
Testing Stochastic Processes through Reinforcement Learning

François Laviolette
Département IFT-GLO
Université Laval Québec, Canada
francois.laviolette@ift.ulaval.ca

Sami Zhioua
Département IFT-GLO
Université Laval Québec, Canada
sami.zhioua@ift.ulaval.ca

Abstract

We propose a new approach to verification of probabilistic processes for which the model may not be available. We show how to use a technique from Reinforcement Learning to approximate how far apart two processes are by solving a Markov Decision Process. The key idea of the approach is to define the MDP out of the processes to be tested, in such a way that the optimal value is interpreted as a divergence between the processes. This divergence can therefore be estimated by Reinforcement Learning methods; moreover, if the two systems are not equivalent, the algorithm returns the test(s) witnessing the non-equivalence. We show how the approach can be adapted to (1) several equivalence notions (trace, ready, etc.) but more importantly to (2) other stochastic formalisms, in particular to MDPs themselves.

1 Introduction

In program verification, the goal is typically to check automatically whether a system (program, physical device, protocol, etc.) conforms to its pre-established specification. For non-probabilistic systems, one usually expects equivalence between the two, and most of the time this equivalence is chosen to be bisimulation. In the verification of probabilistic systems the comparison between the program and the specification should not be based on equivalences [7]: one reason is that the probabilities involved often come from *approximations* of the actual numbers. Hence a slight difference in the probabilities between two processes should not necessarily be interpreted as non equivalence. Instead, one is interested in a notion of distance or divergence¹ to quantify *how far apart* the processes are. When defining a distance, we have two focus: its computability, of course, but also the relation induced by zero distance. The actual value of the distance is usually not relevant but the derived relation, for example bisimulation or trace equivalence, is a guide to evaluate the power or adequacy of the distance.

In real scenarios, the model of the implementation is rarely known and the available information can only be gathered by interacting with the system. Consequently, verification in this setting has to be based on some form of sampling (or testing). In their famous paper on probabilistic transition systems [12], Larsen and Skou defined a test language that corresponds to *probabilistic bisimulation*: two processes are bisimilar if and only if they accept the same tests with the same probabilities. From the maximal difference over the probabilities on these tests, Van Breugel et. al. [2] have defined a divergence (in fact a pseudo-metric) between processes. However the fact that this divergence is based on bisimulation, a strong notion of equivalence, makes it hard to compute. In [6], we introduced K -moment equivalence, and we showed how to compute a divergence whose zero value is K -moment equivalence. Key properties of this new equivalence is that (1) it stands strictly between bisimulation and trace and (2) it is testable. The divergence is, as for Van Breugel et al.'s pseudo-metric, the maximal difference over the probabilities on tests.

¹A divergence is a distance that may not satisfy the triangle inequality and symmetry.

To compute divergences between processes that are based on the maximal difference between probabilities on test, we have chosen to use a Reinforcement Learning (RL) method. RL methods are applicable even when the model is not available. While verification techniques can deal with processes of about 10^{12} states, RL algorithms do a lot better; for example, the *TD-Gammon* program deals with more than 10^{40} possible states [14]. The key idea of our approach is to define a Markov Decision Process (MDP) out of the processes to be tested and to interpret its optimal value as a divergence between the processes. The equivalence (trace, K -moment, etc.) and its associated family of tests determine how the MDP should be constructed. Moreover, our algorithm outputs a test that witnesses the computed divergence. In [5], we showed how it can be done for trace-equivalence and we gave PAC (Probably Approximately Correct) guarantees. In [6], we extended the results to K -moment equivalence. In this paper, we expose the approach, we show how it could also be used to compute divergences between MDPs; we also discuss in more details the concept of prediction.

The plan of the paper is as follows. In the following section, we briefly define LMPs, the stochastic processes that we first worked on, and the testing language for trace equivalence. In Section 3, we informally expose our approach via a one player stochastic game, discuss the concept of prediction, and present briefly experimental results. Section 4 shows how the approach can be applied to other equivalence notions while Section 5 describes how the ideas can be extended to MDPs themselves.

2 Labelled Markov Processes and Trace Equivalence

In *reactive* systems, actions are meant to be synchronized through interaction with the environment and we consider that no internal actions occur. Our models are *Labelled Markov Processes* (LMPs) [1]; while they can be uncountable in general, we restrict to countable ones. Finite LMPs are also called Probabilistic labelled transition systems or Markov decision processes without rewards.

Definition 2.1 *A countable LMP is a tuple (S, i, Act, P) where S is a countable set of processes, $i \in S$ the initial process, Act a finite set of actions, and $P(s, a)$ a sub-probability distribution on S , for $s \in S$ and $a \in Act$. We use the notation $P_{sX}(a)$ for $P(s, a)(X)$, the probability that an a -transition from s ends in X . Given a trace τ (i.e., a sequence of actions), $P^S(\tau)$ is the probability to accept τ from the initial state i , whereas $P_{is}(\tau)$ is the probability to reach s with τ from i . Two processes are probabilistic trace-equivalent (we will simply say trace-equivalent) if they accept the same sequences of actions with the same probabilities.*

We will always assume our models to be tree like; up to bisimulation [1], it is always possible. For example, the process “Spec” in Figure 1 is an LMP and it accepts the set of traces: $\{\varepsilon, a, aa, aaa, aac, c, cc\}$ and $P^{Spec}(aac) = \frac{1}{6}$.

Trace equivalence can be characterized by a testing scenario. The test language has the following syntax :

$$\mathcal{T}_{Trace} : \quad t ::= \omega \mid a.t$$

ω is a dummy test that always terminates with success; test $a.t$ consists in executing action a and, in case of success, proceeding with test t . The execution of a test may result in several possible observations. Let a^\checkmark represents the success of action a and a^\times its failure. The observation set of test t is recursively defined as follows :

$$O_\omega = \{\omega\}, \quad O_{a.t} = \{a^\times\} \cup \{a^\checkmark.e \mid e \in O_t\}$$

To each test t is associated a probability distribution $Q_t^s(e)$ on O_t ; it represents the probability to witness observation e after running t on process s :

$$Q_\omega^s(\omega) = 1, \quad Q_{a.t}^s(a^\checkmark.e) = \sum_{s' \in S} P_{ss'}(a) Q_t^{s'}(e) \quad \text{where } e \in O_t, \quad Q_{a.t}^s(a^\times) = 1 - P_{sS}(a).$$

Theorem 2.2 ([12]) *Two processes are trace-equivalent iff they yield the same probability distribution on observations for any test of the grammar \mathcal{T}_{Trace} .*

3 Testing without the model

The objective is to define a divergence between “Spec”, a model of the specification and “Impl”, a real system; the model of the latter is not necessarily available but we assume that it has the

same set of possible action as “Spec” and that we can interact with it (as a black-box) via a testing machine². We also want this divergence to come as the solution of an MDP, the basic ingredient of RL techniques, on which the learning algorithm works. The rewards in the MDP have to be chosen carefully to make sure that the optimal value will indeed define a divergence as we expect: this is the subject of Section 3.1. We expose our approach in the form a one-player stochastic game, the player being the personification of the learning algorithm. This particular game corresponds to trace equivalence but the ideas can be adapted for other equivalence notions (see Section 4) and for other structures such as MDP (see Section 5).

3.1 Trace equivalence through a Stochastic Game

When interacting with processes “Spec” and “Impl”, the player’s goal is to detect differences between the two. Hence the game (and its corresponding MDP) should give him low reward when they *behave the same* (i.e., both succeed or both fail) and high reward otherwise. However, processes are probabilistic and hence a process may behave differently on different trials of the same action, which could lead the player to find a big difference between identical processes. This will happen more likely when the choice at a state is “wide”, more technically, when the entropy is high. To compensate this uncertainty, we introduce a third process, called “Clone” which is simply a copy of the specification but given in the form of a black-box (exactly as “Impl”) (see Figure 1). The player will get a high reward if “Impl” and “Spec” differ for some action, but this reward could be cancelled if “Spec” and “Clone” also do. Recall that the player does not see the states reached in “Impl” and “Clone” but does see what happens in “Spec”.

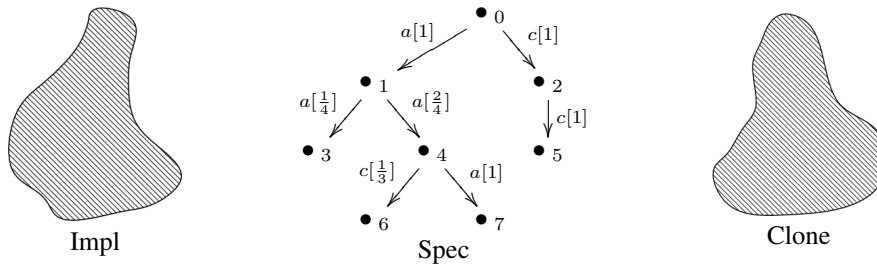


Figure 1: Implementation, Specification, and Clone

Game_{Trace}: The three processes start in their initial states; then

- Step 1**: The player chooses an action a and makes a prediction Pred on its success or failure on “Spec”. We will denote his choice by a^\checkmark for success and by a^\times otherwise.
- Step 2**: a is run on “Impl”, “Spec” and “Clone”. Let $(o_I, o_{Sp}, o_C) \in \{a^\times, a^\checkmark\}^3$ be the outcome of this experiment.
- 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 reward according to the following formula :

$$R := (o_{Sp} = \text{Pred})((o_I \neq o_{Sp}) - (o_{Sp} \neq o_C))$$

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

For example, if a^\checkmark is selected and the observation is $(a^\times, a^\checkmark, a^\checkmark)$ (i.e., Failure in “Impl”, Successes in “Spec” and “Clone”) we obtain a reward of $(a^\checkmark = a^\checkmark)((a^\times \neq a^\checkmark) - (a^\checkmark \neq a^\checkmark)) = 1(1 - 0) = 1$, but for $a^\times a^\times a^\checkmark$, we obtain $0(0 - 1) = 0$.

With the rewards so defined, we will show in Section 3.3.2 that “Spec” and “Impl” are trace-equivalent if, and only if, the optimal strategy has expected reward zero. This will imply that the optimal value of the MDP defined from this game yields has a key property for a suitable notion of divergence.

²In the classification of Van Glabbeek [8], the testing machine we assume is equipped with (1) a series of buttons (one for each action), (2) a reset button and (3) a replication button to generate copies of the current process. To avoid recursive replication, we delete copies in memory once a transition happens from one state to the next.

Remark 3.1 *This game has been inspired by a well known and well studied divergence, the Kullback-Leibler divergence. Our approach exploits the idea that two processes are “equivalent” via testing if, and only if, they yield the same probability distributions on observations for any test generated from the given test grammar. Hence, the divergence between two processes could have been defined with the help of a divergence between the probability distributions on test observations. The Kullback-Leibler divergence (KL divergence) would have been a candidate: it is defined, for two distributions Q and P , as*

$$\text{KL}(Q\|P) := \mathbb{E}_{h \sim Q} \ln \frac{1}{P(h)} - \mathbb{E}_{h \sim Q} \ln \frac{1}{Q(h)}$$

(See [3]). Unfortunately, because of the high number of possible tests (on huge systems), the maximum value over all Kullback-Leibler divergences is not tractable. Nevertheless, let us describe the analogy between **Game**_{Trace} and KL divergence. The entropy of P relativised by Q ($\mathbb{E}_{h \sim Q} \ln \frac{1}{P(h)}$ in the above formula) can be seen as how likely we can obtain different observations when interacting (via some test t) with “Spec” and “Impl”. On the other hand, the entropy of the distribution Q ($\mathbb{E}_{h \sim Q} \ln \frac{1}{Q(h)}$ in the above formula) can also be seen as a quantification over the likelihood to obtain different observations when running the same action on “Spec” twice. Thus, the game expresses the same kind of tradeoff as the KL divergence. This is an indication that one can derive the notion of divergence we are looking for.

3.2 Prediction

It is not clear at first sight why the prediction is important in Step 1 of **Game**_{Trace}. One could suggest to just run a on the three processes and collect the rewards (without multiplying by $o_{Sp} = \text{Pred}$). If “Spec” and “Impl” are trace equivalent, the optimal strategy would indeed have expected reward zero, as wanted. However, the converse would not be true: there are non trace-equivalent LMPs for which the optimal strategy would have expected reward zero. Here is an example: consider systems with one a -transition from one state to another. In “Spec” (and “Clone”), let the probability of this transition be $\frac{1}{2}$ and let it be 1 in “Impl”. Then “Spec” and “Impl” would get a divergence zero even if they are not trace-equivalent.

We proved that such examples always exhibit a specific form of symmetry in “Spec”, that is, the probabilities of the two observations (success and failure) are equal (here, both $\frac{1}{2}$).

This phenomenon (symmetry) is not specific to trace equivalence and can be observed in different other settings (e.g. other equivalence notions, etc.). As shown in the case of trace equivalence, dealing with these symmetric cases is possible through prediction and the number of different prediction values required depends on the number of observations. For instance, in trace equivalence, after an action execution, there are two possible observations on “Spec”: success (a^\checkmark) or failure (a^\times). Therefore, there are two possible prediction values: $\checkmark \times$.

At a first glance, using predictions appears to be expensive since the set of actions is multiplied for each prediction value. It is important to note, however, that the different variants a^{pred1} , a^{pred2} , etc. of the same original action a differs only by their immediate reward: a^{pred1} and a^{pred2} share the same expected values beyond the next state. This contributes to keep the size and complexity of the MDP tractable even if the set of actions is multiplied.

3.3 The Reinforcement Learning framework

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.

3.3.1 Brief recall on MDPs

In an MDP, an agent interacts with a stochastic environment at a discrete, low-level time scale. On each time step t , 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 signal r_{t+1} , which has expected value R_s^a . 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 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.

For a detailed introduction on Markov Decision Process and Reinforcement Learning algorithms, the reader may consult [14].

3.3.2 Constructing the Markov Decision Process \mathcal{M}_{LMP}

The state space of the MDP \mathcal{M}_{LMP} will be the state space of the LMP “Spec” plus one extra state, called *Dead*. This state reflects in the MDP the fact that Game_{Trace} is over: i.e. one of the three LMPs failed to execute the last action chosen by the player. Any other state of \mathcal{M}_{LMP} represents the current state of the LMP “Spec” during the execution of Game_{Trace} . For each action a of the LMP “Spec” (and hence of the LMP “Impl”), both a^\checkmark and a^\times are actions of \mathcal{M}_{LMP} . The probability transitions and the average rewards signals then follow from the rules of Game_{Trace} . Hence, if one only wants to run a Q-learning algorithm [15] on it, only the model of “Spec” and a possibility of interacting with “Clone” and “Impl” is required.

The defined MDP satisfies the following theorem.

Theorem 3.2 *Let \mathcal{M}_{LMP} be the MDP induced by “Spec”, “Impl”, and “Clone”. If $\gamma < 1$ (discount factor) or $|\mathcal{M}_{LMP}| < \infty$ then the optimal value $V^*(i) \geq 0$ for any policy π , and $V^*(i) = 0$ if and only if “Spec” and “Impl” are trace equivalent.*

We can now give the definition of trace equivalence divergence.

Definition 3.3 Let “Spec” and “Impl” be two LMPs and \mathcal{M}_{LMP} their induced MDP. We define their *trace equivalence divergence* as

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

The proofs and definition intuitions are omitted in this paper due to space limitations but can be found in [5].

3.4 Implementation and PAC guarantees

As mentioned in Section 3.1, the full model of the MDP might not be available. Therefore, it is not appropriate to use a Dynamic Programming algorithm such as value iteration [14] 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 $V^*(i)$.

To validate the accuracy of the solution given by the Q-learning algorithm (and then the accuracy of our divergence function), note that there exists a PAC (probably approximately correct) guarantee for the Q-learning algorithm (see [11]). Unfortunately, this guarantee is very difficult to compute, which makes it unusable in practice. However, in the situation where “Spec” and “Impl” are not trace-equivalent, some guarantees about the accuracy of the solution can be achieved. Indeed, from the optimal policy, one can deduce a test that witnesses this non equivalence. Hence, in non trace-equivalence situation, one can guarantee a bottom bound for the optimal value by running this deduced test on both “Spec” and “Impl”. See [5] for a detailed discussion about the PAC guarantee of our approach.

The approach described so far has been implemented for trace equivalence. 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 :

$\tau = \frac{k}{\text{currentEpisode}+l}$ (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.

4 Other equivalence notions

The tests defined by Larsen and Skou [12] have a copy construct that represents running many tests on a given state, and this recursively. The need to maintain an arbitrary number of replicas of states is an obstacle to automatization and has been an argument against bisimulation which is thus considered too strong, even for non-probabilistic processes. The algorithm can be tailored to any equivalence notion that does not require to maintain an *unbounded* number of replicas. Several equivalence notions fall in this category. Some of them are known, namely, Ready, Failure [9], Barb Acceptance, and Barb Failure [13] and some others are less known, as the K-moment equivalences family which we introduced in [6]. Due to space limitations, we only present the test grammars we propose for these notions.

Equivalence	Test Grammar
<i>Trace</i>	$\mathcal{T}_{Trace} ::= \omega \mid a.t$
<i>Ready</i>	$\mathcal{T}_{ready} ::= \omega \mid a.t \mid \{a_1, \dots, a_n\}$
<i>Failure</i>	$\mathcal{T}_{failure} ::= \omega \mid a.t \mid \{\neg a_1, \dots, \neg a_n\}$
<i>Barb Acceptance</i>	$\mathcal{T}_{BarbAcc} ::= \omega \mid a.t \mid \{a_1, \dots, a_n\}a.t$
<i>Barb Failure</i>	$\mathcal{T}_{BarbRef} ::= \omega \mid a.t \mid \{\neg a_1, \dots, \neg a_n\}a.t$
<i>K – moment</i>	$\mathcal{T}_{kmoment} ::= \omega \mid a^k.t$

A test of the form $\{a_1, \dots, a_n\}$ consists in executing the actions a_1, \dots, a_n respectively on n copies of the current process and stop. The test $\{a_1, \dots, a_n\}a.t$ consists in executing test $\{a_1, \dots, a_n, a\}$ on $n + 1$ copies of the current process, and if the a -copy succeeds, proceed with t on this copy (and delete the others). The test $a^k.t$ is equivalent to test $\{a_1, \dots, a_{k-1}\}a.t$ where $a_i = a$ for all i .

5 Application to MDPs

In addition to its applicability on other equivalence notions, the divergence algorithm presented so far can be extended to other formalisms similar to LMPs, in particular to MDPs. Recall that MDPs have a mathematical structure which is very similar to LMPs; the only important difference lies in the concept of reward which is completely absent in the LMPs. With the use of prediction, we will lift our technique from LMPs to MDPs.

For the rest of the section, by analogy, we will denote by “Spec” and “Impl”, the two MDPs on which we want to define a divergence, and as for the LMP case, we will suppose that they are in a tree like representation, and share the same set *Act* of possible actions. Moreover, we will also suppose that they share the same set *Reward* of possible reward signal values. In order to apply our technique to MDPs, we distinguish two cases.

5.1 Case 1 : the set *Reward* is small

In an MDP, when an agent chooses an action and transits to a next state, it receives a reward value out of several possible reward values. This is very similar to the LMPs situation where an action execution yields two possible observations, a^\vee and a^\times . In fact, an LMP with an additional *Dead* state can be seen as an MDP with a binary reward space (0 or 1). In this associated MDP, the agent makes a transition to the *Dead* state if the action failed in the LMP and transits as in the LMP otherwise. Moreover, the associated MDP’s reward signals are all equal to 1 except when the *Dead* state is reached, in which case the reward is 0. From this point of view, it is easy to see that our approach generalizes naturally to MDPs, simply by adapting our testing framework in order to capture the fact that an action execution yields several possible observations (one for each possible reward signal). Then, as discussed in Section 3.2, it is possible to adapt the divergence algorithm by defining a prediction value for each possible reward.

More precisely, the construction of the MDP \mathcal{M}_{MDP} from the MDPs “Spec”, “Impl” (and “Clone”) on which our divergence notion is based will be identical to the one given in Section 4, except for

the set of actions which in this case will be

$$\left\{ a^r \mid a \in Act \text{ and } r \in Reward \right\}.$$

This will lead to an algorithm for which the theorems of Section 3.3.2 hold as well. Note however that the set of actions will be multiplied by the number of rewards. But, as mentioned in Section 3.2, the different variants of each action will differ only by their immediate reward. If the number of different possible reward signals of the MDP is relatively small, the problem should remain tractable.

5.2 Case 2 : the set *Reward* is very large or continuous

In this situation, the previous approach is not appropriate since the MDP \mathcal{M}_{MDP} will then yield a huge or infinite number of actions. Nevertheless, we will see that this is not really a problem because every MDP is equivalent to an MDP whose reward space has exactly two elements.

First note that, by way of rescaling, we can assume w.l.o.g. that *Reward* is contained in the unit interval $[0, 1]$. Now, let us define “Spec_binary” the MDP obtained from “Spec” by replacing its reward signals $R_{s s'}^a$ by the *stochastic* binary reward signals $Reward_{s s'}^a$ defined as follows:

$$\begin{aligned} \Pr (Reward_{s s'}^a = 0) &= 1 - R_{s s'}^a, \\ \Pr (Reward_{s s'}^a = 1) &= R_{s s'}^a. \end{aligned}$$

Clearly, “Spec” and “Spec_binary” have identical expected immediate rewards. Since the reward part of an MDP is only based on the *expected* immediate rewards ($R_{s s'}^a$), two MDPs that differ only by their reward signals (without differing by their expected immediate rewards) are, as MDPs, equivalent. Thus “Spec” and “Spec_binary” are equivalent.

Hence, one can define our divergence notion to arbitrary MDPs by applying the construction of Case 1 to “Spec_binary” and “Impl_binary”.

6 Conclusion

This paper is a continuation of [5] and [6] where (1) a completely new approach to estimate how far apart two LMPs are and (2) a new family of equivalences (*K*-moment) that are a good compromise between trace-equivalence and bisimulation were presented. Indeed, we introduced a notion of 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 RL approach therefore opens a way for analyzing huge systems and even infinite ones. Moreover, it can be adapted to other equivalences that can be tested via recursive replication-free test grammars.

In this paper, we showed how the approach can be extended to compute divergences between MDPs. This opens a way to a theory of approximations on MDPs as it happened for LMPs (for example, see [4]). Indeed, it is surely a good strategy to solve an approximation of a very large MDP rather than trying to solve the MDP itself, provided some guarantees on the fact that an optimal solution on the approximation remains a good solution on the original MDP.

Finally, both for LMPs and MDPs, if two processes are trace-equivalent, our divergence will return zero, otherwise it will provide a number and a test that witness the non equivalence.

References

- [1] 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*.
- [2] F. Breugel, S. Shalit, and J. Worrell. Testing labelled Markov processes. In *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 537–548. Springer, 2002.
- [3] T. M. Cover and J. A. Thomas. *Elements of Information Theory*, chapter 12. Wiley, 1991.
- [4] Vincent Danos and JosÈe Desharnais. Labeled Markov Processes: Stronger and faster approximations. In *Proceedings of the 18th Symposium on Logic in Computer Science*, Ottawa, 2003. IEEE.

- [5] J. Desharnais, F. Laviolette, K. Darsini Moturu, and S. Zhioua. Trace equivalence characterization through reinforcement learning. In *Lectuer Notes in Artificial Intelligence*, volume 4013, pages 371–382, 2006. Canadian Conference on Artificial Intelligence.
- [6] J. Desharnais, F. Laviolette, and S. Zhioua. Testing probabilistic equivalence through reinforcement learning. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS 2006, 2006.
- [7] A. Giacalone, C. Jou, and S. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods*, IFIP TC2, 1990.
- [8] R.J. Van Glabbeek. The linear time - branching time spectrum ii. In *CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory*, pages 66–81, London, UK, 1993. Springer-Verlag.
- [9] C.-C. Jou and S. A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In J. Baeten and J. Klop, editors, *CONCUR 90 1st Int. Conf. on Concurrency Theory*, number 458 in LNCS. Springer-Verlag, 1990.
- [10] L. P. Kaelbling, M. L. Littman, and A. 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] G. Lowe. Representing Nondeterministic and Probabilistic Behaviour in Reactive Processes. Technical report, Progr. Res. Group, Oxford University, 1993.
- [14] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.
- [15] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Univ. of Cambridge, 1989.