How Should Intelligence Be Abstracted in AI Research ...
AAAI Technical Report FS-13-02

How to Abstract Intelligence? (If Verification Is in Order)

Shashank Pathak[1] and **Luca Pulina**[2] and **Giorgio Metta**[1] and **Armando Tacchella**[3]
[1]Istituto Italiano di Tecnologia (IIT) — Via Morego 30, Genova
[2]POLCOMING, Università degli Studi di Sassari — Viale Mancini 5, Sassari
[3]DIBRIS, Università degli Studi di Genova — Viale F. Causa 13, Genova

Abstract

In this paper, we focus on learning intelligent agents through model-free reinforcement learning. Rather than arguing that reinforcement learning is the right abstraction for attaining intelligent behavior, we consider the issue of finding useful abstractions to represent the agent and the environment when verification is in order. Indeed, verifying that the agent's behavior complies to some stated safety property — an "Asimovian" perspective — only adds to the challenge that abstracting intelligence represents per se. In the paper, we show an example application about verification of abstractions in model-free learning, and we argue about potential (more) useful abstractions in the same context.

Introduction

Reinforcement Learning is an adaptive learning paradigm where agent interacts with the environment, both through accomplishing a set of actions and through observing the current state of the environment — see (Sutton and Barto 1998). The agent follows some policy, i.e., a mapping from states to actions, and receives feedback signal in terms of rewards. Interesting policies, i.e., those showing robustness for practical applications, are generally stochastic, which means that the probability of *unsafe* actions can not be diminished beyond a point. Here, and throughout the rest of the paper, by "unsafe action" we mean an action which leads directly to an undesirable state, and by "unsafe state" a state which is itself undesirable. Though the rewarding strategy could be tuned to account for visiting unsafe states by assigning negative rewards, or learning itself could be modified using, e.g., risk-sensitive paradigms like in (Geibel and Wysotzki 2005), these strategies might turn out to be insufficient to bound the probability of unsafe actions, partly due to the asymptotic nature of learning, and partly due to their implicit effects on overall learning. Hence, reinforcement learning could benefit from verification of learned policies through formal techniques such as probabilistic model checking — see, e.g., (Kwiatkowska, Norman, and Parker 2007).

The choice of a verification technique has an impact on the abstraction used for learning. This is because the model adaptively built by reinforcement learning should be expressed in the logic language chosen to verify safety properties. A priori, we cannot assume that representations that will work for learning will also be good for verification, which brings us to the central question of our contribution:

What is the right abstraction to represent the agent and the environment, if agents are to be demonstrably safe?

We consider this question in the framework of *Markovian Decision Processes* (MDPs), which is widely considered as the underlying model of reinforcement learning. An MDP is defined a 4-tuple $\langle \mathcal{S}, \mathcal{A}, R, T \rangle$ where $\mathcal{S}$ is the set of all possible *states* sensed by the agent, and $\mathcal{A}$ is the set of all its possible *actions*. The function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes *utility* of a state-action pair while $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the *transition* function. Notice that, if learning is model-free, then we are assuming that R and T are not known to the agent. In addition to this, MDPs enjoy the *Markovian Property* which implies

$$\begin{aligned} P(X_{t+1} = s | X_t = s_t, X_{t-1} = s_{t-1}, \ldots X_0 = s_0) \\ = P(X_{t+1} = s | X_t = s_t) \end{aligned} \tag{1}$$

where $P(\phi)$ stands for probability of ϕ being true, $s_0, s_1, \ldots s_t \in \mathcal{S}$ are states of the system, and $X_0, X_1, \ldots X_t, X_{t+1}$ are random-variables, interpreted as the value of the random "state" variable X at the discrete time instants $0, 1, \ldots, t, t+1$. In other words, memory of the past history is not required to estimate the probability of the next state. A *policy* π is any mapping from state-space to action-space i.e. $\pi : \mathcal{S} \mapsto \mathcal{A}$.

Reinforcement learning solves a *Markovian Decision Problem*, i.e., an MDP along with *objective function* that assigns a *value* $v : \Pi \mapsto \mathbb{R}$ to each policy $\pi \in \Pi$. In principle, knowing the value function v one could simply compute the *optimal policy* $\pi^* \in \Pi$, i.e., the policy such that $v(\pi^*) \geq v(\pi)$ for all $\pi \in \Pi$. However, the value function is usually not known, and reinforcement learning algorithms must approximate it. One of widely used methods is to consider *quality-value* (Q-value) for all state-action pairs

$$Q^{\pi}(s, a) = E^{\pi}(R_t | s_t = s, a_t = a)$$

where $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \ldots$ is a γ-discounted sum of rewards, where $0 \leq \gamma \leq 1$. Under suitable conditions the update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta \tag{2}$$

$$\delta = (r_{t+1} + \gamma Q_{nxt} - Q(s_t, a_t)) \quad (3)$$

guarantees learning the optimal policy π^*. When $Q_{nxt} = \underset{a \in A}{\text{argmax}} Q(s_{t+1}, .)$ we have *Q-Learning* (Watkins and Dayan 1992), while $Q_{nxt} = Q(s_{t+1}, a^\pi) | a^\pi \leftarrow \pi(s_{t+1})$ yields *SARSA* (Rummery and Niranjan 1994). It should be noted that such algorithms do not require knowledge of R and T, i.e., they are model-free in the sense outlined before. Also, they do not require explicit storage of R_t during learning or representation of learned MDP afterwards.

Algorithms such as SARSA and Q-Learning have made possible on-line learning of systems where state-action space is abstracted in continuous manifolds. The techniques used to learn in such abstractions turn out to be challenging for verification. One may trade-off some accuracy introducing discretizations which make verification of learned policies a feasible and useful goal. This is exactly the point we make throughout the first part of the paper. However, as we discuss further on, discretization may under-represent portions of the state space, and over-represent others. These two effects can pave the way to precision and efficiency issues, respectively, and hamper the ability to learn intelligent agents in interesting domains. In these cases, adaptive continuous-to-discrete abstractions could be a potential solution. In the final part of the paper we present some hints in the direction of adaptive discretization that could improve on precision while, at the same time, preserving the possibility to demonstrate safety of agents in a formal setting.

Discrete-state abstraction

Given a discrete state-action space, the combination of the original MDP and a policy π computed by learning yields a *discrete-time Markov chain* (DTMC) describing all possible interactions of the agent with the environment. Following (Kwiatkowska, Norman, and Parker 2007), given a set of atomic propositions AP, a DTMC is defined as a tuple $(W, \overline{w}, \mathbf{P}, L)$ where W is a finite set of *states*, and $\overline{w} \in W$ is the *initial state*; $\mathbf{P} : W \times W \to [0, 1]$ is the *transition probability matrix*; and $L : W \to 2^{AP}$ is the *labeling function*. An element $\mathbf{P}(w, w')$ gives the probability of making a transition from state w to state w' with the requirement that $\sum_{w' \in W} \mathbf{P}(w, w') = 1$ for all states $w \in W$. A terminating (absorbing) state w is modeled by letting $\mathbf{P}(w, w) = 1$. The labeling function maps every state to a set of propositions from AP which are true in that state. Safety assurance is expressed in terms of Probabilistic Computational Tree Logic (PCTL) formulas, whose syntax and semantic are described formally in (Kwiatkowska, Norman, and Parker 2007), to which we refer the reader for further details.

Let us assume that states and transitions explored during LEARN are logged in a table $T : S \times A \to 2^{S \times \mathbb{N}}$, where S is the state-space and A is the action-space. For each state $s \in S$ and action $a \in A$, we have that $T(s, a) = \{(s_1; n_1), \ldots, (s_k; n_k)\}$ where $s_i \in S$ for all $1 \leq i \leq k$, and n_i is the number of times state s_i occurred as next state in a transition from state s on action a; each cell (s, a) of T stores only the pairs $(s_; n_i)$ such that $n_i > 0$, i.e., state s_i appeared at least once in a transition from an explored state s on action a. For all actions $a \in A$, if the state $s \in S$ was explored, but had no outgoing transitions, then $T(s, a) = \emptyset$; if a state was not explored, then $T(s, a) = \perp$.[1] Assuming transitions can be unsafe, i.e., the action itself is undesirable, we consider a table $F : S \times A \to \{\text{TRUE}, \text{FALSE}\}$ where, for each explored state $s \in S$ and action $a \in A$, we log whether the specific action a is found to be unsafe when started from state s.

Policy (safety) verification

We combine tables T and F with the Q-table returned by LEARN to obtain a DTMC $\mathcal{M} = (W, \overline{w}, \mathbf{P}, L)$. The set AP of atomic propositions has just one element bad which denotes unwanted states. The construction of $\mathcal{M}$ is the following:

1. The set of states W is initialized to explored states logged in T, i.e., all the states $s \in S$ such that for all actions $a \in A$ we have $T(s, a) \neq \perp$, plus three additional states $w_0, w_{ok}, w_{bad} \notin S$. The initial state $\overline{w}$ is w_0. For all $v, w \in W$, we initialize $\mathbf{P}(v, w) = 0$, and $L(w) = \{\}$.
2. Let $I \subseteq S$ be the set of *source states* in T, i.e., $s \in I$ exactly when $(s, n) \notin T(s', a)$ for all $s' \in S$, $n \in \mathbb{N}$ and $a \in A$. We make w_0 the unique initial state by requiring $\mathbf{P}(w_0, w) = \frac{1}{|I|}$ for all $w \in I$,. meaning that w_0 has non-deterministic transition to every source state.
3. Given two states $v, w \in W$ such that $v, w \notin \{w_0, w_{ok}, w_{bad}\}$, the probability of transition is defined as
$$\mathbf{P}(v, w) = \sum_{a \in A} \pi_Q(v, a) \cdot p_T(v, a, w) \quad (4)$$
where π_Q is the stochastic policy extracted from Q, and $p_T(v, \hat{a}, w)$ is the probability of observing w as a successor of v given that action $\hat{a}$ was taken. Assuming $T(v, \hat{a}) = \{(w_1; n_1), \ldots, w_k; n_k)\}$ we have
$$p_T(v, \hat{a}, w) = \begin{cases} \frac{n_w}{\sum_{i=1}^{k} n_i} & \text{if } (w; n_w) = (w_j; n_j) \\ & \text{for some} 1 \leq j \leq k \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

We define $K \subseteq S$ to be the set of *sink states* in T, i.e., $s \in K$ exactly when $T(s, a) = \emptyset$ for all $a \in A$. If the notion of unsafety is attached to transitions, then the construction of $\mathcal{M}$ is completed as follows:

4. The state w_{ok} is an abstraction of all sink states that can be reached by at least one safe transition, and $\mathbf{P}(w_{ok}, w_{ok}) = 1$. For each state v, let *safe*$(v) \subseteq K$ be the set of sink states that can be reached from v via a safe transition, i.e., all states $w \in K$ such that there exist $a \in A$ with $(w; n_w) \in T(v, a)$ and $F(v, a) =$ FALSE; for all states v such that *safe*$(v) \neq \emptyset$, we set

[1]Table T can handle (i) deterministic actions, i.e., a is such that $\forall s \in S. |T(s, a)| = 1$; (ii) non-deterministic actions, i.e.,a is such that for all $s \in S$, both $|T(s, a)| = k$ and $n_i = 1/k$ for all $1 \leq i \leq k$; and (iii) probabilistic actions, i.e., arbitrary values of n_i.

$\mathbf{P}(v, w_{ok}) = \sum_{w \in \textit{safe}(v)} \mathbf{P}(v, w)$ and then $\mathbf{P}(v, w) = 0$ for all $w \in \textit{safe}(v)$.

5. The state w_{bad} is an abstraction for all states – including non-sink ones – that can be reached by at least one unsafe transition. Therefore we set $\mathbf{P}(w_{bad}, w_{bad}) = 1$ and $L(w_{bad}) = \{bad\}$. For each state v, *unsafe*$(v) \subseteq K$ is the set of sink states that can be reached from v via an unsafe transition, i.e., all states $w \in K$ such that there exist $a \in A$ with $(w; n_w) \in T(v, a)$ and $F(v, a) = \text{TRUE}$; for all states v such that *unsafe*$(v) \neq \emptyset$, we set $\mathbf{P}(v, w_{bad}) = \sum_{w \in \textit{unsafe}(v)} \mathbf{P}(v, w)$ and then $\mathbf{P}(v, w) = 0$ for all $w \in \textit{unsafe}(v)$.

Here, we end up removing *orphan* states, i.e,, states $w \in W$ with $w \neq w_0$ such that $\mathbf{P}(v, w) = 0$ for all $v \in W$ with $v \neq w$. For all such states w, we also remove outgoing connections by setting $\mathbf{P}(w, v) = 0$ for every $v \in W$. The process is repeated until no orphan state and related transitions are left.

Given the construction above, while the concrete meaning of safety will depend on the way F is computed, from an abstract point of view we must ensure that the probability of reaching w_{bad} is low, i.e., check

$$\mathcal{M}, w_0 \models \mathcal{P}_{<\sigma}[\mathcal{F}\ bad] \tag{6}$$

where $\sigma \in [0, 1]$ is a safety assurance probability. Typical assertions that can be checked in this context include, e.g., the probability of a bad state is less than 1%, i.e., checking that $\mathcal{M}, w_0 \models \mathcal{P}_{<0.01}[\mathcal{F}\ bad]$ holds. A control policy such that property (6) cannot be verified for small values of σ is clearly undesirable, also in the cases where low-level protection mechanism prevent actual failures to occur.

Policy repair

We define *policy repair* as a verification-based procedure that modifies the Q-table to fix the corresponding policy.

Repair is driven by counterexamples that witness the violation of property (6) for some threshold σ. Since there are no infinite paths in $\mathcal{M}$, property (6) is violated only if σ is less than the probability of choosing a path $\tau = w_0, w_1, \ldots, w_k$ such that $w_k = w_{bad}$. The *probability mass* of a path τ is defined as

$$Prob(\tau) = \Pi_{j=1}^{k} \mathbf{P}(w_{j-1}, w_j) \tag{7}$$

and extended to a set of paths $\mathcal{C} = \{\tau_1, \ldots, \tau_n\} \subseteq Path_{w_0}$ as $Prob(\mathcal{C}) = \sum_{i=1}^{n} Prob(\tau_i)$. If, for all $1 \leq i \leq n$, we have $\tau_i = w_{i,0}, w_{i,1}, \ldots, w_{i,k_i}$ with $w_{i,0} = w_0$, $w_{i,k_i} = w_{bad}$ and $Prob(\mathcal{C}) > \sigma$, then $\mathcal{C}$ is a counterexample of property (6). Notice that a counterexample $\mathcal{C}$ is unique only if it contains all paths starting with w_0 and ending with w_{bad}, and for all $\mathcal{C}' \subset \mathcal{C}$ we have that $Prob(\mathcal{C}') < \sigma$. Given our assumptions, the counterexample $\mathcal{C} = \{\tau_1, \ldots, \tau_n\}$ is a DAG such that $|\mathcal{C}| \leq |Path_{w_0}|$. However, unless learning is highly ineffective, we expect that $|\mathcal{C}| \ll |Path_{w_0}|$, i.e., focusing on the counterexample is an effective way to limit the repair procedure to a handful of "critical" paths. However, focusing on the counterexample is just the first step in verification-based repair, and we also need to find an effective way to alter the policy, i.e., the Q-values, so that probability masses are shifted in the DTMC and property (6) is satisfied. Our implementation leverages two key ideas to accomplish this. The first one is that transitions between states which are "far" from w_{bad} should be discouraged less that those that are "near" w_{bad}. The second idea is that probabilities should "flow away" from paths ending with w_{bad} towards paths ending with w_{ok}. Altering transitions which are close to w_{bad} helps to keep changes to the policy as local as possible, and to avoid wasting useful learning results. Shifting probabilities towards paths ending with w_{ok} avoids moving probability mass from one unsafe path to another that would otherwise yield no global improvement.

Given a model $\mathcal{M}$ built out of tables Q, T, F as described before, a counterexample $\mathcal{C} = \{\tau_1, \ldots, \tau_n\}$ of the safety property (6) for a given threshold σ, and an iteration parameter $\theta \in \mathbb{N}$, repair is implemented by the following algorithm:

1. Make a copy Q' of the (current) Q-table Q.
2. Choose $\tau \in \mathcal{C}$ such that $Prob(\tau) \geq Prob(\tau')$ for all $\tau' \in \mathcal{C}$, i.e., τ is the path in $\mathcal{C}$ with the largest probability mass.
3. For each $w_i \in \tau$ $(0 \leq i < k)$, compute $b_\tau(w_i)$ and let $v = succ_\tau(w)$; find the action $a \in A$ such that $(v; n_v) \in T(w, a)$ and modify $Q'(w, a)$ as follows
$$Q'(w,a) = \begin{cases} Q'(w,a) \cdot z_\tau(w) & \text{if } Q'(w,a) \geq 0 \\ Q'(w,a) \cdot \frac{1}{z_\tau(w)} & \text{if } Q'(w,a) < 0 \end{cases}$$
4. Let $\mathcal{C} = \mathcal{C} \setminus \{\tau\}$; go to step (2), unless $\mathcal{C} = \emptyset$ or more than θ paths were considered.
5. Let $\mathcal{M}'$ be the DTMC obtained by T, F and Q' in the usual way; check if $\mathcal{M}'$ fulfills (6); if so, then stop, otherwise a new counterexample $\mathcal{C}'$ is available: let $\mathcal{C} = \mathcal{C}'$, $\mathcal{M} = \mathcal{M}'$, $Q = Q'$ and go to step (1).

Case study: reach-avoid with iCub

This case study is about the humanoid iCub (Metta et al. 2010) learning to reach for an object (*target*) placed on a table, without damaging other objects (*obstacles*) in the process. The iCub moves its right-arm along with the torso to reach for the object, and has 19 degrees-of-freedom (DoF) to do so. Due to this high redundancy, learning directly in joint-space is not feasible. Inspired by modeling of human-reaching (Berthier 1996), the actions are planned in a lower dimensional Cartesian space, where each action itself is a smooth minimum jerk trajectory. Though such trajectory itself could be learned during motor-babbling (as in infants), we assumed such control trajectory was available. Indeed, choosing a trajectory based on some sound minimum principles (Engelbrecht 2001), could be seen as an abstraction of high dimensional joint-space kinematics to a lower space of low-degree polynomial in joint-angle where, in case of unconstrained minimum-jerk trajectory, this polynomial has degree 5. To further reduce complexity of state-space, we assume higher level actions to be planned and executed in a Cartesian space with orientation, i.e., (x, y, z) and $(\hat{v}, \alpha)$ where x, y, z are Cartesian coordinates and $\hat{v}$ is unit vector

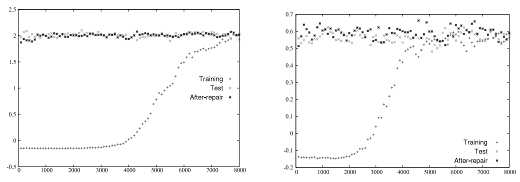

Figure 1: *left* shows advent of learning - in terms of averaged rewards - against real-time (in minutes) spent by iCub arm in the trials. Repair did not deteriorate the policy and in fact decreased its variance. *right* shows probability of unsafe states, where safety is significantly improved after repair.

in direction of rotation and α is magnitude of the rotation. Details of the Cartesian controller implemented in iCub can be found in (Pattacini et al. 2010). Hence each state is a vector $(x, y, z, o) \in S$, where the component (x, y, z) represents the Cartesian coordinates of iCub's hand in a system having origin in its torso, and o represents the orientation of the hand. Each action changes state vector in a point-to-point motion. This is similar to reaching movements in infants, where overall trajectory is composed of many sub-movement with similar looking velocity profiles — see studies by von Hofsten, e.g., (von Hofsten 1991). Using iCub Cartesian controller, we defined the *reach-avoid* task as learning to reaching for a ball kept on table while avoiding a cup that might interfere with it. Effectiveness of a policy learned with the SARSA method is measured in terms of averaged rewards for all successful reaches while safety was defined as probability of not hitting the cup. When execution of policy resulted in this safety value below a threshold, policy was sought to repair. To view robustness of repair, noise was introduced to initial-position of the arm, which resulted in different policies and its repair.[2].

As shown in Figure 1, verification-based repair does not deteriorate overall policy learned by iCub for reaching an object. On contrary, safety of policy is increased quite significantly after repair as shown in Table 1. This will be true as long as primary goal of reaching is not severely constrained by supplementary goal of remaining safe. Robustness of the repair can be gauzed by studying its effect considering uncertainty in the initial position of the hand. In our experiments, we modeled such uncertainty using additive Gaussian noise with zero mean and standard deviation σ. The results are reported in Table 1 and a pictorial view is given in Figure 2. In both cases, we can observe that as σ is increased, the learned policy gets inferior. However, repair actually helps in moderating such policies. In fact, the same safety threshold can be assured in all such cases, and in some of them also with added benefit of slightly better — more effective — policies. This is similar to control-theoretic intuitions of safety being of assistance in certain adaptive control system (Gillula and Tomlin 2012).

[2]Details of the experiment can be found at http://www.mind-lab.it/~shashank/IROS2013

Noise	Initial		Learning		Repair	
	Eff.	Safety	Eff.	Safety	Eff.	Safety
Absent	0.5770	0.0043	2.0127	0.0081	2.0025	$6.10 \cdot 10^{-05}$
Low	0.6068	0.0040	1.0980	0.0675	1.0990	$3.93 \cdot 10^{-06}$
Medium	0.2359	0.0048	0.9996	0.0135	0.9910	$5.07 \cdot 10^{-12}$
High	0.2570	0.0107	0.5626	0.0366	0.5914	$4.053 \cdot 10^{-06}$

Table 1: Effectiveness and safety levels in learning to reach objects. "Noise" is how much the initial position of the hand varies because of noise. "Initial", "Learning" and "Repair" denote that the measurements is done before learning, after learning and after repair, respectively. Effectiveness ("Eff.") is the average reward obtained, and "Safety" is the value of the threshold σ.

Continuous-state abstraction: a proposal

Reinforcement learning algorithms are well understood in discrete state domain, and we have shown that their result can be verified and improved also in a practical case. However, as mentioned in the introduction, continuous state space abstractions proved often to be more effective for applications. Reinforcement learning for continuous domains comes in a variety of flavors, such as functional approximators (Baird 1995), adaptive clustering, neural-networks, fuzzy sets and many others. Some of these methods provide nice theoretical guarantees on convergence, while some can even be coupled with other learning algorithms — like, e.g., policy gradient methods using functional approximators. Indeed, for most of them, it is not straightforward to find a way to verify the learned policy as we do with discrete abstractions. In the following, we discuss a possible approach that should retain some of the advantages of continuous-state abstractions, while enabling formal verification and repair at the same time.

Borrowing the notations of (Li, Walsh, and Littman 2006), let the tuple $M = \langle S, A, P, R, \gamma \rangle$ denote the ground MDP while $\hat{M} = \langle \hat{S}, \hat{A}, \hat{P}, \hat{R}, \gamma \rangle$ denote its *abstraction*. Furthermore, let $\phi(s) \colon S \to \hat{S}$ denote an *abstraction function*, i.e., $\forall s \in S, \phi(s) \in \hat{S}$. We can say that $\phi_2(s)$ is *coarser* than $\phi_1(s)$ iff $\forall s_1, s_2 \in S, \phi_1(s_1) = \phi_1(s_2) \implies \phi_2(s_1) = \phi_2(s_2)$. Since the primary goal of abstraction in reinforcement learning is to reduce the dimensionality, we are interested in *coarser* abstractions of ground MDP. In theory, one would like to obtain an abstraction which is as precise as possible. In (Givan, Dean, and Greig 2003), authors introduce the notion of equivalence and obtain a *bisimilar*, but minimized, model of the original MDP for learning purposes. However, this result cannot be used for abstracting a continuous state-space to a discrete one, and the concept of bisimulation is itself very strict. One may deviate a bit away from model equivalence, and have abstractions that preserve model similarity. Furthermore, an abstraction could preserve state-action values Q^π, or could preserve optimal state-action values Q^* ,or just the optimal action values a^*. In (Li, Walsh, and Littman 2006) it is shown that the order for coarser abstraction is $\phi_{bisim} < \phi_{Q^\pi} < \phi_{Q^*} < \phi_{Q^{a*}}$ and that the optimal policy with respect to the ground MDP is preserved in all these abstractions except the last one.

In particular, ϕ_{Q^*} is Q^*-irrelevance abstraction such that $\forall a \in A, \phi_{Q^*}(s_1) = \phi_{Q^*}(s_2) \implies Q^*(s_1, a) = Q^*(s_2, a)$.

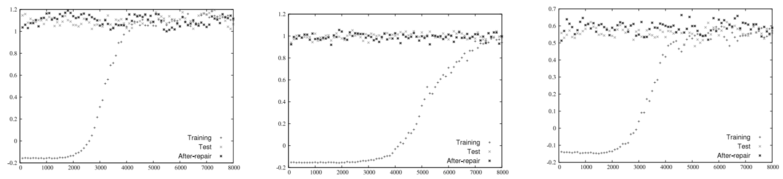

Figure 2: Averaged rewards are plotted against episodes. Noise is defined as uncertainty in initial position of iCub hand. Probability distribution is $\mathcal{N}(s_0, \sigma)$ with s_0 as initial position and σ as standard deviation. *left* is case of low noise $\sigma = .005$, *middle* is with medium-noise of $\sigma = .01$ while *right* is with high-noise of $\sigma = .02$.

```
1: Given initial U-tree $U_0$ with single leaf $L$, no decision node
   $Q$, split-test T-split,leaky-bucket insert scheme Insert
2: Obtain $(\hat{s}, \Delta Q)$ from agent's action
3: State-chunk $S_i \leftarrow$ Fetch($U_0, \hat{s}$)
4: Insert(($\hat{s}, \Delta Q$),$S_i$)
5: Data $D_i \leftarrow$ data of this state-chunk
6: for $d = 1 \rightarrow |\hat{s}|$ do
7:     Sort the data of leaf wrt $d$ $D_{sorted} \leftarrow$ Sort($D_i$,d)
8:     for $p = 1 \rightarrow |D_i|$ do
9:         $(P_{left}, P_{right}) \leftarrow$ Partition($D_{sorted}$,p)
10:        Apply split-test T-split($P_{left}, P_{right}$)
11:        Store current best metric $M_{split}$ , d and p
12:    end for
13: end for
14: if Should-split $\leftarrow M_{split}$ then
15:     $Q_{new}, L_{left}, L_{right} \leftarrow L_{old}$
16: end if
```

Figure 3: Pseudo-code for U-Tree

Since both verification and repair depend on the learned policy, this abstraction turns out to be potentially useful in our context. Such an abstraction may not preserve the rewards or overall outgoing probabilities, i.e., for abstract-space $\hat{S}$, we may have, $\forall a \in A, \phi_{Q^*}(s_1) = \phi_{Q^*}(s_2) \not\Rightarrow R^a_{s_1} = R^a_{s_2}, \not\Rightarrow \sum_{s' \in \phi^{-1}(\hat{S})} P^a_{s_1 s'} = \sum_{s' \in \phi^{-1}(\hat{S})} P^a_{s_2 s'}$. This relaxation allows us to be dispensed of the knowledge of complete MDP for abstracting, which is quite crucial for practical applications. Some algorithms that use Q^* abstraction for learning are G-algorithm and U-tree based reinforcement learning (Uther and Veloso 1998).

In particular, the U-tree algorithm for reinforcement learning, hereafter denoted by *U-tree*, aims at incremental clustering of a multi-dimensional continuous state-space. In Figure 3 we present its pseudocode, but more details can be found in (Uther and Veloso 1998). It is a ϕ_{Q^*} abstraction and hence preserves the optimal policy under abstraction. Also it does not require knowledge of complete MDP whereas it seeks to find equivalent rewards as well as Q-values. The algorithm requires statistical testing on a set of observed Q-values and therefore suffers from non-normal distribution of samples especially when number of samples is not large. The method is not exact in this sense.

We implemented *U-tree* on a well known test benchmark of cart-pole problem (Barto, Sutton, and Anderson 1983), where the task is to learn to balance an inverted pendulum mounted on top of cart, through force on lateral direction of the cart. The state-space is $\{ x, \dot{x}, \theta, \dot{\theta} \}$ denoting lateral position and lateral velocity of the cart, and angular position and angular velocity of the pole, respectively. Implementation details are given on the companion website.[3] Figure 4 shows the decision trees obtained with *U-tree* using different splitting tests. The decision hyper-planes enforced by *U-tree* are aligned to one the state-dimensions, hence local orthogonality of causal effects of each dimension is implicitly assumed. Partition of state-space depends on local variation of the *value* of states. As mentioned before, a discrete-state representation provides succinct knowledge of the policy learned, *U-tree* clubs an infinite set of continuous states trading with approximation error in values of these states. In case of cart-pole, states closer to equilibrium point would all have similar values and hence would be partitioned sparsely. This is shown in Figure 5.

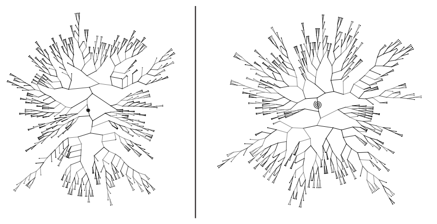

Figure 4: Decision tree (in radial layout) using K-S test (left) and MSE (right) as splitting criteria.The root node is in the center marked with a spiral

When learning with U-tree is over, the policy π is defined over a state space that has been partitioned adaptively during the learning process — we assume that the action space is discrete from the beginning. In particular, the elements of the state space mapped to actions by π are the leaves of the tree that, starting from a single node representing all possible states, is grown by U-tree to partition the learning space in a relevance-based fashion. Now, if we assume that the final policy π needs to be verified and repaired, we can observe that the methods proposed for discrete abstractions can still work as long we consider the leaves of the tree built by U-tree as states of the DTMC to be verified. We outline this process in algorithm 6.

In this case, actions are discrete, *U-tree* provides a deci-

[3]The companion site is available at www.mind-lab.it/~shashank/AAAI_FSS13.

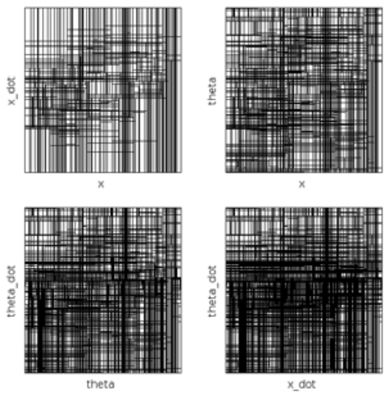

Figure 5: The plot shows partitions of a pair of state-variables, normalized in range [-1,1].

```
1: Given a learned U-tree $U_{learned}$, safety property $\mathcal{P}$, allowed
   probability-bound $P_{bound} \in \mathbb{R}$
2: Construct a DTMC, $D_0$ from leaves of U
3: $D_{en} \leftarrow \textsf{Encode}(D_0)$
4: $\textsf{Verify}(D_{en}, \mathcal{P}, P_{bound})$
5: if Counter-example $C \neq \{\phi\}$ then
6:     $D_{rep} \leftarrow \textsf{Repair}(D_{en}, C, P_{bound})$
7:     $U_{rep} \leftarrow \textsf{Update}(U_0, D_{rep})$
8: end if
```

Figure 6: Pseudo-code for automated repair of U-Tree

sion tree $D_i$for each action i with leaves as Q-values for that state-space. In general, the decision hyperplane would be different for different actions, hence while creating DTMC we consider two continuous states belonging to same discretized state iff every action a_j lies in same leaf of D_j. Obviously, this further increases the space complexity of DTMC, especially if the number of actions is large. Also, the set of next states $\mathcal{S}_{i,j}$ when action a_j is executed in state belonging to D_i, is calculated through recorded observations of state transitions. Thus from the perspective of DTMC encoding, actions are non-deterministic. Approximation in this approach, such as next state being only a part of D_j for some j, does not affect the nature of the policy to be verified. Repair of such DTMC poses different challenge though. Since the actions of DTMC are not actual actions in the policy, it is not clear how the policy should be repaired in light of a counter-example. For example, performing action a in state D_i may yield a safe state D_j and an unsafe state D_k. Transition probability $T(i,k)$ can only be decreased through decreasing value of state D_i, which would in turn decrease transition probability $T(i,j)$. This anomaly arises due to the fact that bifurcation of state-space was safety-agnostic.

Conclusions

Safety poses a credible challenge for reinforcement learning. Given a discrete state-action domain, model checking techniques can be harnessed out-of-the-box to provide quantitative measures of inherent risk in the policy, and to repair it to ensure safety. Continuous state-space (or even action-space) domains result in demonstrable learning tasks on complex systems, but pose the issue of state-abstraction for verification. While there exists a large pool of such abstractions that have been used in practical reinforcement learning, we must identify one which is suitable for verification as well. Pending some further improvements related to repair, U-tree based learning is a reasonable candidate in this direction.

References

Baird, L. C. 1995. Residual algorithms: Reinforcement learning with function approximation. In *ICML*, 30–37.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *Systems, Man and Cybernetics, IEEE Transactions on* (5):834–846.

Berthier, N. E. 1996. Learning to reach: A mathematical model. *Developmental psychology* 32(5):811.

Engelbrecht, S. E. 2001. Minimum principles in motor control. *Journal of Mathematical Psychology* 45(3):497–542.

Geibel, P., and Wysotzki, F. 2005. Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. *J. Artif. Intell. Res. (JAIR)* 24:81–108.

Gillula, J., and Tomlin, C. 2012. Guaranteed Safe Online Learning via Reachability: tracking a ground target using a quadrotor. In *ICRA*, 2723–2730.

Givan, R.; Dean, T.; and Greig, M. 2003. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence* 147(1):163–223.

Kwiatkowska, M.; Norman, G.; and Parker, D. 2007. Stochastic model checking. *Formal methods for performance evaluation* 220–270.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for mdps. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 531–539.

Metta, G.; Natale, L.; Nori, F.; Sandini, G.; Vernon, D.; Fadiga, L.; von Hofsten, C.; Rosander, K.; Lopes, M.; Santos-Victor, J.; et al. 2010. The iCub Humanoid Robot: An Open-Systems Platform for Research in Cognitive Development. *Neural networks: the official journal of the International Neural Network Society*.

Pattacini, U.; Nori, F.; Natale, L.; Metta, G.; and Sandini, G. 2010. An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1668–1674.

Pazis, J., and Parr, R. 2013. Pac optimal exploration in continuous space markov decision processes.

Rummery, G., and Niranjan, M. 1994. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning – An Introduction*. MIT Press.

Uther, W. R., and Veloso, M. M. 1998. Tree based discretization for continuous state space reinforcement learning. In *Proceedings of the National Conference on Artificial Intelligence*, 769–775. JOHN WILEY & SONS LTD.

von Hofsten, C. 1991. Structuring of early reaching movements: a longitudinal study. *Journal of motor behavior* 23(4):280–292.

Watkins, C., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3):279–292.