Using Advice in Model-Based Reinforcement Learning

Rodrigo Toro Icarte
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
rntoro@cs.toronto.edu

Toryn Q. Klassen
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
toryn@cs.toronto.edu

Richard Valenzano
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
rvalenzano@cs.toronto.edu

Sheila A. McIlraith
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
sheila@cs.toronto.edu

Abstract

When a human is mastering a new task, they are usually not limited to exploring the environment, but also avail themselves of *advice* from other people. In this paper, we consider the use of advice expressed in a formal language to guide exploration in a model-based reinforcement learning algorithm. In contrast to constraints, which can eliminate optimal policies if they are not sound, advice is merely a recommendation about how to act that may be of variable quality or incomplete. To provide advice, we use Linear Temporal Logic (LTL), which was originally proposed for the verification of properties of reactive systems. In particular, LTL formulas can be used to provide advice to an agent about how they should behave over time by defining a temporal ordering of state properties that the agent is recommended to avoid or achieve. LTL thus represents an alternative to existing methods for providing advice to a reinforcement learning agent, which explicitly suggest an action or set of actions to use in each state.

We also identify how advice can be incorporated into a model-based reinforcement learning algorithm by describing a variant of R-MAX which uses an LTL formula describing advice to guide exploration. This variant is guaranteed to converge to an optimal policy in deterministic settings and to a near-optimal policy in non-deterministic environments, regardless of the quality of the given advice. Experimental results with this version of R-MAX on deterministic grid world MDPs demonstrate the potential for good advice to significantly reduce the number of training steps needed to learn strong policies, while still maintaining robustness in the face of incomplete or misleading advice.

Keywords: Markov Decision Process, Reinforcement Learning, Model-Based Learning, Linear Temporal Logic, Advice.

Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC). The first author also gratefully acknowledges funding from CONICYT (Becas Chile).

1 Introduction

Reinforcement Learning (RL) methods can often be used to build intelligent agents that learn how to maximize long-term cumulative reward through interaction with the environment. Doing so generally requires extensive exploration of the environment, which can be infeasible in real-world environments in which exploration can be unsafe or requires costly resources. Even when there is access to a simulator and exploration is safe, the amount of interaction needed to find a reasonable policy may be prohibitively expensive.

In this paper we investigate the use of *advice* as a means of guiding exploration. Indeed, when humans try to master a new task, they certainly learn through exploration, but they also avail themselves of linguistically expressed advice from other humans. Here we take "advice" to be recommendations regarding behaviour that may describe suboptimal ways of doing things, may not be universally applicable, or may even contain errors. However, even in these cases people often extract value and we aim to have RL agents do likewise. We use advice to guide the exploration of an RL agent during its learning process so that it more quickly finds useful ways of acting. Advice can be contrasted with hard constraints, the imposition of which potentially eliminates optimal policies if the constraints are incorrect or suboptimal.

We use the language of Linear Temporal Logic (LTL) for providing advice. The advice vocabulary is drawn from state features, together with the linguistically inspired temporal modalities of LTL (*e.g.*, *"Turn out the lights before you leave the office"* or *"Always avoid potholes in the road"*). We propose a variant of the standard model-based RL algorithm *R-MAX* that adds the ability to use given advice to guide exploration. Experimental results on randomly generated grid worlds demonstrate that our approach can effectively use good advice to reduce the number of training steps needed to learn a strong policy. Our approach is robust in the face of incomplete or misleading advice, as we can guarantee convergence to an optimal policy in deterministic settings and convergence to a near optimal policy in non-deterministic environments.

Related Work: Maclin and Shavlik [6] proposed the use of advice in RL via a procedural programming language that supports recommendation of actions using `If-Then` rules and loops. Krening *et al.* [5] grounded natural language advice into a list of *(object,action)* recommendations (or warnings), to encourage (or discourage) an agent to perform an action when it interacts with particular classes of objects. Neither of these methods incorporate advice into a model-based RL algorithm. Moreover, our LTL advice language is less prescriptive: it allows a user to provide a rich temporal ordering of state properties that the agent may wish to avoid or achieve, as opposed to explicitly specifying the actions to take in a given state. This allows for alternative forms of advice to those considered by earlier works.

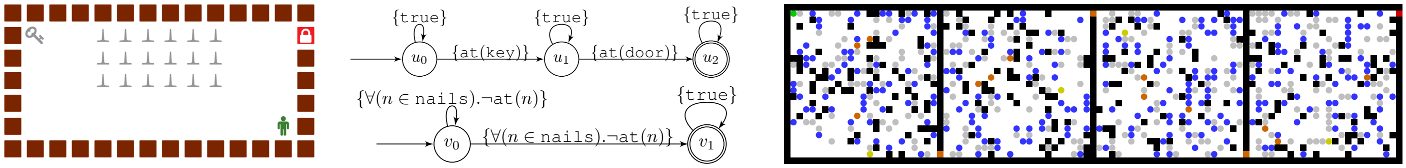
2 Preliminaries

Example environment: Consider a grid world in which the agent starts at some initial location. At various locations there are doors, keys, walls, and nails. The agent can move deterministically in the four cardinal directions, unless there is a wall or locked door in the way. The agent can only enter a location with a door when it has a key, after which point the door and key disappear (*i.e.* the door remains open). The agent automatically picks up a key whenever it visits a location with a key. The agent receives a reward of -1 for every action, unless it enters a location with nails (reward of -10) or reaches the red door with a key (reward of +1000, and the episode ends). Figure 1a, which we use as a running example below, depicts an instance of this domain, the "nail room", in which there is a single door and it is red.

In this environment, we may wish to advise the agent to avoid the nails, or to get the key before going to the door. In order to provide advice to an arbitrary RL agent, we must have a vocabulary from which the advice is constructed. For this purpose, we define a *signature* as a tuple $\Sigma = \langle \Omega, C, \mathrm{arity} \rangle$ where Ω is a set of predicate symbols, C is a set of constant symbols, and $\mathrm{arity} : \Omega \to \mathbb{N}$ assigns an arity to each predicate. For example, in the grid-world environment, we use a signature with only a single predicate called $\mathtt{at}$ (*i.e.* $\Omega = \{\mathtt{at}\}$ and $\mathrm{arity}(\mathtt{at}) = 1$), where $\mathtt{at}(c)$ states that the agent is at the same location as c. Each object in the domain will be represented with a single constant in C (*i.e.* $\mathrm{key1}, \mathrm{door1}, \ldots$). Intuitively, we use this signature to reference different elements about states in the environment when providing advice.

We assume that the set C is finite, and define $GA(\Sigma) := \{P(c_1, \cdots, c_{\mathrm{arity}(P)}) \mid P \in \Omega, c_i \in C\}$. That is, $GA(\Sigma)$ is the set of all *ground atoms* of the first order language with signature Σ. A *ground literal* is either a ground atom or the negation of a ground atom, so $\mathsf{lit}(\Sigma) := GA(\Sigma) \cup \{\neg p : p \in GA(\Sigma)\}$ is the set of ground literals. A *truth assignment* can be given by a set $\tau \subseteq \mathsf{lit}(\Sigma)$ such that for every $a \in GA(\Sigma)$, exactly one of a and $\neg a$ is in τ. Let $T(\Sigma)$ be the set of all truth assignments.

A *Markov Decision Process (MDP)* with a specified initial state is a tuple $\mathcal{M} = \langle S, s_0, A, p, \gamma, \Sigma, L \rangle$ where S is a finite set of *states*, $s_0 \in S$ is the initial state, $A(s)$ is a finite set of *actions* applicable in state $s \in S$, p is a function that specifies *transition probabilities* where $p(s', r|s, a)$ is the probability of transitioning to s' and receiving reward $r \in \mathbb{R}$ if action $a \in A(s)$ is taken in state s, $\gamma \in (0, 1]$ is the *discount factor*, $\Sigma = \langle \Omega, C, \mathrm{arity} \rangle$ is a signature, and $L : S \to T(\Sigma)$ labels each state with a truth assignment. For example, in our grid-world, the labeling function $L(s)$ makes $\mathtt{at}(c)$ true if and only if the location of the agent is equal to the location of c in state s. Note that as $GA(\Sigma)$ is finite, we could equivalently consider a state label to be a vector of *binary features*, with the ith entry being 1 if the ith ground atom holds in that state, and 0 otherwise. Below, we assume that the agent does not know the transition probability function p (as usual in RL).



(a) The nail room, an example

(b) Advice example

(c) An example randomly generated grid world map

Figure 1: (a) The agent receives reward for going to the locked door after having visited the key. It is penalized for stepping on nails. (b) NFAs corresponding to the LTL formula $\Diamond(\texttt{at(key)} \wedge \bigcirc\Diamond\texttt{at(door)}) \wedge \Box\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$. (c) Random map that contains an agent (●), walls (■), nails (●), holes (●), keys (●), cookies (●), doors (■), and a final door (■).

3 Providing and Utilizing Advice While Learning

In this section, we describe our LTL advice language and how corresponding automata can be used to monitor satisfaction of such advice. We then describe our method for using the automata to guide exploration in a variant of R-MAX.

Linear Temporal Logic: A Language for Providing Advice

Providing advice to an agent requires a language for communicating that advice. Given that RL agents are typically engaged in an extended interaction with the environment, this language must allow us to suggest how the agent should behave over time (*e.g. "Get the key and then go to the locked door."*). To this end, we use Linear Temporal Logic (LTL), a modal temporal logic originally proposed for the verification of reactive systems [8], that has subsequently been used to represent temporally extended goals and preferences in planning [4]. Here, we use it as the basis for expressing advice.

Suppose that we have an MDP $\mathcal{M} = \langle S, s_0, A, p, \gamma, \Sigma, L\rangle$ for which we wish to provide advice. The language of LTL contains formulae consisting of propositional symbols, which we take to be the ground atoms of $GA(\Sigma)$ (*e.g.* $\texttt{at(key2)}$), and all formulae that can be constructed from other formulae using the standard set of connectives from propositional logic — namely *and* ($\wedge$), *or* ($\vee$), and *negation* ($\neg$) — and the temporal operators *next* ($\bigcirc$) and *until* (U). From these temporal operators, we can also derive other useful operators such as *always* ($\Box$) and *eventually* ($\Diamond$). We will also make use of *universal* ($\forall$) and *existential* ($\exists$) quantifiers in abbreviating conjunctions and disjunctions. If $T = \{t_1, \ldots, t_k\} \subseteq C$ is a set of constant symbols, then $\forall(x \in T).\varphi(x) := \varphi(t_1) \wedge \cdots \wedge \varphi(t_k)$ and $\exists(x \in T).\varphi(x) := \varphi(t_1) \vee \cdots \vee \varphi(t_k)$.

To provide some intuition about LTL, we consider some possible example advice formulae for the problem in Figure 1a which use the unary operators $\Box$, $\Diamond$, and $\bigcirc$. The formula $\Box\neg\texttt{at}(\text{nail1})$ literally means "always, it is not the case that the agent is at the location of nail1"; used as advice it can be taken to say that "at all times the agent *should* not be at nail1." The formula $\Diamond(\texttt{at}(\text{key}) \wedge \bigcirc(\Diamond\texttt{at}(\text{door})))$ can be understood as "the agent should eventually get to a state where it is at the key and then eventually get to a state where it is at the door." If $\texttt{nails} \subset C$ is the set of objects that are nails, we can use $\Box\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$ to advise that at all times, the agent should not be at a location where there is a nail.

The truth value of an LTL formula φ is determined relative to a sequence $\sigma = \langle s_0, \ldots, s_n\rangle$ of states from $\mathcal{M}$ (*i.e.* the states visited in an episode). The truth of a ground atom at time t is determined by the label of s_t, and the truth values of more complicated formulae are built up according to the formal semantics of LTL (see De Giacomo *et al.* [3] for more detail).

From LTL to Finite State Automata and Tracking Advice Progress

Any LTL formula φ can be converted into a *Nondeterministic Finite State Automaton (NFA)* such that a finite sequence of states σ will be accepted by the NFA if and only if σ satisfies φ [1, 3]. We can represent the NFA as a directed graph with edges labelled by formulae from $\mathcal{L}_\Sigma$, the subset of LTL formulae that does not include temporal operators. Each edge represents a set of NFA transitions, one for each truth assignment satisfying the edge label. Because the NFA is non-deterministic, it may be in multiple states at once. Intuitively, the state(s) that an NFA is in after visiting a sequence of MDP states represent the progress that has been made towards satisfying φ. As such, we use an NFA state set to identify the relevant portion of a given advice formula in any state.

To translate from LTL to automata we use the system developed by Baier and McIlraith [1], which, for computational efficiency, constructs a set $\mathcal{N}$ of small NFAs rather than one potentially very large NFA. $\mathcal{N}$ is considered to accept σ if every NFA in $\mathcal{N}$ accepts σ. For example, Figure 1b shows the two NFAs generated from $\Diamond(\texttt{at(key)} \wedge \bigcirc\Diamond\texttt{at(door)}) \wedge \Box\forall(n \in \texttt{nails}).\neg\texttt{at}(n)$. This advice states that the agent should get the key and then go to the door, and also avoid nails. We now demonstrate how we track the NFA state set for the top NFA. At the beginning of an episode, the agent is in the initial state of the MDP and the NFA is in state u_0. Thus, the NFA state set is initialized to $\{u_0\}$. This NFA state set will remain constant until the agent reaches an MDP state s' for which $\texttt{at(key)} \in L(s')$ (*i.e.*, the agent gets the key). Since the transition to u_1 is now possible in the NFA, but it is also possible to remain in u_0 because $\texttt{true}$ holds in s', the NFA state set is updated to $\{u_0, u_1\}$. The state set will then remain constant until the agent reaches the door.

We note that the above procedure may lead to an empty NFA state set on some NFAs and state sequences. For example, notice that in the bottom NFA in Figure 1b, there is no transition to follow when the agent enters a state with a nail. This occurs because once the agent is at a nail, the episode can never satisfy the advice formula regardless of how the rest of the episode proceeds. We call such situations *NFA dead-ends*. Since the advice may still be useful even if it has been violated, we handle NFA dead-ends as follows: if an NFA state set becomes empty, we revert it to the previous set. The NFA in Figure 1b will therefore continue to suggest that the agent avoid nails even if the agent already failed to do so.

Background Knowledge Functions

Advice like "get the key" would not be very useful if the agent had no notion of what behaviour might lead to the key. To give advice, we must presuppose that the advised agent has some capacity to understand and apply the advice. To this end, in this paper we assume that the agent has a *background knowledge function* $h_B : S \times A \times \mathsf{lit}(\Sigma) \to \mathbb{N}$, where $h_B(s, a, \ell)$ is an estimate of the number of primitive actions needed to reach the first state s' where the literal ℓ is true (*i.e.*, where $\ell \in L(s')$) if we execute a in s (note that a itself is not counted). Intuitively, h_B represents the agent's prior knowledge — which may not be perfectly accurate — about how to make ground atomic formulae either true or false.

Since h_B only provides estimates with respect to individual ground literals, we believe that for many applications it should be relatively easy to manually define (or learn, *e.g.* [7]) a reasonable h_B. For example, in the grid world environment, we define the background knowledge function as $h_B(s, a, \texttt{at}(c)) = |\texttt{pos}(\mathit{agent}, s).x - \texttt{pos}(c, s).x| + |\texttt{pos}(\mathit{agent}, s).y - \texttt{pos}(c, s).y| + \Delta$ where $\texttt{pos}(c, s)$ provides the coordinates of c in state s and Δ is equal to -1 if the action a points toward c from the agent's position and 1 if it does not. Hence, if the agent is three locations to the left of the key, $h_B(s, \text{right}, \texttt{at}(\text{key}))$ will return 2 (right is the action to move right), even if there is a wall in the way. Furthermore, we defined $h_B(s, a, \neg\texttt{at}(c))$ to be equal to 1 if $h_B(s, a, \texttt{at}(c)) = 0$ and 0 otherwise.

Given any background knowledge function h_B, we can construct a function $h : S \times A \times \mathcal{L}_\Sigma \to \mathbb{N}$ that extends h_B to provide an estimate of the number of primitive actions needed to satisfy an arbitrary formula $\varphi \in \mathcal{L}_\Sigma$. We recursively define $h(s, a, \varphi)$ as follows: $h(s, a, \ell) = h_B(s, a, \ell)$ for $\ell \in \mathsf{lit}(\Sigma)$, $h(s, a, \psi \wedge \chi) = \max\{h(s, a, \psi), h(s, a, \chi)\}$, and $h(s, a, \psi \vee \chi) = \min\{h(s, a, \psi), h(s, a, \chi)\}$. In the next subsection, we describe how to drive the agent's exploration using h.

Advice-Based Action Selection in R-MAX

Model-based Reinforcement Learning solves MDPs by learning the transition probabilities and rewards. The R-MAX family of algorithms [2] are model-based methods that explore the environment by assuming that unknown transitions give maximal reward R_{max}. In practice, if R_{max} is big enough (and $\gamma < 1$), the agent ends up planning towards reaching the closest unknown transition. Instead, we propose to use the advice to plan towards *promising* unknown transitions.

Suppose the agent is at state s in the MDP, and we have kept track of the state sets that each NFA might be in. Intuitively, following the advice in s involves having the agent take actions that will move the agent through the edges of each NFA towards its accepting states. To this end, we begin by identifying *useful* NFA edges which may actually lead to progress towards the accepting states. For some NFA state u in an NFA state set, a transition from u to u' is useful if u is not an accepting state, and there exists a path in the NFA from u' to an accepting state that does not pass through u.

We now construct the *advice guidance* formula $\hat{\varphi}$ and the *advice warning* formula $\hat{\varphi}_w$. The advice guidance formula is given by the disjunction of all the formulae labelling useful edges in the NFAs. This means that $\hat{\varphi}$ will be satisfied by a transition in the MDP that achieves one of the formulae needed to transition over a useful edge. We define $\hat{h}(s, a) := h(s, a, \hat{\varphi})$ which can be used to rank actions based on an estimate of how close they are to making progress in satisfying the advice guidance formula $\hat{\varphi}$. While we omit the details, the advice warning formula $\hat{\varphi}_w$ is constructed similarly, but in such a way that it is satisfied by a transition in the MDP if and only if at least one transition is made in all of the NFAs. We then define the set $W(s) = \{a \in A(s) : h(s, a, \hat{\varphi}_w) \neq 0\}$. The idea is that W contains those actions which the evaluation function predicts will lead to an NFA dead-end.

We implemented a simple variant of an R-MAX algorithm that can take advantage of the advice. Instead of planning towards *the closest* unknown transition, we plan towards the closest unknown transition that is not in W and has minimum $\hat{h}$-value. If every unknown transition is in W, then we just go to the closest unknown transition with minimum $\hat{h}$-value.

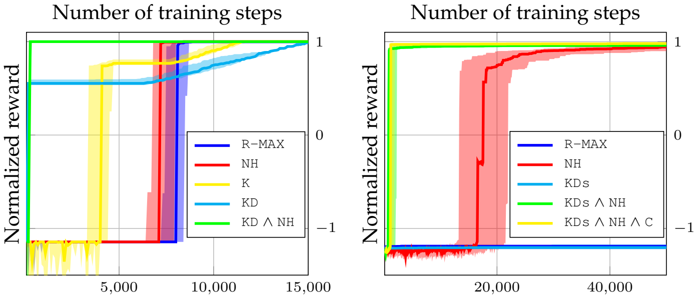
Convergence: As our approach plans towards unknown transitions (regardless of the advice), eventually all reachable transitions become known. By that point, the agent converges to a policy using its estimation of the transition probabilities. This policy is optimal with respect to $\mathcal{M}$ if the transitions are deterministic and near optimal otherwise [2].

4 Evaluation and Discussion

We tested our approach using various pieces of advice on grid world problems of the sort defined in Section 2, using the signature and background knowledge functions described in Sections 2 and 3. In addition to the domain elements of walls, keys, and nails described earlier, some problems also had *cookies* and *holes*. When the agent reaches a location with a cookie, it gets a reward of +10 and the cookie disappears. For reaching a hole, the reward is -1000 and the episode

Abbreviation	Advice formula (and informal meaning)
R-MAX	*No advice* Standard R-MAX.
C	$\forall(c \in \texttt{cookies}).\Diamond\texttt{at}(c)$ Get all the cookies.
NH	$\Box(\forall(x \in \texttt{nails} \cup \texttt{holes}).\neg\texttt{at}(x))$ Avoid nails and holes.
K	$\Diamond\texttt{at(key)}$ Get the key.
KD	$\Diamond(\texttt{at(key)} \wedge \bigcirc\Diamond(\texttt{at(door)}))$ Get the key and then go to the door.
KDs	$\forall(k \in \texttt{keys}).\Diamond(\texttt{at}(k) \wedge \bigcirc\Diamond(\exists(d \in \texttt{doors}).\texttt{at}(d)))$ For every key in the map, get it and then go to a door.

(a) Formula abbreviations for legends



(b) Nail room (25 × 50 version)

(c) Random grid world maps

Figure 2: In (b) and (c), reward is normalized so that 1 represents the best policy found on each map. The shaded areas represent the first and third quartiles. The formulae in the legends are explained by (a). Graphs are best viewed in colour.

ends. As a baseline, we reported the performance of standard R-MAX, which does not use any form of advice. To measure performance, we evaluated the agent's policy every 100 training steps. At training time, the agent explored the environment to learn a good model. At test time, we evaluated the best policy that the agent could compute using its current model (ignoring the unknown transitions). We used a discount factor of 1 in all experiments.

Figure 2b shows the median performance (over 20 independent trials) in a 25 × 50 version of the motivating example in Figure 1a. R-MAX, without advice, has poor initial performance, but converges to an optimal policy more quickly than when using the more detailed advice of K or KD (see Figure 2a for what these formulae are). However, K and KD quickly find a policy that follows the advice and then slowly converges to the optimal policy. As such, there is a trade-off between quickly learning the model (exploring nearby areas) and moving towards promising states suggested by advice. We also had an experiment with misleading advice (on a different map), where the agent was advised to go to a location far off any optimal path. The results were comparable to those seen for Figure 2b and are excluded due to lack of space.

We also randomly generated 10 grid world problems (Figure 1c shows one of them), each consisting of a sequence of four 25 × 25 rooms with doors between consecutive rooms. The agent always starts in the upper-left corner of the leftmost room and the red (*i.e.* goal) door is on the rightmost wall. Each space within a room had a 1/9 probability of being each of a nail, hole, or wall. A key and three cookies were randomly placed in each room among the remaining empty spaces, such that each key, door, and cookie was reachable from the starting location. These grid world maps are challenging because there is sparse reward and it is fairly easy to die. Figure 2c shows the performance over the 10 maps using five different pieces of advice. We ran 5 trials per piece of advice on each map; the graph reports the median performance across both trials and maps. Without advice, the agent was never able to reach the last door in $50K$ training steps. Providing advice that the agent should get keys and go to doors (KDs) was also not enough, because the agent always fell in a hole before reaching its target. Stronger results were seen with safety-like advice to avoid holes and nails (NH), and the best performance was seen when this safety-like advice was combined with advice that guides agent progress (KDs ∧ NH and KDs ∧ NH ∧ C). It is worth mentioning that none of these pieces of advice solves the problem for the agent, as the agent must still figure out how to turn the given advice into primitive actions, and handle cases where the advice is inappropriate (*e.g.* when the advice guides the agent towards cookies that are in a different room).

Future work: We plan to extend our work by prioritizing some parts of advice over others (for example, getting the key may be more important than getting every cookie), combining advice with hard constraints, experimenting with non-deterministic MDPs, and learning the background knowledge function h_B given a signature Σ. Finally, while the investigation here was limited to discrete MDPs, we also believe that many of the basic principles can be extended to a hybrid setting where discrete advice specified in LTL is used to guide search in a continuous space.

References

[1] J. Baier and S. McIlraith. Planning with first-order temporally extended goals using heuristic search. In *AAAI*, 2006.

[2] R. Brafman and M. Tennenholtz. R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2002.

[3] G. De Giacomo, R. De Masellis, and M. Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *AAAI*, 2014.

[4] A. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 2009.

[5] S. Krening, B. Harrison, K. Feigh, C. Isbell, M. Riedl, and A. Thomaz. Learning from explanations using sentiment and advice in RL. *IEEE Transactions on Cognitive and Developmental Systems*, 2016.

[6] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 1996.

[7] B. Peng, J. MacGlashan, R. Loftin, M. Littman, D. Roberts, and M. Taylor. A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In *AAMAS*, 2016.

[8] A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.