment, which could in principle be provided by any relational MDP. However, since the dynamics of complex physical environments cannot be captured by purely relational models (like STRIPS), we will use a two-level model called an object-oriented MDP (OOMDP) from [6]. In an OOMDP, each state consists of a set of objects with attributes, as well as a set of relations between the objects. The set of relations are defined based on object attributes (such as *On*(X, Y) := X.yCoord = Y.yCoord +1). In an OOMDP, the agent's actions have stochastic affects on object attributes, which in turn may causes changes to the relational state.

B. Combining the VFSM and OOMDP Models

Recall that the VFSM does not directly encode a policy for verb execution. Thus, to plan for verb execution, we must combine the dynamics model for the environment M_e (an OOMDP) and the qualitative verb model M_v (a VFSM). This corresponds to line 2 in Algorithm 2. This requires combining the state spaces and transition functions of the two models and using the terminal states of the VFSM to indicate reward. Specifically, we build a combined MDP $M_C = \langle S_C, A, T_C, R_C, \gamma \rangle$ where A and γ come from the OOMDP but the states $S_C = S_E \times S_V$ are any pairing of an OOMDP environment state and a set of states in the VFSM. The transition function T_C is then:

$$T_C(s_C, a, s'_C) = T(s'_e|a, s'_e)\mathbb{I}(\delta(s_v, P[s'_e]) = s'_v) \quad (1)$$

where $P[s]$ denotes the propositions that are true in s and δ comes from the VFSM. The reward function is based only on the completion of the verb, 0 for states in A_p from M_v, and otherwise -1. Formally, the reward function can be stated as:

$$R_C(s_C) = -\mathbb{I}(s_v \notin A_p \vee s_v \in A_n) \quad (2)$$

Note that the reward function here depends solely on the VFSM. Incorporating the environment's reward function R_e is possible, but requires weighting the agent's desire to complete

the verb versus maximizing reward in the environment itself. While this would be useful for an agent to, say, consider its own safety while completing the verb, it is a complex topic beyond the scope of this work.

To mitigate the size of the combined state space and the sparsity of reward, we would like to leave some "breadcrumbs" throughout the state space so that small rewards are given for completing each stage of a verb. A simple mechanism for encoding such subgoals is to initialize the values of each state ($V(s_C)$) using a *heuristic function* $\Phi(s_C)$. We use the heuristic function $\Phi(s_C) = -\rho(s_v)$, where $\rho(s_v)$ is the shortest distance in the VFSM from s_v to an accepting state in M_v^+ without reaching an accepting state in M_v^-. This is the minimum number of stages remaining in the VFSM to successfully complete the verb activity. This heuristic draws the planner's search towards areas where it can progress through stages of the verb, but will not stop it from backtracking if the currently explored branch does not allow for the verb's completion in the current environment.

C. Planning in the Combined Space

We now have a fully specified combined MDP $M_C = \langle S_C, \mathcal{A}, T_C, R_C, \gamma \rangle$, as well as a heuristic function $\Phi(s_C)$. To actually perform the verb, the agent requires a planner that can map states in M_C to actions. When the environment is deterministic, we can use the cost and heuristic functions constructed above with simple A* search [7] to find the optimal policy. But if the MDP dynamics are stochastic we will need to use a more general planner.

If the MDP is small enough, then standard MDP planning algorithms like Value Iteration (VI) [5] could be used. However, because OOMDPs usually have a large ground state space to begin with, and since $|S_C|$ contains the cross product of all of these states with all of the S_v states, the computational dependence of algorithms like VI on $|S_C|$ is likely to be prohibitive. Instead, our experiments with stochastic environments employ Upper Confidence for Trees (UCT) [8], a sample-based planner that sidesteps a dependence on $|S_C|$ by only guaranteeing to produce a policy for the start state s_0 (hence it may need to be called at every timestep, as in line 6). While UCT is not expressly built to utilize a heuristic function such as our $\Phi(s_C)$ described above, we simply translated this heuristic into a *reward shaping* function [9] that served to guide UCTs search to areas where stages of the verb could be quickly completed.

VI. Experiments

We evaluated the performance of the VFSM on a set of 4 verbs, and against baseline methods described later in this section. The verbs tested were the following:

- *go*(Robot,Target): Travel to the target location.
- *go-via*(Robot,Waypoint,Target): Travel to the target location via the waypoint location.
- *deliver*(Robot,Object,Target): Travel to the object, pick it up, then travel to the target and place the object there.

Algorithm 2 Planning, Executing, and Updating with an OOMDP and VFSM

1: **Input**: An OOMDP M_e, a VFSM M_v, Environment e with initial state s_{e0}, and horizon H
2: Create combined MDP M_C with $S_C = (S_e \times S_v)$ and T_C from (1) and R_C from (2)
3: Create the heuristic function $\Phi(\langle s_e, s_v \rangle) = -\rho(s_v)$
4: $t = 0$, $s_t = \langle s_{e0}, s_{v0} \rangle$
5: **while** the s_v component of s_t is not terminal and $t < H$ **do**
6: a_t = Planner.recommend(M_C, Φ, s_T)
7: Execute a_t in e, observe s'_e
8: Query $M_v(s_v, a_t, s'_e)$ for s'_v
9: $s_{t+1} = \langle s'_e, s'_v \rangle$
10: $t = t + 1$
11: **end while**
12: Use the teacher's label l of τ to call UpdateVerb(v,τ,l)
13: If the teacher provides a demonstration trace and label $\langle \tau', l' \rangle$, UpdateVerb($v$,$\tau'$,$l'$)

- *intercept*(Robot,Enemy,Target): Make contact with the enemy robot before it reaches the target.

Our experiments used two simulated mobile robot domains: the Gazebo robot simulator[1], where we ensured that the robot's actions had deterministic effects, and the Wubble World 2D (WW2D) simulator[2], where actions had stochastic effects. Both environments contained similar objects (robots, blocks, and locations) and relations (mostly spatial relations such as DistanceDecreased or InFrontOf).

As a baseline for execution, we also implemented the Maximum Likelihood verb model of Kollar et al. [1], which we will refer to as ML. ML proceeds by iteratively simulating all possible sequences of actions up to some depth (a breadth-first search), and then executing the sequence that maximizes the likelihood of the verb under a Naive Bayesian model. This process terminates when all possible actions would decrease the likelihood. ML only models the percentage of time that each relation is true, and assumes all relations are independent of each other. We also implemented the Inverse Reinforcement Learning (IRL) method of Abbeel and Ng [10] as a baseline. We discuss the suitability of IRL for learning verb models in Section VI-C.

The teaching protocol defined in Section II provides natural opportunities for evaluation. During each teaching episode, the robot is asked to perform the verb in a situation it has not encountered before, which is a form of hold-one-out evaluation. From many learning trajectories, we can estimate the probability of success after a given number of teaching episodes. This measure of performance for the VFSM and the ML baseline at each teaching episode is shown in Figure 2. The order of presentation by the teacher was randomized for each learning trajectory. Success is determined by an automatic

[1]http://playerstage.sourceforge.net/
[2]http://code.google.com/p/wubbleworld2d/

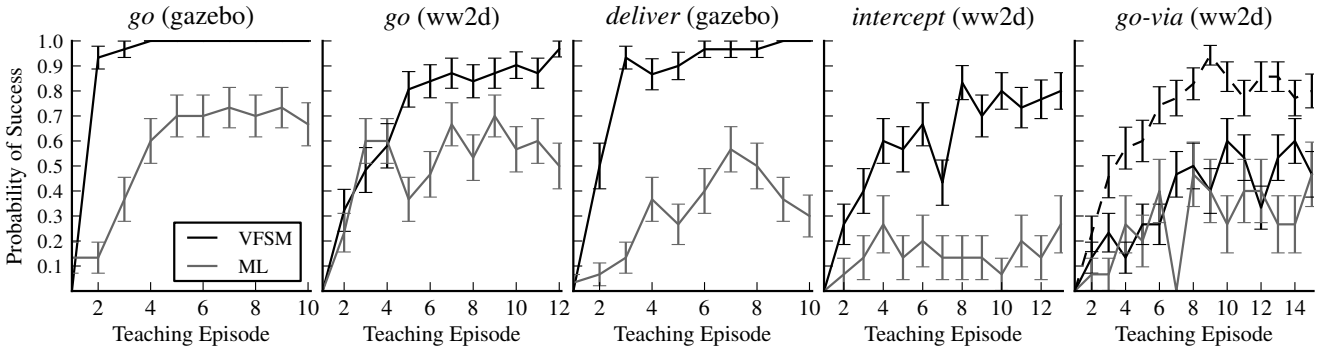

Fig. 2. Experimental results for execution and recognition of verb phrases. Error bars show one standard deviation (n $\geq$ 15). The dashed line shows the performance of a bootstrapped version of *go-via*, an example of verb composition discussed in Section VI-B.

validation procedure for each verb.

A. Learning Verbs

We tested all three algorithms on the verb phrases listed above. The simplest verb, *go*, was tested in both domains, Gazebo and WW2D. In each case, the VFSM was able to master the verb, achieving a high rate of success after only a few training examples, as shown in Figure 2. However, the ML baseline was not able to match the performance of the VFSM, reaching a plateau around 70% success rate, somewhat lower than the 90% success rate reported by Kollar et al. [1]. While the VFSM explicitly models the completion of a verb, ML instead relies on a decrease in the likelihood function for termination, meaning that it does not always stop at the goal, reducing its success rate.

VFSM significantly outperforms ML on the more complex verb *deliver*, which is naturally described by a sequence of stages. As previously discussed, the VFSM can represent this sequence of stages, and provide incremental feedback to a planner based on the state of underlying FSM. The failure of ML to master *deliver* is not unexpected, as Kollar et al. reported a low rate of success (29%) for the similar verb *bring* [1]. However, the reasons for this failure underscore the importance of the sequential nature of the VFSM for modeling verbs. ML struggles with *deliver* because the features it is trained on are order-independent averages over full traces, making it difficult to model the sequential semantics of *deliver*. Since ML attempts to match these averages, it frequently performs behaviors inappropriate for the current stage of verb execution. For example, when given the object at the start of the verb, it will typically put it down and wait briefly, so that the *Holding*(R,O) relation will not be true "too often."

We also examined a more dynamic verb *intercept* in which the enemy is another agent moving along a known path to a target location. This is perhaps the most difficult verb because the agent must act effectively in a constantly changing stochastic environment. As shown in Figure 2, the VFSM representation outperformed the ML baseline significantly on this verb. Unlike the VFSM, the ML method does not learn a model of verb constraints, meaning it cannot learn from its failures. Also, its reliance on a full BFS means it can look ahead only a limited number of steps, reducing the precision of control and thus failing to catch the enemy more often. By contrast, planners like UCT consider full trajectories without (in general) searching all possible action sequences.

B. Bootstrapping Verbs

Our next experiment concatenates two VFSMs to compose the meaning of a new verb, as described in Section IV-A. We compare two ways of arriving at the verb *go-via*: Teaching the verb directly, and defining *go-via* in terms of the simpler verb *go*. Figure 2 compares the performance of the agent when taught *go-via* directly and bootstrapping *go-via* from *go*. Even at a relational level, there are several possible configurations of the waypoint and final destination, each of which will become a path in the VFSM when *go-via* is learned directly, slowing learning. By contrast, the teacher can simply define *go-via*(X,Y) as follows: *go*(Y) then *go*(X). With *go-via* defined this way, the teacher can teach the simpler verb *go* and the agent's ability to perform *go-via* will improve. The dashed line in Figure 2 shows the performance on *go-via* after varying numbers of exposures to *go*. The agent learns to perform the verb reliably, and with much lower average planning time (roughly 8 vs. 80 seconds), since the bootstrapped VFSM is much smaller. This example demonstrates how combining smaller verbs can accelerate learning of verbs by allowing the agent to quickly bootstrap the core verb semantics.

C. Comparison to Inverse Reinforcement Learning

We also considered the Inverse Reinforcement Learning (IRL) method of Abbeel and Ng [10] as a baseline, but our examination of it revealed properties not well suited to a general verb representation. Because IRL makes a linear cost assumption, its behavior on a "goal oriented" verb like *go* was highly dependent on the amount of "padding" at the end of an example showing the agent sitting at the destination. With a large amount of padding the agent successfully completed all test instances with one teacher trace, but without padding, even with all the training data, it failed in all the test examples because it preferred *approaching* the goal to actually *reaching* it. For a verb like *deliver*, IRL without padding had trouble finding a weight for *Carrying*, since this was a "good" relation

as long as the agent was not at the destination (a non-linear relationship). With padding, performance improved, although if there was another way to get the object to the destination (perhaps triggering a violation), the agent might prefer this method. These results indicate that while IRL can learn certain verbs very quickly, it cannot capture the full range of complicated (non-linear) verb definitions.

VII. Related Work

A number of previous works (e.g. [3], [11]) have considered the problem of learning classifiers or recognizers for verb phrases from situated language data. Our work differs from these approaches because we provide a framework for executing the verbs. For verb execution, other than the work of Kollar et al., the most relevant work is in the fields of Robot Learning from Demonstration (LfD) [12] and various subfields of reinforcement learning. Our verb-model inference and transformation to a reward function can be considered a form of inverse reinforcement learning (IRL) [10], but most IRL methods assume the cost function is linear in the feature space, making it hard to uncover verb stages that are triggered by specific sequential combinations of features. Our VFSM-based cost function is an example of a *non-Markovian reward function*, however other recent work in this area focuses on logical formalisms for planning with such rewards (e.g., [13]), rather than learning the cost functions themselves, as we have in this work. In the classical planning literature, Temporally-Extended Goals (TEGs) [14] capture some temporal and sequential aspects of verb meanings. However, the TEG literature generally assumes deterministic and fully symbolic domains.

Another approach is to learn a policy directly from teacher demonstrations, forgoing the inference of a reward function. Various representations have been used to represent such an inferred policy (e.g., finite state controllers [15] and decision lists [16]). but all such approaches explicitly specify which action to take from each state, resulting in a more restrictive model than our VFSM, which only specifies relational states that must be achieved and therefore can lead to different grounded policies for enacting the same verb.

VIII. Future Work and Conclusions

In this work we have shown how to combine a teaching protocol for learning verbs from humans with dynamics models to create an interactive system for teaching and executing verb phrases. This method learns a verb model that engenders a cost and heuristic function for use with a planner. This end-to-end system for learning and executing verb commands considerably outperforms other recent methods, while maintaining several properties of verbs, such as stages of execution, qualitative description, and composability.

There is still much work to be done in verb learning. The space of possible verbs is large and highly diverse, so defining exactly the set of verbs that can be modeled effectively by a VFSM is an important challenge. In addition, the potential to use the learned VFSMs for recognition has not been explored in this work. An exploration strategy may be especially useful for learning precise violation conditions (important for verbs such as *avoid*), since it is difficult for the teacher to correctly anticipate violations the student is likely to perform. In addition, while VFSMs can naturally represent verbs with primarily sequential structure (e.g., *fetch*), verbs with complex looping/conditional structures (e.g., *patrol*) may pose a challenge to the current system.

There are a number of extensions for future investigation. First, we assumed throughout this work that the OOMDP (M_e) was known and that the reward function of the ground environment R_e could be replaced by a uniform step-cost function. These decisions were made to highlight the verb learning and execution portion of our work, but prior work [2] has shown such relational models can be learned from teacher demonstrations; so a version of our system that learns both the verb semantics and low-level dynamics is a natural next step. In addition, incorporating R_e into M_C (in Equation 2) could allow the agent to reason about physical costs when performing a verb.

References

[1] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Grounding Verbs of Motion in Natural Language Commands to Robots," in *International Symposium on Experimental Robotics*, 2010.

[2] T. J. Walsh, M. L. Littman, and C. Diuk, "Generalizing Apprenticeship Learning across Hypothesis Classes," in *Proceedings of the Twenty-Seventh International Conference on Machine Learning*, 2010.

[3] W. Kerr, A. Tran, and P. Cohen, "Activity Recognition with Finite State Machines," in *International Joint Conference on Artificial Intelligence*, 2011.

[4] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[5] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley and Sons, 1994.

[6] C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[7] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2003.

[8] L. Kocsis and C. Szepesv, "Bandit Based Monte-Carlo Planning," in *European Conference on Machine Learning*, 2006.

[9] E. Wiewiora, "Potential-Based Shaping and Q-Value Initialization are Equivalent," *Journal of Artificial Intelligence Research*, vol. 19, pp. 205–208, 2003.

[10] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*. New York, New York, USA: ACM Press, 2004.

[11] S. Tellex, G. Shaw, N. Roy, and D. Roy, "Grounding Spatial Language for Video Search," in *International Conference on Multimodel Interfaces*, 2010.

[12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, 2009.

[13] S. Thiebaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza, "Decision-Theoretic Planning with non-Markovian Rewards," *Journal of Artificial Intelligence Research*, vol. 25, pp. 17–74, 2006.

[14] F. Bacchus and F. Kabanza, "Planning for temporally extended goals," *Annals of Mathematics and Artificial Intelligence*, vol. 22, pp. 5–27, 1998.

[15] D. Grollman and O. Jenkins, "Incremental Learning of Subtasks from Unsegmented Demonstration," in *Proceedings of the International Conference on Intelligent Robots and Systems*. Citeseer, 2010, pp. 261 – 266.

[16] S. Yoon, A. Fern, and R. Givan, "Inductive Policy Selection for First-Order MDPs," in *Uncertaintly in Artificial Intelligence*, 2002.