2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

Ensuring safety of policies learned by reinforcement: Reaching objects in the presence of obstacles with the iCub

Shashank Pathak, Luca Pulina, Giorgio Metta, Armando Tacchella

***Abstract*— Given a stochastic policy learned by reinforcement, we wish to ensure that it can be deployed on a robot with demonstrably low probability of unsafe behavior. Our case study is about learning to reach target objects positioned close to obstacles, and ensuring a reasonably low collision probability. Learning is carried out in a simulator to avoid physical damage in the trial-and-error phase. Once a policy is learned, we analyze it with probabilistic model checking tools to identify and correct potential unsafe behaviors. The whole process is automated and, in principle, it can be integrated step-by-step with routine task-learning. As our results demonstrate, automated fixing of policies is both feasible and highly effective in bounding the probability of unsafe behaviors.**

I. Introduction

Reinforcement Learning (RL), see, e.g., [1], is one of the most widely adopted paradigms to obtain intelligent behavior from interactive robots. Several RL applications have been proposed in the literature wherein control policies are synthesized by RL for a variety of tasks However, as mentioned in [2], the asymptotic nature of guarantees about performance of RL makes it difficult to bound the probability of damaging the controlled robot and/or the environment. This limits its applicability in all those cases where robots are deployed, e.g., in close contacts with humans, or in other safety-critical contexts.

Our goal is to guarantee that, given a control policy synthesized by RL, such policy will keep the robot and the environment safe with high probability. Safety could be ensured through control-theoretic methods, e.g, Lyapunov functions [3], or risk-sensitive learning as in [4], but these methods require knowledge of either domains or algorithms, while we are interested in Model Checking techniques [5], [6] to verify safety properties automatically. Since we wish to consider stochastic policies, safety cannot be assessed in terms of Boolean logic, but probabilistic reasoning is to be used instead. This can be achieved through Probabilistic Model Checking techniques – see, e.g., [7] – wherein system and properties have probability values associated to them. We consider also the problem of automating repair, i.e., once the policy is found unsatisfactory, it should be fixed without manual inspections.

Shashank Pathak and Giorgio Metta are with Robotics, Brain and Cognitive Sciences (RBCS), Istituto Italiano di Tecnologia (IIT), Via Morego, 30 – 16163 Genova – Italy Shashank.Pathak@iit.it - Giorgio.Metta@iit.it. Luca Pulina is with POLCOMING, Università degli Studi di Sassari, Viale Mancini 5 – 07100 Sassari – Italy lpulina@uniss.it. Armando Tacchella is with Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS). Università degli Studi di Genova, Via Opera Pia, 13 – 16145 Genova – Italy Armando.Tacchella@unige.it.

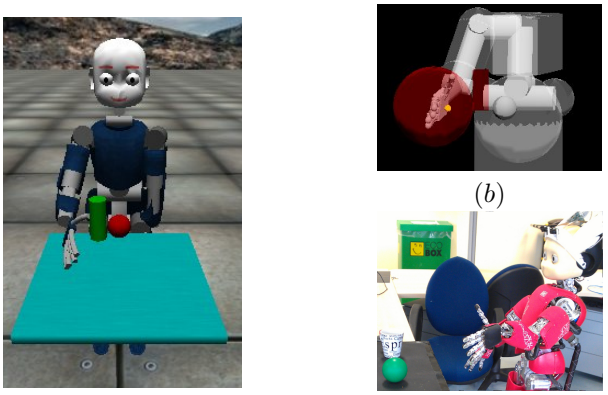

Fig. 1: Experimental setup with the iCub. Picture (a) shows the scene as seen in the iCub simulator during learning: the red ball is the target, and the green cylinder is the obstacle; reaching is performed with the right hand. Picture (b) is a snapshot from MoBeE tool, that we utilize to check collisions between iCub and objects which is not supported by the iCub simulator. Picture (c) is the red iCub during the test phase: the green ball is to be reached with the right hand, and the paper cup is the obstacle.

We contribute a new methodology based on probabilistic model checking to ensure safety of stochastic policies obtained with RL. In particular, we provide an automated procedure to verify and fix learned policies until they comply to stated safety requirements. We showcase our method with the iCub humanoid [8] considering a task where the robot is to reach a target object on a table in the presence of nearby objects (obstacles) which should not be affected by the robot's movement. The setup is presented in Figure 1: Figure 1 (a) is a snapshot of the iCub simulator [9] used in the learning phase in order to avoid damage to the real iCub. Since the simulator does not support detection of collisions with external objects, we added MoBeE [10] – shown in Figure 1 (b) – in order to detect collisions and assign proper rewards and/or control the duration of learning episodes. Finally, we tested repaired policies on the "red iCub" depicted in Figure 1 (c). Notice that in both simulated and physical setups, we did not consider any other source of difficulty other than the proximity of the target objects and the obstacle, which itself makes the learning problem quite complex in terms of safety guarantees – further details on the case study are presented in Section II.

After a short introduction to probabilistic model checking in Section III, Section IV describes our main contribution. The interaction between iCub and the environment is modeled as a deterministic time Markov chain computed at the end of the learning process. The likelihood of ill behaviors is

encoded as reachability properties on the Markov chain. Repair leverages the results of model checking to fix agents in order to comply to stated safety requirements. Experimental evaluation of verification and repair with the state-of-the-art probabilistic model checker COMICS [11] is shown in Section V. Repair is evaluated in terms of *efficiency*, i.e., how much time it is required to "fix" agents, *effectiveness*, i.e., how compliant is the repaired agent to the stated safety requirements, and *invasiveness*, i.e., how strong is the performance degradation of repaired agents with respect to the original ones. Our empirical evaluation shows that probabilistic model checking is essential to achieve safety guarantees that cannot be obtained using RL alone.

II. Reaching objects with iCub

Our case study is about the humanoid iCub [8] learning to reach for an object (*target*) placed on a table, without damaging other objects (*obstacles*). The task is accomplished with the iCub acting at two levels: A low-level controller ensures that the trajectory of the robot is smooth and efficient – see [12].[1] A corresponding high-level controller has to choose an action suggested by the learner. State-space at this level is discrete and the overall hierarchical structure ensures practical feasibility and scalability – see, e.g., [13].[2] In particular, the first three dimensions of state space represent the Cartesian coordinates, and the fourth dimension encodes orientation. The action-space is a discretized 5-D space, wherein the first four dimensions coincide with the state-space, and the fifth dimension is the time taken for action execution. Hence, constrained reaching movement is modeled as a spatio-temporal task. For sake of simplicity, objects are simple convex shapes and orientation was simple enumeration of few standard poses.

We formalize the learning problem considering a finite set S which contains all possible positions in the discretized state-space. The compact sets $S_i, S_t, S_o \subset S$ represent *initial*, *target*, and *obstacle* positions, respectively. We consider a finite set of actions A and, for each state $s \in S$, $A_s \subseteq A$ is the set of all *plausible* actions from state s, i.e., the set of actions which is effectively enabled in that state – for instance, actions that would lead the robot out of the working space will not be plausible in "boundary" states. We denote with $succ(s, a) \in S$ the successor of state s when action a is executed. Given two states $s, t \in S$, the *minimum distance* $d(s, t)$ between them is defined recursively as

$$d = \begin{cases} 0 & \text{if } s = t \\ 1 + \min_{a \in A_s} d(succ(s, a), t) & \text{otherwise} \end{cases} \quad (1)$$

[1]While various principles could be applied to prefer one trajectory over another, the controller implemented in iCub generates minimum-jerk trajectories with desired orientation as fitness function and desired position as constraint.

[2]The choice of the task and related learning are bio-inspired, taking cue from physiological studies on cognitive development of infants [14]. In particular, we have used the fact that infants learn to reach for an object by making series of similar sub-movements, a discrete set of small bell-shaped velocities rather than one large smooth movement.

Since we do not have a model of the underlying transition system, but we can assume stationarity and history-independence, learning to reach objects in the above setting can be reduced to the solution of a goal-directed, deterministic, uninformed reinforcement learning problem. In order to guarantee convergence in finite time, it is sufficient to have $\forall s \in S_i, g \in S_t, d(s, g) < \infty$ – see,e.g. [15] – which is clearly satisfied in our case. Following [15], a nominal penalty is attached to each action, which also helps in making such problems tractable.

Given the problem as stated above, a policy for implementing reaching on iCub can be obtained using the *SARSA* [16] RL algorithm. In our learning setup we know that there exists a tight bound of $O(|S| \sum_{s \in S} |A_s|)$ on the number of trials required to converge to the shortest possible path, i.e., an optimal control policy [17]. However, since the state-action space is about 2.1 million elements, a naive schema rewarding goal-state alone would require about 12 billion trials to converge. We thus consider the following, more elaborate, but also more effective rewarding schema. Let us assume that the simulator can detect collisions and end-of-episode, and that such conditions are signalled by flags *collision* and *reset*, resepctively. At each time step, the reward received is:

$$R = \begin{cases} R_t + f(T) & \text{if only } \textit{reset}\text{=TRUE} \\ R_o & \text{if } \textit{collision}\text{=TRUE} \\ R_n + g(d) & \text{if all flags are FALSE} \end{cases} \quad (2)$$

where T is the number of time-steps required to reach the goal, and $f : \mathbb{N} \to \mathbb{R}$ is a monotonic bounded-downwards function defined as $f(T) = max(-C_1, 1 - e^{C_2 \cdot T})$, with $C_1 > 0$ and C_2 that can be seen as negative utility for longer actions; d is the distance of the current state from the goal, and $g : \mathbb{N} \to \mathbb{R}$ is a monotonic bounded-downwards function defined as $g(d) = min(K_1, \frac{K_2}{1-e^{-d}})$ with $K_1 > K_2 > 0$, and K_2 representing how preferable is a closer location with respect to a farther one considering the goal. R_n is a small penalty imposed on each action in order to quicken convergence of learning. R_t is the reward agent receives when the target is reached, whereas R_o is the negative reward for hitting the obstacle, in both cases the episode comes to an end. To summarize, an episode starts with initial state $s \in S_i$, continues the episode by taking actions and noticing next states, and comes to end when either $s \in S_o$ or $s \in S_t$ or simply upper bound on d is reached. No extra rewards are levied in the last case.

For iCub to learn effective policies in the above setup, various conflicting goals should be taken into account. While a straight movement toward the target could fetch a higher reward because of $f(T)$, it might also entail a lower reward R_o on account of hitting the obstacle. The other rewards could also interfere with one another. Hence, naive way of providing infinitely negative reward for unsafe states, would guarantee safe behavior but significantly hamper the prospects of effective learning. In other words, it is not straightforward to devise a learning algorithm that ensures task completion with desired level of probabilistic guarantee

against collisions. Since our learning algorithm depends on Q-value of states, one could change the update rule to reflect frequency of unsafe states as was done in [18]. However, this relation between rewarding function and desired safety limit would still remain implicit and hard to implement. Notice that our interpretation of collision is that of a fatal event. This is because, on the real robot, collisions may not be detected timely, or may not even be detected at all, so their occurrence should be avoided at all times.

III. Probabilistic Model Checking

RL techniques model the interaction between the robot and the environment as a Markov Decision Process (MDP) wherein an agent can perceive a set of distinct states S of its environment and has a set of actions A that it can perform. At each discrete time step i, the agent senses state s_i, and performs action a_i. The environment responds by giving the agent a reward R_i and by producing a transition to the succeeding state s_{i+1}. For all states, RL yields a policy $\pi : S \to A$ which suggest an action to be perfomed, so that the cumulative reward will be (close to) optimal. In order to demonstrate that safety requirements are satisfied, we need to combine π with the underlying MDP to yield a *discrete-time Markov chain* (DTMC) describing all possible interactions of the robot with the environment. Analyzing the DTMC, we compute the safety level of π and, in case of an unsatisfactory answer, we attempt to fix it. This can be done in the framework of *Probabilistic model checking* (PMC) – see, e.g., [7], for a comprehensive survey. Briefly stated, in PMC we are given a DTMC $\mathcal{M}$ and a requirement φ, and we establish whether φ holds in $\mathcal{M}$ by means of an (implicit) exhaustive exploration of the possible executions of $\mathcal{M}$.

Following [7], given a set of Boolean facts AP, a DTMC is defined as a tuple $(W, \overline{w}, \mathbf{P}, L)$ where

- W is a finite set of *states*
- $\overline{w}$ is the *initial state*
- $\mathbf{P} : W \times W \to [0,1]$ is the *transition probability matrix*.[3]
- $L : W \to 2^{AP}$ 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 Boolean facts from AP which are true in that state. An example of such a fact is a flag *bad* which is true in all the states which are undesirable.

Requirements for DTMCs can be expressed in terms of Probabilistic Computational Tree Logic (PCTL) formulas, whose syntax is defined considering the set Σ of *state formulas*, and the set Π of *path formulas*. Σ is defined inductively as:

- if $\varphi \in AP$ then $\varphi \in \Sigma$;
- $\top \in \Sigma$; if $\alpha, \beta \in \Sigma$ then also $\alpha \wedge \beta \in \Sigma$ and $\neg\alpha \in \Sigma$
- $\mathcal{P}_{\bowtie p}[\psi] \in \Sigma$ where $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0,1]$ and $\psi \in \Pi$.

Notice that path formulas only occur as the parameter of the *probabilistic path operator* $\mathcal{P}_{\bowtie p}[\psi]$. The set Π contains exactly the expressions of type $X\alpha$ (*next*), $\alpha \mathcal{U}^{\leq k} \beta$ (*bounded until*) and $\alpha \mathcal{U} \beta$ (*until*) where $\alpha, \beta \in \Sigma$ and $k \in \mathbb{N}$. We also abbreviate $\top \mathcal{U} \beta$ as $\mathcal{F}\beta$ (*eventually*) and $\top \mathcal{U}^{\leq k} \beta$ as $\mathcal{F}^{\leq k}\beta$ (*bounded eventually*). A *path* is defined as a non-empty sequence of states $w_0, w_1, w_2, \ldots$ where $w_i \in W$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. We denote the ith state of τ by $\tau(i)$. Paths can be either finite of infinite. In case of finite paths, the *length* of the path is just the number of states in the path, and we denote with $Path_w$ the set of (infinite) paths starting from w.

Given a DTMC $\mathcal{M} = (W, \overline{w}, \mathbf{P}, L)$ and a state $w \in W$, let $\mathcal{M}, w \models \varphi$ denote that $\varphi \in \Sigma$ is satisfied in s. Similarly, we write $\mathcal{M}, \tau \models \psi$ for a path τ of $\mathcal{M}$ satisfying a formula $\psi \in \Pi$. When $\mathcal{M}$ is clear from the context, we omit it and write $w \models \varphi$ and $\tau \models \psi$, respectively. The semantics of path formulas can thus be defined as follows:

- $\tau \models X\alpha$ exactly when $\tau(1) \models \alpha$;
- $\tau \models \alpha \mathcal{U}^{\leq k} \beta$ exactly when there is some $i \leq k$ such that $\tau(i) \models \beta$ and $\tau(j) \models \alpha$ for all $j < i$;
- $\tau \models \alpha \mathcal{U} \beta$ exactly when there is some $k \geq 0$ such that $\tau \models \alpha \mathcal{U}^{\leq k} \beta$.

Accordingly, the semantics of state formulas can be stated as follows:

- $w \models a$ with $a \in AP$ iff $a \in L(w)$;
- $w \models \top$ for all $w \in W$; $w \models \alpha \wedge \beta$ exactly when $w \models \alpha$ and $w \models \beta$; $w \models \neg\alpha$ exactly when $w \models \alpha$ does not hold;
- $w \models \mathcal{P}_{\bowtie p}[\psi]$ exactly when $Prob_w(\{\tau \in Path_w \mid \tau \models \psi\}) \bowtie p$

Intuitively, $Prob_w(\ldots)$ is the probability that the paths in $Path_w$ that also satisfy ψ are taken. $Prob_w(\ldots)$ is provably measurable for all possible PCTL path formulas, and it can be computed by the transition probability matrix $\mathbf{P}$ – see [7] for details.

IV. Analyzing and fixing policies

In this section, we describe how to obtain a DTMC $\mathcal{M}$ representing expected interactions of the robot with the environment, we discuss which PCTL properties are to be checked on $\mathcal{M}$ to assess safety, and how counterexamples of such properties can aid in fixing policies. To this end, we assume that states and transitions explored while learning are logged in a table $T : S \times A \to 2^{S \times \mathbb{N}}$, where S denotes the set of states perceived by the agent, and A denotes the set of possible actions. 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 the 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 $(s'; n)$ such that $n > 0$, i.e., state s' appeared at least once in a transition from an explored state s on action a. For all actions

[3] $[0,1]$ denotes a closed sub-interval of $\mathbb{R}$, i.e., every $x \in \mathbb{R}$ such that $0 \leq x \leq 1$.

$a \in A$, 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 $T(s,a) = \perp$.[4] We also keep track of collisions detected while learning with the table $F : S \to \{\text{TRUE}, \text{FALSE}\}$, where $F(s) = \text{TRUE}$ exactly when state s is a terminal state wherein a collision occurred.

A. Encoding agent's behaviors and safety

A DTMC $\mathcal{M} = (W, \overline{w}, \mathbf{P}, L)$, representing the interaction between the agent and the environment, is obtained by combining tables T and F and the Q-table filled by the learning algorithm. The set AP of atomic propositions has just one element bad which denotes undesirable states, i.e., collisions. 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
$$\forall w \in I,\ \mathbf{P}(w_0, w) = \frac{1}{|I|} \tag{3}$$
meaning that w_0 has an equal-probability 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
$$\mathbf{P}(v,w) = \sum_{a \in A} p_Q(v,a,w) \cdot p_T(v,a,w) \tag{4}$$
The probability $p_Q(v, \hat{a}, w)$ is the probability of taking action $\hat{a} \in A$ in state v, and it is computed from $Q(v,a)$ using a *softmax criterion* [1]
$$p_Q(v, \hat{a}, w) = \frac{e^{Q(v,\hat{a})}}{\sum_{b \in A} e^{Q(v,b)}} \tag{5}$$
This is coherent with the fact that action selection on the iCub is carried out with a softmax-based stochastic policy extracted from the Q-table. The probability $p_T(v, \hat{a}, w)$ 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
$$p_T(v, \hat{a}, w) = \frac{n_w}{\sum_{i=1}^{k} n_i} \tag{6}$$
if $(w; n_w) = (w_j; n_j)$ for some $1 \leq j \leq k$, and $p_T(v, \hat{a}, w) = 0$ otherwise.
4) Let $K \subseteq S$ be the set of *sink states* in T, i.e., $s \in K$ exactly when $T(s,a) = \emptyset$ for all $a \in A$. The state w_{ok} is an abstraction of all sink states $s \in K$ such that $F(s) = \text{FALSE}$. We make w_{ok} absorbing by setting $\mathbf{P}(w_{ok}, w_{ok}) = 1$. For each state v, let $\textit{safe}(v) \subseteq K$ be the set of all safe sink states that can be reached from v, i.e., all states $w \in K$ such that there exist $a \in A$ with $(w; n_w) \in T(v,a)$ and $F(w) = \text{FALSE}$; for all states v such that $\textit{safe}(v) \neq \emptyset$, we set $\mathbf{P}(v, w_{ok}) = \sum_{w \in \textit{safe}(v)} \mathbf{P}(v,w)$ and then $\mathbf{P}(v,w) = 0$ for all $w \in \textit{safe}(v)$.
5) Symmetrically to w_{ok}, the state w_{bad} is an abstraction for all unsafe sink states. Therefore we set $\mathbf{P}(w_{bad}, w_{bad}) = 1$ and $L(w_{bad}) = \{bad\}$. For each state v, $\textit{unsafe}(v) \subseteq K$ is the set of all unsafe sink states that can be reached from v, i.e., all states $w \in K$ such that there exist $a \in A$ with $(w; n_w) \in T(v,a)$ and $F(w) = \text{TRUE}$; for all states v such that $\textit{unsafe}(v) \neq \emptyset$, 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 $w \in \textit{unsafe}(v)$.
6) As a last step, with the exception of w_0, we remove from W all *orphan* states, i.e,, states $w \in W$ such that $\mathbf{P}(v,w) = 0$ for all $v \in W$ with $v \neq w$. Notice that, by construction, orphan states will be only sink states that were collapsed to w_{ok} and w_{bad} in the previous two steps.

In particular, we wish to guarantee that the probability of performing an unsafe action is (very) low in $\mathcal{M}$. This amounts to show that the probability of reaching w_{bad} is low, because w_{bad} abstracts all the states wherein a collision occurred during learning. Since w_{bad} is the only state $w \in W$ such that $L(w) = \{bad\}$, this amounts to check
$$\mathcal{M}, w_0 \models \mathcal{P}_{<\sigma}[\mathcal{F}\ bad], \tag{7}$$
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 failure being less than 1%, i.e., checking that $\mathcal{M}, w_0 \models \mathcal{P}_{<0.01}[\mathcal{F}\ bad]$ holds.

B. Fixing agents using counterexamples

We define *agent repair* as a verification-based procedure that modifies the Q-table obtained by RL. Since the agent's policy is extracted from the Q-table, the goal of repair is to fix the agent by altering state-action values. Repair is driven by counterexamples that witness the violation of property (7) for some threshold σ. Since there are no infinite paths in $\mathcal{M}$, property (7) 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
$$Prob(\tau) = \Pi_{j=1}^{k} \mathbf{P}(w_{j-1}, w_j) \tag{8}$$
and extended to a set of paths $\mathcal{C} = \{\tau_1, \ldots, \tau_n\} \subseteq Path_{w_0}$ as
$$Prob(\mathcal{C}) = \sum_{i=1}^{n} Prob(\tau_i) \tag{9}$$
$\mathcal{C}$ is a counterexample of property (7) if, for all $1 \leq i \leq n$, we have $\tau_i = w_{i,0}, w_{i,1}, \ldots, w_{i,k_i}$ with $w_{i,0} = w_0$ and $w_{i,k_i} = w_{bad}$ and $Prob(\mathcal{C}) > \sigma$. Notice that a counterexample $\mathcal{C}$

[4] Notice that playing an action in a state should have a deterministic effect as shown in Section II. However, coarse quantization, as well as slight inaccuracies of the iCub minimum-jerk controller, cause some observable uncertainity in the effects of actions that must be taken into account.

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 whose safety can hardly be improved by finitely-many learning trials. 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 (7) is satisfied. Our implementation leverages two key ideas to accomplsh 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}. In particular, 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. Ensuring that probabilities are shifted towards paths ending with w_{ok} is needed to avoid moving probability mass from one unsafe path to another, yielding no global improvement.

We formalize the intuitions above with the following definitions. Given a path $\tau = w_0, \ldots, w_k$ such that $\tau \in \mathcal{C}$, let $\tau^{-1}(w) : W \to \{0, \ldots, k\}$ be the *index* of w in τ, i.e., $\tau^{-1}(w) = i$ if $w = w_i$ in τ; The *successor of* w *along* τ, denoted by $succ_\tau(w)$, is the state v such that $\tau^{-1}(v) = \tau^{-1}(w) + 1$. Given a counterexample $\mathcal{C}$ and path $\tau = w_0, \ldots, w_k$ such that $\tau \in \mathcal{C}$, we know that $L(w) = \{bad\}$ only if $w = w_k$. Therefore, we can define the *distance to the bad state along* τ as $b_\tau(w) = k - \tau^{-1}(w)$. Given a counterexample $\mathcal{C}$ and a path $\tau \in \mathcal{C}$, for every state $w \in \tau$ we consider a *penalty* function

$$z_\tau(w) = (1 - Prob(\tau)) \cdot (1 - e^{-\frac{b_\tau(w)}{\delta}}) \qquad (10)$$

where $\delta \in \mathbb{R}^+$ is a real constant modulating the decay of the exponential for growing values of $b_\tau(w)$: all other things being equal, the higher is δ, the slower is the convergence of of $z_\tau(w)$ to $1 - Prob(\tau)$ for increasing values of $b_\tau(w)$. Notice that, given two states w, w' along the same path τ such that $b_\tau(w) < b_\tau(w')$, we have $z_\tau(w) > z_\tau(w')$; given two paths $\tau, \tau' \in \mathcal{C}$ such that $Prob(\tau) > Prob(\tau')$, for all states w along τ and all states w' along τ', if $b_\tau(w) = b_{\tau'}(w')$ we have that $z_\tau(w) > z_\tau(w')$. In conclusions, states which are close to the bad state on a path whose probability mass is relatively high, are always more penalized than states which are far from the bad state, or come along a path whose probability mass is relatively small.

Given a model $\mathcal{M}$ built out of tables Q, T, F as described in Section IV, a counterexample $\mathcal{C} = \{\tau_1, \ldots, \tau_n\}$ of the safety property (7) for a given threshold σ, and a 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 back to step (2), unless $\mathcal{C} = \emptyset$ or the number of paths considered so far is higher than θ.
5) Let $\mathcal{M}'$ be the DTMC obtained by T, F and Q' in the usual way; check if $\mathcal{M}'$ fulfills (7); 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 back to step (1).

V. Experimental results

In the experimental setup, the state-space coordinates have the following ranges (in meters): $x \in [-0.35, -0.1]$, $y \in [-0.15, 0.25]$, $z \in [-0.05, 0.15]$ with discretization step of 0.01m, 0.02m and 0.04m, respectively. The orientation $o \in \{-1, 0, 1\}$ considers only three positions: "face down", "face up", and "face left", respectively. This yields a total of 9828 possible states. Additional signals reveal collisions with the obstacle, and reaching in postures other than $o = 1$. As shown in Figure 2 (left), after about 8000 episodes, the robot can reach the ball while simultaneously avoiding contact with the obstacle in most cases. Because of the realistic task execution time in the iCub simulator, this result requires nearly 12 hours of wall-clock time. Moreover, as shown in Figure 2 (right), the progress in safety alongside with learning is quite erratic.

We checked safety of the DTMC obtained considering a policy learned after about 13 hours of wall-clock time on the simulator. The DTMC has 590 states and 2148 transitions, and COMICS is able to complete the safety check in less than 0.01 CPU seconds. As for repair, in Figure 2 we see that data obtained before repair (red line) and after repair (green line) suggest that fixing the policy improves on safety and it is not invasive: starting from an initial $\sigma = 0.0081$ we repair the policy, till we obtain $\sigma = 6.10 \cdot 10^{-5}$. To confirm this observation, we introduce artificial noise in the state feedback. In this way, we can make learning convergence slower, and have a chance to experiment with repair on suboptimal policies. In Table I, we show the results of this experiment. The first row ("Absent") reports data corresponding to Figure 2. From the second row, we start introducing additive gaussian noise to the iCub's hand coordinates. Noise has zero mean and increasing standard deviation s: "Low" corresponds to $s = 0.005m$, "Medium" to $s = 0.01m$, and "High" to $s = 0.02m$. We can observe that noise always impacts negatively on effectiveness and safety, but repair — see last column of the table — is able to fix safety issues without changing effectiveness in a substantial way. Noticeably, for "High" noise, repairing the

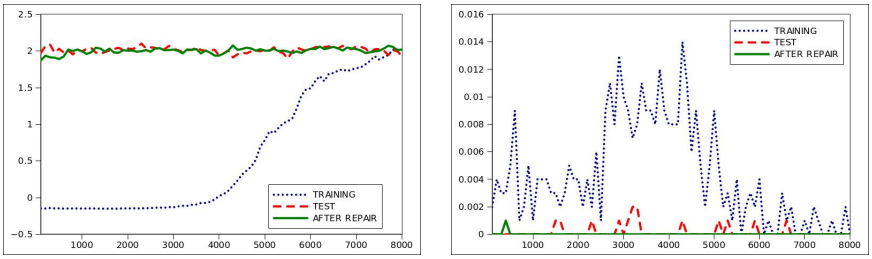

Fig. 2: Performance (left) and safety (right) while learning (blue dotted line), during testing (red dashed line), and after repair (green solid line). The x-axis in both plots indicates the total time (in seconds) for trial episodes. The plot on the left reports the average reward attained per episode, and the plot on the right reports the average number of collisions.

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 I: 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.

policy turns out to have a positive impact on effectiveness as well.

VI. Conclusions

Summing up, we have shown a method based on probabilistic model checking to automate verification and repair of policies learned by reinforcement. Our experimental results support the feasibility of our method in a realistically-sized learning application on the iCub. In particular, we have shown that it is possible to automatically modify learned policies with unsatisfactory failure rates, and achieve failure rates in the order of one-per-ten-thousands, consuming negligible amounts of CPU time, and without decreasing the performance of the iCub in a substantial way. Further details about the tools cosidered in the paper, analyses, data and videos about our experimental sessions can be found in the companion site of the paper at `http://www.mind-lab.it/~shashank/IROS2013/`.

Further developments along this line of research will include more complex setups where the iCub is to guarantee safety at *all* times and learning control programs for real robots, in order to assess scalability issues of verification and repair. Another interesting direction, would be considering safety of agents whose policies map a continuous state space to a (continuous or discrete) action space and lack a suitable discrete abstraction. While this would improve scalability of learning itself, it will also introduce steep challenges from verification methods that would probably require introducing abstraction-refinement techniques in this context.

References

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

[2] J. Gillula and C. Tomlin, "Guaranteed Safe Online Learning via Reachability: tracking a ground target using a quadrotor," in *ICRA*, 2012, pp. 2723–2730.

[3] T. Perkins and A. Barto, "Lyapunov design for safe reinforcement learning," *The Journal of Machine Learning Research*, vol. 3, pp. 803–832, 2003.

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

[5] J. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR," in *International Symposium on Programming*. Springer, 1982, pp. 337–351.

[6] E. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, p. 263, 1986.

[7] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," *Formal methods for performance evaluation*, pp. 220–270, 2007.

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

[9] V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori, "An open-source simulator for cognitive robotics research: the prototype of the icub humanoid robot simulator," in *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*. ACM, 2008, pp. 57–61.

[10] M. Frank, J. Leitner, M. Stollenga, G. Kaufmann, S. Harding, A. Förster, and J. Schmidhuber, "The modular behavioral environment for humanoids and other robots (mobee)," in *Intl. Conference on Informatics in Control, Automation and Robotics*, 2012.

[11] E. Abrahám, N. Jansen, R. Wimmer, J. Katoen, and B. Becker, "DTMC model checking by SCC reduction," in *Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the*. IEEE, 2010, pp. 37–46.

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

[13] R. Schoknecht and M. Riedmiller, "Speeding-up reinforcement learning with multi-step actions," *Artificial Neural NetworksICANN 2002*, pp. 137–137, 2002.

[14] N. E. Berthier, "Learning to reach: A mathematical model." *Developmental psychology*, vol. 32, no. 5, p. 811, 1996.

[15] S. Koenig and R. G. Simmons, "Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains," DTIC Document, Tech. Rep., 1992.

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

[17] S. Koenig and R. G. Simmons, "The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms," *Machine Learning*, vol. 22, no. 1, pp. 227–250, 1996.

[18] O. Mihatsch and R. Neuneier, "Risk-sensitive reinforcement learning," *Machine Learning*, vol. 49, no. 2, pp. 267–290, 2002.