Trace Equivalence Characterization Through Reinforcement Learning

Josée Desharnais, François Laviolette,
Krishna Priya Darsini Moturu, and Sami Zhioua

IFT-GLO, Université Laval, Québec (QC) Canada, G1K-7P4
{first_name.last_name}@ift.ulaval.ca

Abstract. In the context of probabilistic verification, we provide a new notion of trace-equivalence divergence between pairs of Labelled Markov processes. This divergence corresponds to the optimal value of a particular derived Markov Decision Process. It can therefore be estimated by Reinforcement Learning methods. Moreover, we provide some PAC-guarantees on this estimation.

1 Introduction

The general field of this research is program verification. Typically, the goal is to check automatically whether a system (program, physical device, protocol, etc.) conforms to its pre-established specification. Both the specification and the implementation are represented using a formal model (generally a transition system) and then a verification algorithm checks for conformance between the two models. This is usually done with the knowledge of the models of the two processes. Our work fits in a more realistic setting where we want to check the conformance between a specification and an implementation (physical device, piece of code, etc.) for which only the model of the specification is available.

The model we work with is called a *Labelled Markov Process* (LMP) [2]; it is a labelled transition systems where transitions are labelled by an action and weighted by a probability (see Figure 1). The action is meant to be synchronized

Fig. 1. Labelled Markov Processes

through interaction with the environment. Apart from the modelling of intrinsically probabilistic systems like randomized protocols, probabilities are also used

as an abstraction mechanism, for example to hide complex details in suitably chosen probability distributions. The state space of an LMP can be infinite; finite LMPs are also called Probabilistic labelled transition systems or Markov decision processes without rewards. In this paper we will always suppose that it has a tree like representation which, up to bisimulation [2], is always possible. In recent years, a lot of work has been done on pseudo-metrics between probabilistic systems. The reason is that when one wants to compare systems, an equivalence like bisimulation is quite limited: systems are equivalent or they are not. In the presence of probabilities this is even worse because, for one thing, a slight change in the probabilities of equivalent processes will result in non equivalent processes. A pseudometric is an indication of how far systems are. Few pseudometrics have been defined but none of them come with an efficient way to compute it. Moreover, these metrics can only deal with processes whose models are known.

This paper is a first step towards the computation of a suitable measure of non equivalence between processes in the presence of unknown models. We have observed that while verification techniques can deal with processes of about 10^{12} states, Reinforcement Learning (RL) algorithms do a lot better; for example, the *TD-Gammon* program deals with more than 10^{40} possible states [13]. Thus we define a *divergence* notion between LMPs, noted $\text{div}_{\text{trace}}(\cdot \parallel \cdot)$, that can be computed with RL algorithms. We call it a divergence rather than a distance because it is not symmetric and does not satisfy the triangle inequality. However, it does have the important property that it is always positive or zero and it is equal to zero if and only if the processes are *probabilistic trace-equivalent* [9]. Two processes are *probabilistic trace-equivalent* (we will simply say trace-equivalent) if they accept the same sequences of actions with the same probabilities. For example, T_1 and T_2 in Figure 1 accept the same traces: $\varepsilon, a, aa, ab, ac, aac, aca$ but they are not trace-equivalent since $P^{T_1}(aac) = \frac{1}{4}$ whereas $P^{T_2}(aac) = \frac{1}{6}$.

2 The Approach

We first informally expose our approach through a one-player stochastic game. Then, we will define a divergence between a specification model (denoted “Spec”) and a real system (denoted “Impl”) for which the model is not available but on which we can interact (exactly as a black-box). This divergence will be the value of the maximal expected reward obtained when playing this game. The formalization and proofs will follow.

2.1 Defining a Game Using Traces

A *trace* is a sequence of actions (possibly empty) that is meant to be executed on a process. Its execution results in the observation of success or failure of each action¹. For example, if trace *aca* is executed, then four observations are possible:

¹ Note that the execution is sometimes called the trace itself whereas the sequence of actions is called a test.

$a^\times$, $a^\vee c^\times$, $a^\vee c^\vee a^\times$ and $a^\vee c^\vee a^\vee$, where $a^\vee$ means that a is accepted whereas $a^\times$ means that it is refused. To each trace is associated a probability distribution on observations. For example, in T_2 of Figure 1, the observations related to trace aca have the distribution $p_{a^\times} = 0$, $p_{a^\vee c^\times} = \frac{1}{1} \frac{1}{2} = \frac{1}{2}$, $p_{a^\vee c^\vee a^\times} = \frac{1}{1} \frac{1}{2} \frac{3}{4} = \frac{3}{8}$, $p_{a^\vee c^\vee a^\vee} = \frac{1}{8}$. Based on this setting, we have the straightforward result [12]:

Proposition 1. *Two processes are trace-equivalent iff they yield the same probability distribution on observations for every trace.*

In the light of this result, a suitable choice to define our divergence could simply be the maximum value, over all traces τ , of the Kullback-Leibler divergence between the probability distributions on observations when running each trace τ on “Spec” and “Impl”. The Kullback-Leibler divergence between two distributions Q and P is defined as $\text{KL}(Q\|P) := E_{h \sim Q} \ln \frac{1}{P(h)} - E_{h \sim Q} \ln \frac{1}{Q(h)}$ [4]. Unfortunately, because of the high number of possible traces (on huge systems), the maximum value over all Kullback-Leibler divergences is not tractable.

However, $E_{h \sim Q} \ln \frac{1}{Q(h)}$, the entropy of Q , can be seen as a quantification over the likeliness to obtain different observations when interacting with “Spec” and a perfect clone of it (which we call “Clone”). “Clone” is simply a copy of the specification but given in the form of a black-box (exactly as “Impl”). In some sense, $E_{h \sim Q} \ln \frac{1}{P(h)}$ can also be seen as how likely we can obtain different observations when interacting (via some τ) with both “Spec” and “Impl”. Hence, the maximum possible Kullback Leibler divergence should be obtained when executing a suitable trade-off reflecting the fact that the probability of seeing different observations between “Spec” and “Impl” should be as large as possible, and as small as possible between “Spec” and “Clone”. Here is a one-player stochastic game on which a similar tradeoff underlies the optimal solution (see Figure 2).

Fig. 2. Implementation, Specification, and Clone

Game₀: The player plays on “Spec”, starting in the initial state; then

Step 1 : The player chooses an action a .

Step 2 : Action a is run on “Spec”, “Impl” and on a clone of “Spec”.

Step 3 : If a succeeds on the three processes, the new player’s state is the reached state in “Spec”; go to Step 1. Else the game ends and the player

gets a (+1) reward for different observations between “Spec” and “Impl” added up with a (−1) reward for different observations between “Spec” and “Clone”. That is, writing I for “Impl”, C for “Clone” and Obs for observation,

$$R := (Obs.I \neq Obs(Spec)) - (Obs(Spec) \neq Obs.C)$$

where 0 and 1 are used as both truth values and numbers.

For example, if action a is executed on the three LMPs and observation FSS is obtained (i.e., Failure in “Impl”, Successes in “Spec” and “Clone”), then an immediate reward of (+1) is given. Notice that once an action fails in one of the three processes the game is stopped. Hence the only scenario allowing to move ahead is: SSS . It is easy to see that if “Spec” and “Impl” are trace-equivalent, the optimal strategy will have expected reward zero, as wanted. Unfortunately the converse is not true: there are trace-inequivalent LMPs for which every strategy has expected reward zero. Here is a counterexample: consider three systems with one a -transition from one state to another one. The probability of this transition is $\frac{1}{2}$ for the “Spec” (and “Clone”) and 1 for the implementation.

Thus, the maximal possible expected reward of Game_0 will not lead to a notion of trace-equivalence divergence, but we will show that the following slight modification of Game_0 does lead to a suitable notion of divergence.

Definition 1. Game_1 is Game_0 with *Step 1* replaced by

Step 1’ : The player chooses an action and makes a prediction on its success or failure on “Spec”

and the reward function is replaced by

$$R := (Obs(Spec) = \text{Pred})((Obs.I \neq Obs(Spec)) - (Obs(Spec) \neq Obs.C)) \quad (1)$$

For example, if $a^\vee$ is selected and the observation is FSS we obtain a reward of $1((F \neq S) - (S \neq S)) = 1(1 - 0) = 1$, but for FFS , we obtain $0(0 - 1) = 0$.

Observe that because the player has a full knowledge of what is going on in “Spec”, the prediction in Step 1’ together with this new reward signal gives (implicitly) more control to the player. Indeed, the latter has the possibility to reduce or augment the probability to receive a non zero reward if he expects this reward to be negative or positive. Next section will formalize these ideas in a more Reinforcement Learning setting.

2.2 Constructing the MDP $\mathcal{M}$

In artificial intelligence, Markov Decision Processes (MDPs) offer a popular mathematical tool for planning and learning in the presence of uncertainty [10]. MDPs are a standard formalism for describing multi-stage decision making in probabilistic environments (what we called a one-player stochastic games in the preceding section). The objective of the decision making is to maximize a cumulative measure of long-term performance, called the reward.

In an MDP, an agent interacts with an environment at a discrete, low-level time scale. On each time step, the agent observes its current state $s_t \in S$ and chooses an action a_t from an action set A . One time step later, the agent transits to a new state s_{t+1} , and receives a reward r_{t+1} . For a given state s and action a , the expected value of the immediate reward is denoted by $R_{s\ s}^a$ and the transition to a new state s' has probability $Pr_{s\ s'}^{\mathcal{M}}(a)$, regardless of the path taken by the agent before state s (this is the *Markov property*). The goal in solving MDPs is to find a *way of behaving*, or policy, which yields a maximal reward. Formally, a policy is defined as a probability distribution for picking actions in each state. For any policy $\pi : S \times A \rightarrow [0, 1]$ and any state $s \in S$, the *value function* of π for state s is defined as the expected infinite-horizon discounted return from s , given that the agent behaves according to π : $V^\pi(s) := E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t = s\}$ where s_0 is the initial state and γ is a factor between 0 and 1 used to discount future rewards. The objective is to find an optimal policy, π^* which maximizes the value $V^\pi(s)$ of each state s . The *optimal value function*, V^* , is the unique value function corresponding to any optimal policy.

If the MDP has finite state and action spaces, and if a model of the environment is known (i.e., immediate rewards $R_{s\ s'}^a$ and transition probabilities $Pr_{s\ s'}^{\mathcal{M}}(a)$), then DP algorithms (namely policy evaluation) can compute V^π for any policy π . Similar algorithms can be used to compute V^* . RL methods, in contrast, compute approximations to V^π and V^* directly based on the interaction with the environment, without requiring a complete model, which is exactly what we are looking for in our setting.

We now define the MDP on which the divergence between two LMPs will be computed. The model of the first LMP, (called “Spec”) is needed and must be in a tree-like representation. It is a tuple $(States, s_0, Actions, Pr^{Spec})$ where s_0 is the initial state and $Pr_{s\ s'}^{Spec}(a)$ is the transition probability from state s to state s' when action a has been chosen (for example, in Figure 2, $Pr_{1\ 4}^{Spec}(a) = \frac{2}{4}$). Since it has a tree-like structure, for any state $s \in States$, there is a unique sequence of actions (or trace, denoted $tr.s$) from s_0 to s . For the second LMP (called “Impl”), only the knowledge of all possible conditional probabilities $P^I(a^\vee | \tau)$ and $P^I(a^\times | \tau)$ of observing the success or failure of an action a given any successfully executed trace τ is required. For example, in Figure 2, $P^C(c^\vee | aa) = \frac{2}{4} \div (\frac{2}{4} + \frac{1}{4}) = \frac{2}{3}$. Finally, we write $P^C(a^\vee | \tau)$ and $P^C(a^\times | \tau)$ for the conditional probabilities of a copy of the first LMP (called “Clone”): this is for readability and is no additional information since $P^C(a^\vee | \tau) = P^S(a^\vee | \tau)^2$.

Definition 2. Given “Impl”, “Spec”, and “Clone”, the set of states of the MDP $\mathcal{M}$ is $S := States \cup \{Dead\}$, with initial state s_0 ; its set of actions is $Act := Actions \times \{\vee, \times\}$. The next-state probability distribution is the same for $a^\vee$ and $a^\times$ (which we represent generically with $a^\cdot$); it is defined below, followed with the definition of the reward function.

² Note that the complete knowledge of $P^I(\cdot | \cdot)$ and $P^C(\cdot | \cdot)$ is needed to construct the MDP, but if one only wants to run a Q-learning algorithm on it, only a possibility of interaction with “Clone” and “Impl” is required.

$$Pr_{s s'}^{\mathcal{M}}(a^-) := \begin{cases} Pr_{s s'}^{Spec}(a) P^I(a^\vee | tr.s) P^C(a^\vee | tr.s) & \text{if } s' \neq Dead \\ 1 - P^S(a^\vee | s) P^I(a^\vee | tr.s) P^C(a^\vee | tr.s) & \text{if } s' = Dead \end{cases}$$

$$R_{s s'}^{a^-} := \begin{cases} 0 & \text{if } s' \neq Dead \\ \frac{1}{Pr_{s s'}^{\mathcal{M}}(a^-)} P^S(a^- | s) \Delta_{tr.s}^{a^-} & \text{if } s' = Dead \end{cases}$$

where $\bullet P^S(a^\vee | s) := \sum_{s' \in States} Pr_{s s'}^{Spec}(a)$, and $P^S(a^\times | s) := 1 - P^S(a^\vee | s)$,
 $\bullet \Delta_\tau^{a^\vee} := P^I(a^\times | \tau) P^C(a^\vee | \tau) - P^I(a^\vee | \tau) P^C(a^\times | \tau)$ and $\Delta_\tau^{a^\times} := -\Delta_\tau^{a^\vee}$.

The additional *Dead* state in the MDP indicates the end of an episode and is reached if at least one of the three systems refuses the action. This is witnessed by the next state probability distribution.

Let us now see how the reward function $R_{s s'}^{a^-}$ accords with Game_1 presented in the previous section. The simplest case is if the reached state is not *Dead*: it means that the action succeeded in the three systems (*SSS*) which indicates a similarity between them. Consequently the reward is zero for both $a^\vee$ and $a^\times$, as indicated by Step 3 of the game. Let us now consider the case $s' = Dead$. We want the following relation between $R_{s s'}^{a^-}$ and $R_{s s}^{a^-}$, precisely because the latter represents the average reward of running action a^- on a state s :

$$R_{s s}^{a^-} = Pr_{s Dead}^{\mathcal{M}}(a^-) R_{s Dead}^{a^-} + \sum_{s' \in S \setminus \{Dead\}} Pr_{s s'}^{\mathcal{M}}(a^-) \underbrace{R_{s s'}^{a^-}}_0 = Pr_{s Dead}^{\mathcal{M}}(a^-) R_{s Dead}^{a^-} \quad (2)$$

However Game_1 and Equation (1) give us another computation for $R_{s s}^{a^-}$. In the case where $a^- = a^\vee$ is chosen, then the reward is computed only if a succeeds in “Spec”, that is, we get a (+1) reward on observation *FSS*, a (−1) on *SSF*, and (+0) on observations *SSS* or *FSF*. If a fails, the reward is (+0). Thus,

$$R_{s s}^{a^\vee} = \underbrace{P^I(a^\times | tr.s) P^S(a^\vee | s) P^C(a^\vee | tr.s)}_{FSS \mapsto (+1)} - \underbrace{P^I(a^\vee | tr.s) P^S(a^\vee | s) P^C(a^\times | tr.s)}_{SSF \mapsto (-1)}$$

$$= P^S(a^\vee | s) \Delta_{tr.s}^{a^\vee}.$$

In the case where $a^\times$ is chosen, the opposite mechanism is adopted, that is, the reward is computed only if a fails in “Spec” and hence only the situations *SFF* (+1) and *FFS* (−1) are relevant. We therefore obtain the following equation:

$$R_{s s}^{a^\times} := P^S(a^- | s) \Delta_{tr.s}^{a^-}. \quad (3)$$

The reward $R_{s Dead}^{a^-}$ in Definition 2 is obtained by combining Equations (2) and (3).

2.3 Value of a Policy in $\mathcal{M}$

In order to define a formula for policy evaluations on any state s in our setting, we start from the Bellman Equation [13].

$$\begin{aligned}
V^\pi(s) &= \sum_{a^- \in Act} \pi(s, a^-) \sum_{s' \in S} Pr_{s s'}^{\mathcal{M}}(a^-) (R_{s s'}^{a^-} + \gamma V^\pi(s')) \\
&= \sum_{a^- \in Act} \pi(s, a^-) (Pr_{s Dead}^{\mathcal{M}}(a^-) R_{s Dead}^{a^-} + \sum_{s' \in S \setminus \{Dead\}} Pr_{s s'}^{\mathcal{M}}(a^-) (0 + \gamma V^\pi(s'))) \\
&= \sum_{a^- \in Act} \pi(s, a^-) (R_{s s}^{a^-} + \sum_{s' \in S \setminus \{Dead\}} Pr_{s s'}^{\mathcal{M}}(a^-) \gamma V^\pi(s')) \tag{4}
\end{aligned}$$

where the second equality follows from Equation (2).

Now, let us denote by S_i the set of states of $S \setminus \{Dead\}$ at depth i (i.e., that can be reached from s_0 in exactly i steps³). We call *trace-policy* a deterministic (not stochastic) policy for which the same action is selected for states at the same depth. A trace-policy can thus be represented by a trace of the MDP (e.g.: $\pi = a_1^\vee a_2^\times a_3^\times a_4^\vee a_5^\times$). Let $P_j^{\mathcal{M}+\pi}(s)$ be the probability to be in s , j steps away from the initial state when following trace-policy π . For example, starting from the LMPs of Figure 2 with trace-policy $\pi := a^\vee a^\vee b^\vee$, $P_2^{\mathcal{M}+\pi}(4) = Pr_{01}^{Spec}(a) Pr_{14}^{Spec}(a) = \frac{2}{4}$, but $P_3^{\mathcal{M}+\pi}(7) = 0$. The value of the trace-policy $\pi = a_1^\vee a_2^\times \dots a_n^\times$ on s_0 is thus determined as follows from Equation (4):

$$\begin{aligned}
V^\pi(s_0) &= \sum_{a^- \in Act} \pi(s_0, a^-) (R_{s_0 s}^{a^-} + \sum_{s' \in S \setminus \{Dead\}} Pr_{s_0 s'}^{\mathcal{M}}(a^-) \gamma V^\pi(s')) \\
&= R_{s_0 s}^{a_1^\vee} + \sum_{s' \in S \setminus \{Dead\}} Pr_{s_0 s'}^{\mathcal{M}}(a_1^\vee) \gamma \left(R_{s' s}^{a_2^\times} + \sum_{s'' \in S} Pr_{s' s''}^{\mathcal{M}}(a_2^\times) \gamma V^\pi(s'') \right) \\
&\quad \vdots \\
&= \sum_{i=0}^{n-1} \sum_{s \in S_i} P_{i+1}^{\mathcal{M}+\pi}(s) \gamma^i R_{s s}^{a_{i+1}^\times} \tag{5}
\end{aligned}$$

$$\begin{aligned}
&= \sum_{i=0}^{n-1} \gamma^i \sum_{s \in S_i} P_{i+1}^{\mathcal{M}+\pi}(s) P^S(a_{i+1}^\times | s) \Delta_{tr.s}^{a_{i+1}^\times} \\
&= \sum_{i=0}^{n-1} \gamma^i P^S(a_{i+1}^\times | a_1 \dots a_i) \Delta_{a_1 \dots a_i}^{a_{i+1}^\times} \tag{6}
\end{aligned}$$

Note that there is an abuse of language when we say that a trace-policy is a policy of the MDP, because we assume that after the last action of the sequence, the episode ends even if the *Dead* state is not reached.

3 Theorems and Definition of $\text{div}_{\text{trace}}(\cdot \parallel \cdot)$

We prove in this section that solving the MDP induced by a specification and an implementation yields a trace equivalence divergence between them which is positive and has value 0 if and only if the two processes are trace-equivalent.

Let us first state three easy lemmas which will be useful in this section.

³ Since “Spec” is assumed tree-like, the S_i ’s are pairwise disjoint.

Lemma 1. *The following are equivalent.*

1. $P^I(a^\times | \tau) = P^C(a^\times | \tau)$.
2. $P^I(a^\vee | \tau) = P^C(a^\vee | \tau)$.
3. $\Delta_\tau^{a^-} = 0$.

Lemma 2. *Let $\pi = a_1^- a_2^- \dots a_n^-$ be a trace-policy, and $\tau_\pi = a_1 a_2 \dots a_n$ its corresponding trace. Then for any $a^- \in \text{Act}$, writing πa^- for $a_1^- a_2^- \dots a_n^- a^-$, we have*

$$(V^{\pi a^-}(s_0) = V^\pi(s_0)) \text{ iff } (P^I(a^- | \tau_\pi) = P^C(a^- | \tau_\pi) \text{ or } P^S(a^- | \tau_\pi) = 0).$$

Proof. By Equation (6), we have $V^{\pi a^-}(s_0) - V^\pi(s_0) = \gamma^n \Delta_{\tau_\pi}^{a^-} P^S(a^- | \tau_\pi)$; this implies the result because by Lemma 1, $\Delta_{\tau_\pi}^{a^-} = 0 \Leftrightarrow P^I(a^- | \tau_\pi) = P^C(a^- | \tau_\pi)$. $\square$

Lemma 3. $\forall a^- \in \text{Act}, s \in S \setminus \{\text{Dead}\} \quad P^S(a^- | tr.s) = 0 \Rightarrow P^S(a^- | s) = 0$

Proof. Because “Spec” is tree-like, the probability of reaching s by following $tr.s$ is strictly positive. By contraposition, assume that the probability of observing a^- from s is strictly positive, then so is the probability of observing a^- after $tr.s$. $\square$

From now on, we will use the notation $\overline{a^-}$ to mean the opposite of a^- , that is, if $a^- = a^\vee$ then $\overline{a^-} = a^\times$ and if $a^- = a^\times$ then $\overline{a^-} = a^\vee$.

Theorem 1. *The following are equivalent:*

- (i) *The specification and implementation processes are trace-equivalent.*
- (ii) $\forall a^- \in \text{Act}, s \in S \setminus \{\text{Dead}\} \quad R_s^{a^-} = 0$
- (iii) $\forall \text{ trace-policy } \pi \quad V^\pi(s_0) = 0$

Proof. (i) $\Rightarrow$ (ii). Since $tr.s$ is a trace $\forall s \neq \text{Dead}$, and since $P^S(. | .) = P^C(. | .)$,

$$\begin{aligned} P^S(. | .) = P^I(. | .) &\Rightarrow \forall a^- \in \text{Act}, s \in S \setminus \{\text{Dead}\} \quad P^C(a^- | tr.s) = P^I(a^- | tr.s) \\ &\Leftrightarrow \forall a^- \in \text{Act}, s \in S \setminus \{\text{Dead}\} \quad \Delta_{tr.s}^{a^-} = 0 \quad (\text{by Lemma 1}) \\ &\Rightarrow \forall a^- \in \text{Act}, s \in S \setminus \{\text{Dead}\} \quad R_s^{a^-} = 0 \quad (\text{by Equation (3)}) \end{aligned}$$

(ii) $\Rightarrow$ (i). Let τ be a trace and $a^- \in \text{Act}$. Case 1: there exists an $s \in S \setminus \{\text{Dead}\}$ such that $\tau = tr.s$. Then by (ii) and Equation (3), we have $\Delta_{tr.s}^{a^-} P^S(a^- | s) = 0$. This implies either $\Delta_{tr.s}^{a^-} = 0$ or $P^S(a^- | s) = 0$. By Lemma 1, we know that the first case implies the result. The second case requires to see the opposite action. Indeed, $P^S(a^- | s) = 0$ implies that $P^S(\overline{a^-} | s) = 1$. By (ii), $R_s^{\overline{a^-}} = 0$. By the same argument, we can deduce that either $\Delta_{tr.s}^{\overline{a^-}} = 0$ or $P^S(\overline{a^-} | s) = 0$, and therefore that $\Delta_{tr.s}^{\overline{a^-}} = 0$. Since $\overline{\overline{a^-}} = a^-$, by Lemma 1 (replacing a^- by $\overline{a^-}$), we have $P^I(a^- | tr.s) = P^C(a^- | tr.s)$, as wanted. Case 2: if no such s exists, then τ has probability zero in “Spec” and trivially $P^S(a^- | \tau) = 1 = P^I(a^- | \tau)$ as wanted.

(ii) $\Rightarrow$ (iii) follows from Equation (5).

(iii) $\Rightarrow$ (ii). Fix $a^- \in \text{Act}$ and $s \in S \setminus \{\text{Dead}\}$, and let $n \geq 0$ such that $s \in S_n$, and let $a_1 \dots a_n = tr.s$. Now, define $\pi = a_1^\vee \dots a_n^\vee$. By (iii) and Equation (6), we have $0 = 0 - 0 = V^{a_1^- \dots a_n^- a^-}(s_0) - V^\pi(s_0) = \gamma^n \Delta_{tr.s}^{a^-} P^S(a^- | tr.s)$. By Lemma 3, $\Delta_{tr.s}^{a^-} P^S(a^- | s) = 0$, which, by Equation 3, implies the result. $\square$

Theorem 2. *Two LMPs are not trace-equivalent if and only if $V^\pi(s_0) > 0$ for some trace-policy π .*

Proof. ($\Rightarrow$): by Theorem 1, we have that $V^\pi(s_0) \neq 0$ for some trace-policy $\pi = a_1^- a_2^- \dots a_n^-$. Define $J := \{j \in \{1, \dots, n\} \mid P^C(a_j^- | a_1 \dots a_{j-1}) \Delta_{a_1 \dots a_{j-1}}^{a_j^-} < 0\}$, and note that $P^C(a_j^- | a_1 \dots a_{j-1}) > 0$ and $\Delta_{a_1 \dots a_{j-1}}^{a_j^-} < 0$ for any $j \in J$. Thus, for any such j , $\Delta_{a_1 \dots a_{j-1}}^{a_j^-} > 0$ (see Definition 2). Let π_1 be the policy obtained from π by replacing each action $a_j^- \in J$ by its opposite. Then, by Equation (6), $V^{\pi_1}(s_0) > 0$, as desired. ($\Leftarrow$): from Theorem 1. $\square$

Lemma 4. *For every trace-policy π and every $a^- \in \text{Act}$,*

$$V^\pi(s_0) \leq V^{\pi a^-}(s_0) \quad \text{or} \quad V^\pi(s_0) \leq V^{\pi \overline{a^-}}(s_0).$$

Proof. As for Theorem 2, the result follows from Equation (6) and the fact that $\Delta_{\tau_\pi}^{a^-} = -\Delta_{\tau_\pi}^{\overline{a^-}}$ where τ_π is the trace corresponding to π . $\square$

Theorem 3. *Let $\mathcal{M}$ be the MDP induced by “Spec”, “Impl”, and “Clone”. If $\gamma < 1$ or $|\mathcal{M}| < \infty$ then $V^*(s_0) \geq V^\pi(s_0)$ for any trace-policy π .*

Proof. As explained in the preceding section, a trace policy $a_1^- \dots a_n^-$ is not a policy of $\mathcal{M}$ but of the sub-MDP whose state space is $S \setminus \cup_{i=n+1}^\infty S_i$. If $|\mathcal{M}| < \infty$, the result is a direct consequence of Lemma 4. Otherwise, it is sufficient to show

$$\forall \epsilon > 0 \quad \forall \text{ trace-policy } \pi \quad \exists \text{ policy } \pi' \quad \text{such that} \quad |V^{\pi'}(s_0) - V^\pi(s_0)| < \epsilon.$$

Let $\epsilon > 0$ and $\pi = a_1^- \dots a_n^-$ be a trace-policy. Because of Lemma 4, w.l.o.g., we may suppose n to be large enough to satisfy $\sum_{i=n+1}^\infty \gamma^i < \epsilon$. Since on each episode, the reward signal is (-1) , (0) or $(+1)$, it is easy to see that any policy π' of $\mathcal{M}$ that coincides with π on $\mathcal{M}_\pi$ will have the desired property. $\square$

We can now give the definition of the central notion of this paper.

Definition 3. Let “Spec” and “Impl” be two LMPs and $\mathcal{M}$ their induced MDP. We define their *trace-equivalence divergence* as

$$\text{div}_{\text{trace}}(\text{“Spec”} \parallel \text{“Impl”}) := V^*(s_0).$$

Clearly, $\text{div}_{\text{trace}}(\text{“Spec”} \parallel \text{“Impl”}) \geq 0$, and $= 0$ iff “Spec” and “Impl” are trace-equivalent.

4 Implementation and PAC Guarantees

As mentioned in Section 2, the full model of the MDP might not be available. Therefore, it is not appropriate to use a Dynamic Programming [1] algorithm such as value iteration [13] to solve the MDP. Instead, we use a Q-Learning algorithm [15]. Q-Learning is an off-policy Temporal Difference (TD) control algorithm which directly approximates $V^*(s_0)$. The algorithm has been proven to converge to the optimal value [8, 14, 16]. Moreover, some results about its

convergence rates have been proposed [5]. However, in the field of verification, the main goals are (*) to find the difference between the implementation of a system and its specification and also (**) to have a guarantee on the fact that this difference is very small in the case where we do not find any such difference during the investigation. Hence, from that perspective, a PAC-guarantee for the Q-Learning algorithm is the most appropriate tool.

Definition 4. We say that we have a *PAC (Probably Approximately Correct) guarantee* for a learning algorithm on an MDP $\mathcal{M}$ if, given an a priori precision $\epsilon > 0$ and a maximal probability error δ , there exists a function $f(\mathcal{M}, \epsilon, \delta)$ such that if the number of episodes is greater than $f(\mathcal{M}, \epsilon, \delta)$, then

$$Prob\{|\overline{V^{\hat{\pi}}}(s_0) - V^*(s_0)| \leq \epsilon\} \geq 1 - \delta \quad (7)$$

where $\hat{\pi}$ is the policy returned by the Q-learning algorithm and $\overline{V^{\hat{\pi}}}(s_0)$ is the estimation of $V^*(s_0)$ given by this algorithm.

The Q-learning algorithm does have a PAC guarantee [11], but the function $f(\mathcal{M}, \epsilon, \delta)$ is very difficult to compute, which makes this guarantee unusable in practice. The Fiechter RL algorithm [6] comes with a simpler PAC guarantee and hence one can use it in the current setting. The main drawback of the Fiechter algorithm remains its inefficiency compared to Q-Learning.

However we can still reach goal (*) using any RL learning algorithm. Indeed, in the case where the two processes are not trace-equivalent, we can guarantee a bottom bound for the optimal value using Hoeffding inequality [7] based on the following idea. Let $\hat{\pi}$ be the policy returned by the RL algorithm. Let $\overline{V^{\hat{\pi}}}(s_0)$ be the estimation of $V^{\hat{\pi}}(s_0)$ using a Monte Carlo [13] algorithm with m episodes. Given $\epsilon, \delta \in]0, 1[$, according to the Hoeffding inequality, if $m \geq \frac{1}{\epsilon^2} \ln(\frac{2}{\delta})$, we have Equation (7) with $V^*(s_0)$ replaced by $\overline{V^{\hat{\pi}}}(s_0)$. Since $V^{\hat{\pi}}(s_0)$ never exceeds the optimal value $V^*(s_0)$, we have the following PAC guarantee: $Prob\{\overline{V^{\hat{\pi}}}(s_0) - V^*(s_0) \leq \epsilon\} \geq 1 - \delta$. Note that, in addition, the algorithm returns the policy $\hat{\pi}$ that proves the result. A specific difference between the implementation and its specification is therefore identified, orienting the debugging phase that would follow. This is a major difference with the traditional testing approach consisting of simply running an a priori defined test during the debugging phase.

Finally, observe that if $\overline{V^{\hat{\pi}}}(s_0)$ is too close to zero, or exactly zero, then the Hoeffding inequality gives no guarantee on the trace equivalence or inequivalence. Recall that this difficulty occurs only if we use an algorithm without a tractable PAC guarantee: the problem does not happen with the Fiechter RL algorithm. Nevertheless, even with Q-Learning, we think that our approach is better than traditional testing because, as explained above, it can find new tests by its own.

4.1 Experimental Results

The approach described so far has been implemented using Java. Two action selection algorithms have been experimented: ϵ -greedy and SoftMax. For both methods, we tried several functions to decrease the ϵ (resp. the τ) values. The

combination that produced the best results is SoftMax such that the temperature τ is decreasing from 0.8 to 0.01 according to the function : $\tau = \frac{k}{\text{currentEpisod}+l}$ (variables k and l are constants). The learning rate α (also called step size) must decrease in order to assure convergence of the Q-Learning algorithm. We tried several decreasing functions and the best convergence results are with $\frac{1}{x}$ where x is the number of times the state-action has been visited. The discount factor γ is fixed to 0.8 in our algorithm. The two following graphics show how the

Q-Learning algorithm converges to the optimal value. This value is, in our setting, the trace-equivalence divergence between the specification and the implementation processes. In the above graphics, we tracked the optimal value in one execution of 10000 episodes. In graphic (a), it is easy to see that the estimated value gets close to the real optimal value (0.132 represented by the line) as the number of episodes increases. Graphic (b), however, is obtained by running the algorithm on trace-equivalent processes. It shows how the estimated value for that case converges to zero.

5 Conclusion

The main contribution of this paper is a completely new approach to estimate how far two LMPs are from being trace-equivalent. Indeed, we introduce a notion of trace-equivalence divergence $\text{div}_{\text{trace}}(\cdot \parallel \cdot)$ that can be estimated via some Monte-Carlo estimation using Reinforcement Learning algorithms. Traditional approaches, on the other hand, are based on costly complete calculations on the models. The advantage of using an RL approach therefore opens a way for analyzing very huge systems and even infinite ones.

In this paper, we showed that $\text{div}_{\text{trace}}(\cdot \parallel \cdot)$ is a real divergence operator and gave some related PAC-guarantees. However, we did not have enough room to show how $\text{div}_{\text{trace}}(\cdot \parallel \cdot)$ is increasing as the LMPs are farther from being trace-equivalent. This is intuitively clear. The proof, however, is related to the fact

that, as for Kullback-Leibler divergence, the divergence D_π induced by any fixed trace-policy π is of Bregman [3] (here the Bregman function related to D_π is $\sum_{i \in I} x_i (1 - \frac{1}{2}x_i)$).

For future work, we want to improve our PAC-guarantees, and even obtain a PAC-guarantee that will not depend on the state space S of the MDP $\mathcal{M}$. We also want to modify the construction of $\mathcal{M}$, in order to speed up the calculation. For example, we could generalize our notion of divergence in a setting where the FFF observation does not stop the episode anymore. Finally, since the LMP formalism is mathematically quite similar to the MDP and HMM (Hidden Markov Model) formalisms, the next step will be to apply our approach on these formalisms.

References

1. R. E. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.
2. R. Blute, J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. In *Proc. of the Twelfth IEEE Symposium On Logic In Computer Science, Warsaw, Poland, 1997*.
3. Y. Censor. *Parallel Optimization: Theory, Algorithms, Applications*. Oxford University Press, 1997.
4. T. M. Cover and J. A. Thomas. *Elements of Information Theory*, chapter 12. Wiley, 1991.
5. E. Even-Dar and Y. Mansour. Learning rates for Q-learning. In *COLT '01/EuroCOLT '01: Proc. of the 14th Annual Conference on Computational Learning Theory*, pages 589–604, London, UK, 2001. Springer-Verlag.
6. C. N. Fiechter. *Design and Analysis of Efficient Reinforcement Learning Algorithms*. PhD thesis, Univ. of Pittsburgh, 1997.
7. W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.
8. T. Jaakkola, M. I. Jordan, and S. P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In J. D. Cowan, G. Tesauro, and J. Alsppector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 703–710. Morgan Kaufmann Publishers, 1994.
9. C.-C. Jou and S. A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR 90 First International Conference on Concurrency Theory*, number 458 in Lecture Notes In Computer Science. Springer-Verlag, 1990.
10. Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
11. M. Kearns and S. Singh. Finite-sample convergence rates for q-learning and indirect algorithms. In *Proc. of the 1998 conference on Advances in neural information processing systems II*, pages 996–1002, Cambridge, MA, USA, 1999. MIT Press.
12. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
13. R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.
14. J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.
15. C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Univ. of Cambridge, 1989.
16. C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.