SAMI ZHIOUA

STOCHASTIC SYSTEMS DIVERGENCE THROUGH REINFORCEMENT LEARNING

Thèse présentée
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de doctorat en informatique
pour l'obtention du grade de Philosophiæ Doctor (Ph.D.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2008

Résumé

Les mathématiques offrent un cadre convenable pour raisonner rigoureusement sur les systèmes et phénomènes réels. Par exemple, en génie logiciel, les méthodes formelles sont parmi les outils les plus efficaces pour détecter les anomalies dans les logiciels. Plusieurs systèmes réels sont stochastiques par nature dans le sens où leur comportement est sujet à un aspect d'incertitude. La représentation de ce genre de systèmes requiert des modèles stochastiques comme les processus de Markov étiquetés (LMP), les processus de Markov décisionnels (MDP), etc. Cette thèse porte sur la quantification de la différence entre les systèmes stochastiques. Les contributions majeures sont :

1. une nouvelle approche pour quantifier la divergence entre les systèmes stochastiques basée sur l'apprentissage par renforcement,
2. une nouvelle famille de notions d'équivalence qui se situe entre l'équivalence par trace et la bisimulation, et
3. un cadre plus flexible pour la définition des notions d'équivalence qui se base sur les tests.

Le résultat principal de la thèse est que l'apprentissage par renforcement, qui est une branche de l'intelligence artificielle particulièrement efficace en présence d'incertitude, peut être utilisé pour quantifier efficacement cette divergence. L'idée clé est de définir un MDP à partir des systèmes à comparer de telle sorte que la valeur optimale de cet MDP corresponde à la divergence entre eux. La caractéristique la plus attrayante de l'approche proposée est qu'elle est complètement indépendante des structures internes des systèmes à comparer. Pour cette raison, l'approche peut être appliquée à différents types de systèmes stochastiques. La deuxième contribution est une nouvelle famille de notions d'équivalence, que nous appelons K-*moment*, qui est plus forte que l'équivalence par trace mais plus faible que la bisimulation. Cette famille se définit naturellement à travers la coïncidence de moments de variable aléatoires (d'où son nom) et possède une caractérisation simple en terme de tests. Nous montrons que K-*moment* fait partie d'un cadre plus grand, appelé test-observation-equivalence (TOE), qui constitue la troisième contribution de cette thèse. Il s'agit d'un cadre plus flexible pour la définition des notions d'équivalence basé sur les tests.

Abstract

Modelling real-life systems and phenomena using mathematical based formalisms is ubiquitous in science and engineering. The reason is that mathematics offer a suitable framework to carry out formal and rigorous analysis of these systems. For instance, in software engineering, formal methods are among the most efficient tools to identify flaws in software. The behavior of many real-life systems is inherently stochastic which requires stochastic models such as labelled Markov processes (LMPs), Markov decision processes (MDPs), predictive state representations (PSRs), etc. This thesis is about quantifying the difference between stochastic systems. The main contributions are:

1. a new approach to quantify the divergence between pairs of stochastic systems based on reinforcement learning,

2. a new family of equivalence notions which lies between trace equivalence and bisimulation, and

3. a refined testing framework to define equivalence notions.

The important point of the thesis is that reinforcement learning (RL), a branch of artificial intelligence particularly efficient in presence of uncertainty, can be used to quantify efficiently the divergence between stochastic systems. The key idea is to define an MDP out of the systems to be compared and then to interpret the optimal value of the MDP as the divergence between them. The most appealing feature of the proposed approach is that it does not rely on the knowledge of the internal structure of the systems. Only a possibility of interacting with them is required. Because of this, the approach can be extended to different types of stochastic systems. The second contribution is a new family of equivalence notions, *K-moment*, that constitute a good compromise between trace equivalence (too weak) and bisimulation (too strong). This family has a natural definition using coincidence of moments of random variables but more importantly, it has a simple testing characterization. *K-moment* turns out to be part of a bigger framework called test-observation-equivalence (TOE), which we propose as a third contribution of this thesis. It is a refined testing framework to define equivalence notions with more flexibility.

Avant-propos

Je ne peux pas déposer ma thèse sans faire les remerciements à ceux qui les méritent. Tout d'abord rien ne serait possible sans la chance que Dieu m'a accordé pour arriver à ce niveau. Une chance que plusieurs sur cette terre n'ont pas.

Je remercie mes parents qui m'ont supporté durant les 20 premières années de ma vie avant de me laisser voler de mes propres ailes. Les sacrifices qu'ils ont faits pour moi et pour mes sœurs sont loin d'être décris en deux lignes. Cette thèse est leur réussite avant qu'elle soit la mienne.

Je remercie ma chère femme Karima qui a sacrifié beaucoup pour cette thèse, en commençant par sa carrière professionnelle. Son soutien était inestimable dans les périodes les plus difficiles de cette thèse.

Mes deux directeurs de recherche Josée Desharnais et François Laviolette étaient des vrais parents académiques pour moi. Ils ont joué leurs rôles d'encadreurs d'une façon merveilleuse. Je les remercie pour leur disponibilité qu'on retrouve rarement chez les directeurs de recherche. Je les remercie pour les longs après-midi que nous avons passés à discuter des problèmes de symétrie et de convergence ! Je les remercie pour toutes les nuits blanches que nous avons passées ensemble à travailler sur des articles à soumettre. Je les remercie pour m'avoir transmis une expertise inestimable dans la démonstration des théorèmes, dans la rédaction des articles et dans la recherche scientifique en général. Josée, c'est le sérieux, la discipline mais surtout la gentillesse. François, c'est l'ouverture, l'esprit critique, mais aussi les bonnes manières. Ensemble ils forment un couple extraordinaire, dans le travail mais aussi dans la vie, avec qui c'est tellement agréable de travailler.

Mes remerciements vont aussi à mes anciens directeurs de recherche Nadia Tawbi et Mourad Debbabi. C'est avec eux que j'ai fait mes premiers pas dans la recherche. Nadia Tawbi m'a repérée et m'a aidé dans les premières semaines de mon arrivé à l'Université Laval. Elle m'a soutenu personnellement, académiquement et financièrement. Avec

elle j'ai fait une maîtrise dont je suis très fier. Mourad Debbabi est un chercheur exceptionnel duquel j'ai appris énormément de choses. C'est un professeur qui m'a marqué à jamais et sans qui je ne serais peut être pas arrivé à ce niveau. À tous deux je suis très reconnaissant.

Je tiens à remercier mes amis et mes collègues avec qui j'ai vécu au département pendant plus de 6 années : Mahjoub Langar qui était un véritable frère pour moi, Hamdi Yahyaoui, Lamia Ketari, Chamseddine Talhi, Abdelouahed Gherbi, Eric Lacoursière et ma coéquipière pendant plus qu'une année dans ma thèse : Krishna Priya Moturu. Je remercie aussi les professeurs du département d'informatique qui m'ont aidé à certaines étapes de la thèse, en particulier Mohamed Mejri et Béchir Ktari ainsi que mes autres collègues : Mohamed Mbarki, Mohamed Barkaoui, Hatem Mahbouli, Marouène Ben Jabeur, Claude Bolduc, Raphael Khoury, Amir Pirzadeh, Sara Shanian, Alexandre Cormier et Daniel Godbout.

Je voudrais adresser un remerciement sincère à tout le personnel du département d'informatique pour la qualité de leur service, et plus précisément notre Lynda Goulet nationale !

Enfin je remercie les organismes qui m'ont supporté financièrement je cite, le gouvernement de Tunisie, le FQRNT, le CRSNG et la fondation de l'Université Laval.

Montréal, février 2008
Sami Zhioua

Cette thèse a été soutenue avec succès le jeudi 31 janvier 2008 à la salle du conseil scientifique de l'Université Laval. Le jury était composé de :

Président
Mario Marchand, Ph.D. (Université Laval)

Examinateurs
Josée Desharnais, Ph.D., directrice de thèse (Université Laval)
Danny Dubé, Ph.D. (Université Laval)
Marta Kwiatkowska, Ph.D. (Oxford University)
François Laviolette, Ph.D., co-directeur (Université Laval)
Doina Precup, Ph.D. (McGill University)

Contents

Chapter 1

Introduction

> *January* 1987*,*
> *Yakima, Washington, USA.*
> *A cancer patient went to the hospital to receive his habitual treatment through Therac-*25*, a system used for the treatment of cancer by administering regularly a dose of radiation to a specific part of the body in order to hopefully kill the malignant tumor. When the treatment was underway, an error paused the machine, which indicated that the dose was not administered yet. Unfortunately, the technician repeated the treatment. An overdose was administered and the man died three months later. It was the sixth deadly incident due to an undetected software bug in the Therac-*25 *system. Therac-*25 *is considered one of the most devastating computer related engineering disasters to date [32].*

Nowadays, software and hardware systems are controlling our lives. They are controlling the temperature inside our fridges, the toys of our children, and the remote control of our TVs. They are needed in supermarkets to prepare our bills, in banks to transfer our salaries, and in hospitals to monitor our heart pace if we are sick. Thanks to them, we are exploring space, we are finding new cures for diseases, and we are aware of everything that happens in the whole world. The more dependant we are on these systems, the more important is the need for them to function correctly. However, all such systems are designed and developed by human beings and hence they can include errors and faults that prevent them form behaving as intended. These errors can have a variety of effects with different levels of inconvenience to the user. Some have only a negligible effect on the system functionality and may lie undetected for a long time and some have extremely serious consequences such as the one mentioned above. The presence of such systems in safety-critical applications coupled with their increasing

complexity makes the verification task more challenging.

The most rigorous tool to detect flaws is *formal verification*. Formal verification is the process of proving or disproving the correctness of a system with respect to a certain *formal specification* which represents the desired system behavior. Typically, this process consists in representing the real system and its specification using an abstract mathematical formalism and then comparing them in some form or another. Examples of popular mathematical formalisms used to model systems are timed automata, process algebra, transition systems, etc. In this thesis, we use the transition systems approach to model systems. A transition system describes an application as a set of states and transitions. A state represents a picture of the system at some moment while the transitions represent state changes in the application. For example, a coffee machine can very well be modelled through a set of states and transitions. Before any interaction with the user, the coffee machine is in an "idle" state. When the user inserts a coin, the machine makes a transition to a new state which can be called "coin inserted" or "waiting for selection". Then, the user chooses the type of coffee he wants and the machine switches to another state which can be called "preparing coffee" and so on.

As described, the coffee machine is a deterministic system since one can predict its behavior, in particular the next state, with certainty. However, the behavior of many real-life processes is inherently *stochastic*. The word stochastic is used to describe subjects that contain some element of random behavior. For a system to be stochastic, one or more of its parts should have randomness associated with them. For example, any game based on dice defines a stochastic behavior since the next state depends on the result of the throw of the dice. The models to represent such systems usually quantify such randomness using probabilities[1]. Historically, traces of gambling such as the perfectly shaped dice found in Egyptian excavations show that there has been an interest in quantifying the ideas of probability for millennia. Exact mathematical foundations, however, arose much later with Huygens (17^{th} century) and Bernouilli (18^{th} century). Since then, probability theory became a major field of mathematics and entered almost all scientific fields.

1.1 Motivation

In software engineering, probabilities are very useful to study the non-functional aspects of system behaviors such as performance and reliability. Indeed, when designing

[1] In this thesis, we use the words probabilistic, randomized and stochastic as synonyms.

a new system or software, it is important to investigate the average response time or the probability that a certain failure occurs. This will allow to produce more realistic models. More generally, performance and reliability properties are particularly important to investigate for the so-called shared resources systems (or concurrent systems) where a varying number of components compete for the same resources. Typical examples are transportation and processing systems such as telecommunication networks, manufacturing systems, distributed systems including all types of modern data processing machines. The consequences of concurrency are mutual interference, delays to contention and varying service quality during different periods of time in addition to transmission errors and resource failures. An analysis of these and similar properties requires that some form of information about the statistical behavior of the system and the probability of the occurrence of relevant events is put into the model. The concept of stochastic process allows to accurately model and investigate these properties and phenomena.

In artificial intelligence too, stochastic models are useful to represent *dynamical systems*. A dynamical system is one whose state changes over time often with a certain amount of uncertainty. Typically, in studying dynamical systems one is interested in one of two objectives: learning and controlling. Learning a dynamical system consists in estimating, more or less precisely, the parameters of the model. This will allow one to understand the inner-working of the environment and figure out the patterns behind it. A good example is how human gene patterns can be learned from raw DNA sequences. Controlling dynamical systems, on the other hand, consists in finding an optimal strategy (or policy) of behaving given a goal task. An example is an autonomous robot navigating in a room and trying to find the shortest path to the exit door.

There exist several stochastic formalisms that can be used to represent stochastic systems. The choice of the model depends on the characteristics of the system to be modelled. To mention only some of these characteristics, the state space can be discrete or continuous, the transition probabilities leaving a state may sum up to exactly 1 or less than 1, the state space can be observable or partially observable, the task can be controllable or uncontrollable, etc.

In real scenarios, very often one needs to confront systems and/or models to see whether they define the same behavior. It is a notable feature of concurrency theory that there are many different notions of process equivalence e.g. trace equivalence, bisimulation, etc. These notions differ in their distinguishing capabilities. Hence, two processes can be considered equivalent by a "weak" notion such as trace equivalence whereas a stronger notion such as bisimulation can distinguish between them. Let us consider a program verification problem where the goal is to check whether an

implementation conforms to its pre-established specification. The implementation is the final product or the constructed physical device whereas the specification should represent in some form or another the desired properties of the implementation. For non-probabilistic systems, one usually expects equivalence between the two, and most of the time this equivalence is chosen to be bisimulation. When verifying stochastic systems, however, the task is more challenging due to their quantitative nature. In such situations, it has been observed that the comparison between the program and the specification should not be based on equivalences: one reason is that probability values often come from approximations and hence a slight difference in the probabilities between two systems should not be necessarily interpreted as non equivalence. A more appropriate approach would be to focus on "how close" the systems are. This implies a notion of distance or divergence[2] between stochastic systems.

In artificial intelligence too, the relative distance between a system and its model indicates how good the latter is. A model, by nature, is an abstraction of the real system. Hence, the model may deviate more or less from the original behavior. To which level the model deviates from the original behavior is an important issue to assess the accuracy and fidelity of the model representation. In particular, one may wonder to which level an optimal policy of a model remains optimal in the real system. Furthermore, it is very common that one needs to confront two dynamical systems to see whether they define the same behavior. For example, in the presence of two robots, each one trying to find the exit door in a different room, it is interesting to figure out which robot has the simpler task.

A divergence notion between processes will assign a number to every pair of processes giving an indication of how far they are from each other. A distance of zero means that the processes are equivalent and processes that are very "close" will yield a smaller distance than processes that are far apart. The value of the divergence itself is usually not relevant; however the derived relation is of particular importance.

Analyzing processes is more complex in the stochastic case than in the deterministic one. The reason is that a stochastic process may behave differently on different trials with same input. This will happen more likely when the entropy of the stochastic process is high. Entropy is a notion from information theory [47]. Intuitively, a state of a process is said to have a high entropy if it can make transitions to several next states. The task is further complicated when the internal structure of the stochastic process is not known. This case is very common in real scenarios. For instance, a client who wants to verify a software built by a third party will proceed without any knowledge of the internal structure of the software. One way of dealing with such

[2] A divergence can be defined as a pseudo-distance.

situations is via a testing framework. In such a framework, the process is viewed as a black-box with a hidden internal structure and an interface consisting of buttons that can be pressed. A test is viewed as an algorithm for how to experiment the black-box. It specifies which button should be pressed and when. In their famous paper on probabilistic transition systems [30], Larsen and Skou have defined a test framework that corresponds to "probabilistic" bisimulation. Their framework requires the ability to take multiple copies of any state in order to experience a different test on each copy. The problem with this replication capability is that it is recursive; a test executed on one copy can in turn require to take multiple copies of next states. The need to maintain an arbitrary number of copies 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. Trace equivalence does not require any replication capability and hence defines a simpler testing framework. However, for many applications it does not discriminate enough.

1.2 Contributions

The three main contributions of this thesis are:

1. a new approach to quantify the divergence between pairs of stochastic systems based on reinforcement learning,
2. a new family of equivalence notions, called K-*moment* which lies between trace equivalence and bisimulation, and
3. a refined testing framework, called TOE, to define equivalence notions.

The main result of this thesis is an algorithm to estimate divergences between stochastic systems based on reinforcement learning (RL). RL is a branch of machine learning concerned with planning and learning in probabilistic environments. The environment is typically formulated as a Markov decision process (MDP) which is a standard formalism to describe multi-stage decision making. Any MDP has an optimal policy of behaving which is guaranteed to get the maximum cumulative reward, called also the optimal value. The objective of RL is to find this optimal policy. The field of RL has become one of the most active research areas in machine learning and artificial intelligence. It has developed strong mathematical foundations and impressive applications in robotics, control theory, simulation, etc. The choice of using RL is motivated by mainly two reasons. The first is that, while classical verification techniques can deal

with processes of about 10^{10} states [40], RL algorithms can do a lot better. For example, the TD-Gammon program [56] deals with more than 10^{20} possible states. The second is that RL techniques are particularly efficient in the absence of a complete knowledge of the problem. Recall that in several realistic scenarios, the models of the stochastic processes to be compared are unknown.

The key idea of our approach is to define a Markov decision process out of the processes to be tested and to interpret the optimal value of this MDP as a divergence between the processes. This optimal value and the corresponding policy can then be estimated by any RL algorithm. In a very simple scenario, the approach can be used to estimate the divergence between stochastic systems of the same type. However, since no assumptions are made about the models of these processes, the comparison can be carried out between two different types of models or systems, for instance between a labelled Markov process (LMP) and a probabilistic labelled transition system (PLTS) or between a partially observable Markov decision process (POMDP) and a predictive state representation (PSR). The algorithm does not need more than the possibility to test the stochastic processes via a testing framework.

A divergence is best defined with respect to an equivalence notion which is, in particular, induced by the zero distance. Hence, one can define trace equivalence divergence, bisimulation divergence, etc. The algorithm we propose can be tailored to any such divergences but with only one condition; the equivalence notion should come with a testing framework that does not use recursive replication. This might suggest that the proposed approach will work only for weak equivalences and any strong equivalence such as bisimulation will not fit. Here comes the second contribution of the thesis. We propose a new family of equivalences called *K-moment* which constitutes a good compromise between trace (too weak) and bisimulation (too strong). This family is called *K-moment* because it is based on moments coincidence of random variables. An interesting aspect of *K-moment* is its simple testing formulation that uses replication but not recursively. Indeed, the tester is allowed to take multiple copies of the current state but when a transition to the next state occurs, all these copies are discarded from memory.

K-moment turns out to be part of a bigger class of equivalences we call test-observation-equivalence (TOE for short), which we propose as a third contribution of this thesis. Typically, when equivalence notions are defined via a testing framework, the main element to specify is the test language from which tests are generated. However, there is another element that can play an important role, that is: how observations are grouped. Indeed, generally, only some aspects of observations are significant. Grouping of "similar" observations will allow focusing on important aspects of observations.

TOE is an improved technique to define equivalence notions by putting more emphasis on the observation function which indicates how observations are grouped. All TOEs including *K-moment* are recursive replication free and consequently they do fit in the RL based algorithm to compute the divergence between stochastic processes.

1.3 Thesis Organization

The organization of the thesis is depicted in Figure 1.1.

In the next chapter, we present the necessary background on stochastic processes from the point of view of formal verification. We recall the definitions of probabilistic transition systems, in particular, labelled Markov processes (LMPs). We make a brief survey of the probabilistic equivalence notions such as trace equivalence and bisimulation by giving their testing and logical characterizations. Finally, we present related work concerning metrics for stochastic processes.

Chapter 3 is a survey of the field of reinforcement learning. It introduces the basic notions such as MDP, policy, and value function and then goes through the best known techniques of RL including dynamic programming (DP), Monte Carlo (MC), and temporal difference learning (TD).

In Chapter 4, we present the main contribution of the thesis, that is, how to formulate the problem of estimating the divergence between stochastic processes using RL. In this chapter, the stochastic processes are assumed to be LMPs and the divergence notion we define is related to trace equivalence. The main ideas are exposed through a one-player stochastic game and we provide the formal proofs along with the experimental results.

In Chapter 5, we introduce the family of *K-moment* equivalence notions and we discuss it from different point of views, namely, moments coincidence, logical, and testing.

In Chapter 6, we present the test-observation-equivalence (TOE) framework which is composed of a generic test grammar and a generic observation function. The first part of the chapter describes in detail these two components. The second part is dedicated to showing that our proposed approach to quantify the divergence between stochastic processes (Chapter 4) can be tailored to any equivalence notion defined through the TOE framework and we provide the necessary formal proofs. Finally, we give a TOE

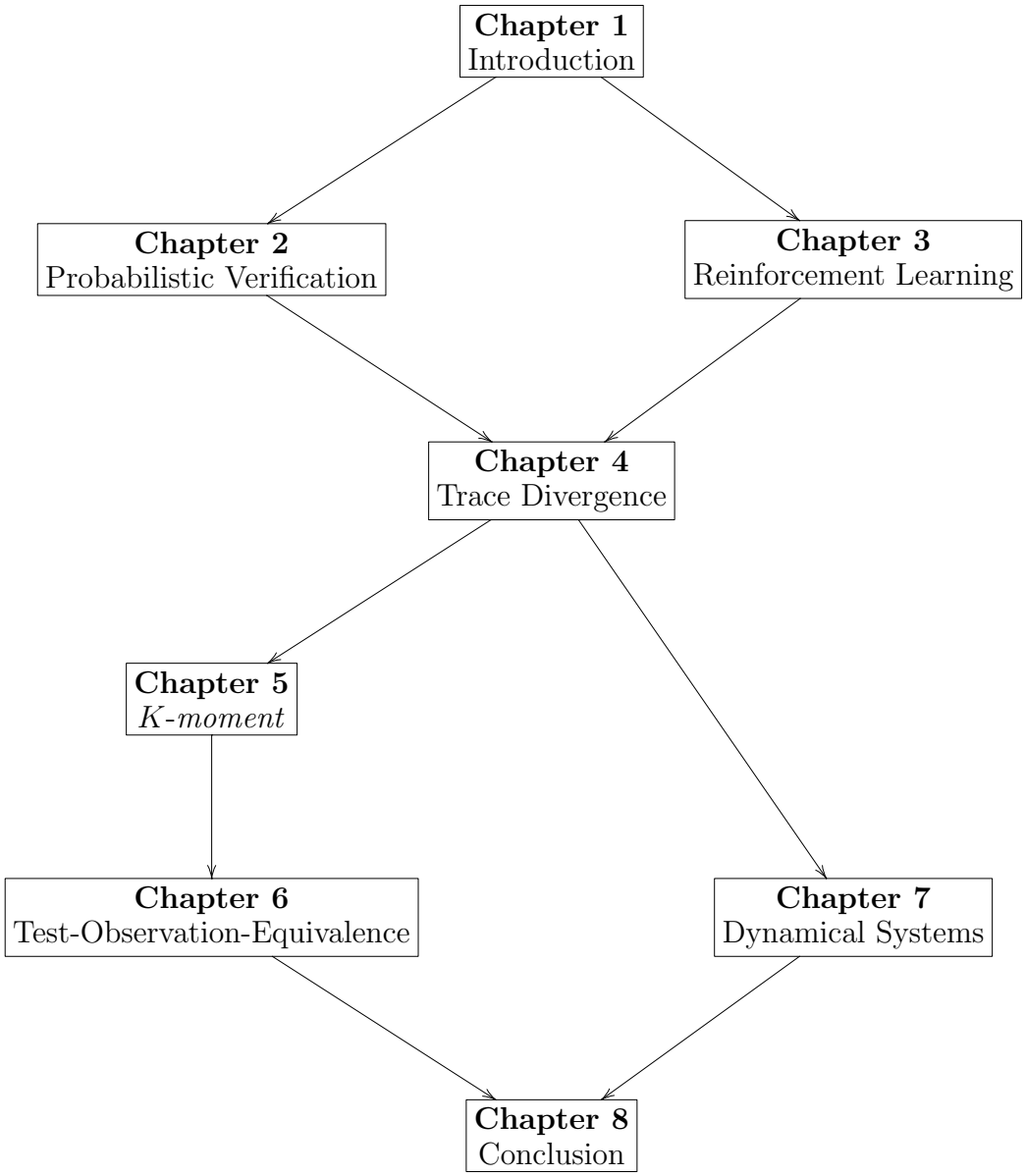


Figure 1.1: Thesis organization

definition of the equivalence notions presented in the previous chapters and in the light of this, we organize them in a hierarchy based on their distinguishing capabilities.

In Chapter 7, we make a survey of the stochastic models used in artificial intelligence such as POMDPs, HMMs, PSRs, etc. We classify them with respect to criteria such as observability and controllability. Then, we show how our proposed approach can be used to quantify the divergence between POMDPs and we indicate how it can be extended to the other stochastic models. Finally, we present the results of experimental analysis carried out on POMDP benchmarks.

Chapter 8 contains a short summary of the contributions and a discussion on future work.

In Appendix A, we present part of the process we carried out to tune the parameters of the selected RL algorithm, namely, Q-learning.

In Appendix B, we detail two use cases of HMM that can benefit from our proposed approach. The first is about automatic speech recognition and the second is about DNA sequence analysis.

Chapter 2

Background on Probabilistic Verification

2.1 Introduction

This thesis is about probabilistic processes. In this introductory chapter, we present the necessary background on probabilistic processes from the point of view of formal verification. Later, in Chapter 7, we complete this view by presenting stochastic models used in artificial intelligence. The first section of the chapter is dedicated to necessary definitions of probabilistic transition systems. The following section focuses on equivalence notions for probabilistic processes. The last section is a survey of the state of the art regarding metrics and distances for probabilistic processes.

2.2 Transition Systems

The term "transition system" originates from viewing the processes as a set of states and a set of transitions between these states. In formal verification, one can alternatively describe a process in a syntactic form using a process algebra. However, both descriptions have the same expressive power. In this thesis, we restrict ourselves to the transition systems view and we will not discuss their syntactic description.

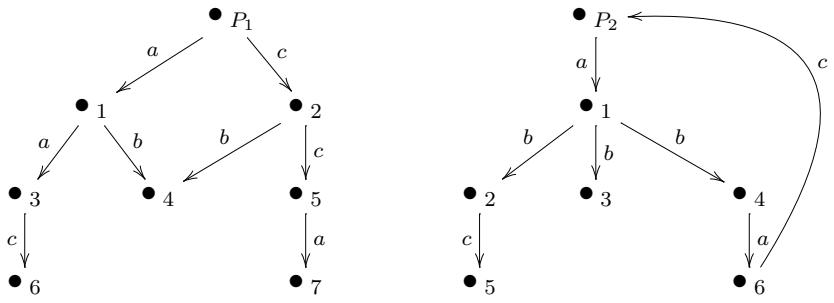


Figure 2.1: A deterministic and a non-deterministic labelled transition systems

2.2.1 Labelled Transition Systems (LTS)

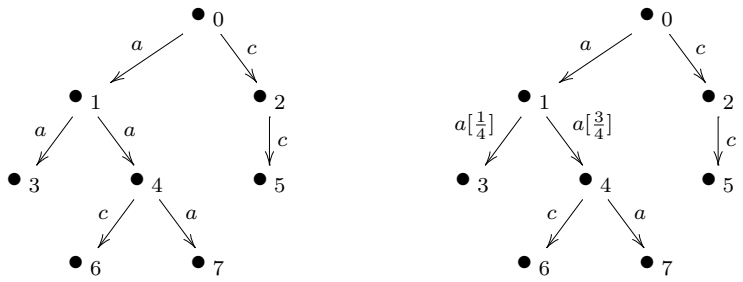
The labelled transition system (LTS) formalism represents the operational behavior of a system. From an operational standpoint, a system can be seen as a set of states (processes, configurations, etc.) and a set of transitions between these states. A transition requires the occurrence of an action (or event). Hence, an execution of a program can be seen as a sequence of states and actions, that is, a path through the LTS.

The LTS description of a process emerges from its interaction with the environment. Indeed, the process and its environment are synchronized on actions : if the process is in state s and the environment chooses an action a, which is enabled (possible) in s, then the process makes an a-labelled transition to the next state. In practice, the process can make an "internal" transition (typically called τ) which is not synchronized with the environment and hence cannot be observed by an external observer. In this thesis, we focus only on observable actions.

Definition 2.1. *An LTS is a tuple $(\mathcal{S}, s_0, \mathcal{A}, T)$ where:*

- *$\mathcal{S}$ is the set of states (processes).*
- *s_0 is the initial state.*
- *$\mathcal{A}$ is the set of actions (events or labels), and*
- *T: $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a transition relation such that (s, a, s') means that running action a on state s may yield a transition to state s'.*

An LTS can be deterministic or non-deterministic. In a deterministic LTS, the execution of an action results in a transition to only one state whereas in a non-deterministic LTS, the execution of an action may lead to one of several possible next states. P_1 in Figure 2.1 is an example of a deterministic LTS whereas P_2 is a non-deterministic one because running action b in state 1 may lead to state 2, 3, or 4.



(a) LTS (b) PLTS

Figure 2.2: Labelled transition systems

2.2.2 Probabilistic Labelled Transition Systems (PLTS)

Probabilistic labelled transition systems (PLTS) are an extension of LTS where transitions are supplied with probabilities in $[0, 1]$. Probabilities are introduced in transition systems in order to quantify non-determinism. As mentioned above, non-determinism occurs when running an action from a state can lead to two or more next states and for which no information is available as to which one of them is chosen. For example, in Figure 2.2(a) there is non-determinism at state 1. This non-determinism is quantified in Figure 2.2(b) by assigning to each transition a probability in $[0, 1]$. Notice that the edge of the graph is labelled by the action and the probability between brackets. We will often drop the probability when it is 1.

Definition 2.2. *A PLTS is a tuple $(\mathcal{S}, \mathcal{A}, P)$ where:*

- *$\mathcal{S}$ is the set of states.*
- *$\mathcal{A}$ is the set of actions.*
- *$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition probability function such that $\forall\, a \in \mathcal{A}$, $s \in \mathcal{S}, P(s, a, s')$ represents the probability to run action a from state s and end-up in state s' and*

$$\sum_{s' \in \mathcal{S}} P(s, a, s') = 0 \text{ or } 1.$$

If the transition probabilities for a given state sum up to zero ($\sum_{s' \in \mathcal{S}} P(s, a, s') = 0$) we say that the state s is terminal. Every state s in $\mathcal{S}$ determines a process having s as its initial state. Once an action a is chosen by the environment, the transition probability function P will be used to select which transition to take. Hence the choice of the action is left to the environment. This is called a reactive model. It is the model

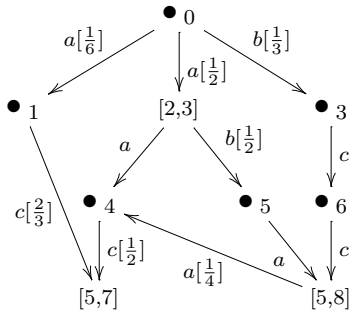


Figure 2.3: An example of a labelled Markov process

we use throughout this thesis. A description of other models (generative and stratified) can be found in [59].

2.2.3 Labelled Markov Processes (LMP)

A labelled Markov process is a PLTS where the state space can be continuous (rather than discrete) and where the probability distributions may sum up to less than 1.

By allowing the state space to be continuous, an LMP can be used to model a larger set of systems than a PLTS. Indeed, several systems, in particular physical devices, involve parameters with continuous values such as temperature, speed, distance, etc.

Also, unlike a PLTS, an LMP allows the probability distribution to sum up to less than 1. Probabilities from the same state with the same action summing up to 1 means that the execution of the action will necessarily yield a transition. A sum of 0 means that the state is terminal. When this sum is strictly between 0 and 1, it means that the action may be refused (with the remaining probability). For example, if the sum of the transition probabilities in a state s after action a is $\frac{4}{5}$, then with probability $\frac{1}{5}$, the action is not accepted. There is another interpretation for the fact that the total probability out of a state is strictly between 0 and 1. The process is considered *underspecified*; this is useful when one is interested in a notion of preorder between processes (see Chapter 2 in [13] for more details).

Definition 2.3. *A labelled Markov process with action set $\mathcal{A}$ is a tuple $(\mathcal{S}, s_0, \Sigma, \tau)$, where $\mathcal{S}$ is the set of states, which is assumed to be an analytic space, s_0 is the initial state, and Σ is a Borel σ-field on $\mathcal{S}$, and $\forall a \in \mathcal{A}$,*

$$\tau_a : \mathcal{S} \times \Sigma \longrightarrow [0, 1]$$

is a transition sub-probability function.

The transition sub-probability $\mu_a(x, C)$ represents the probability of the system, starting in state x and executing action a, of making a transition into one of the states in C. More formally, a transition sub-probability function is defined as follows:

Definition 2.4. *A transition sub-probability function on a measurable space (X, Σ) is a function $\mu : X \times \Sigma \longrightarrow [0, 1]$ such that for each fixed $x \in C$, the set function*

$$\mu(x, \cdot) : \Sigma \longrightarrow [0, 1]$$

is a sub-probability measure, and for each fixed $C \in \Sigma$ the function

$$\mu(\cdot, A) : X \longrightarrow [0, 1]$$

is a measurable function.

In this thesis we consider only countable (discrete) LMPs. Hence, we will use the following definition of LMP.

Definition 2.5. *An LMP is tuple $(\mathcal{S}, s_0, \mathcal{A}, P)$ such that:*

- *$\mathcal{S}$ is the set of states.*
- *s_0 is the initial state.*
- *$\mathcal{A}$ is the set of actions, and*
- *P: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a transition probability function such that for states s and s' and action a, $P(s, a, s')$ (noted also $P_{s \to s'}(a)$) represents the probability to run action a on state s and end up in state s' and*

$$0 \leq \sum_{s' \in \mathcal{S}} P_{s \to s'}(a) \leq 1.$$

We use the notation $P_{s \to C}(a)$ for the probability that an a-labelled transition from s ends in C ($C \subseteq \mathcal{S}$):

$$P_{s \to C}(a) = \sum_{s' \in C} P_{s \to s'}(a).$$

2.3 Equivalences and Characterizations

In several practical scenarios of formal verification, one is confronted with the problem of checking whether two processes are equivalent. For example, if one process represents

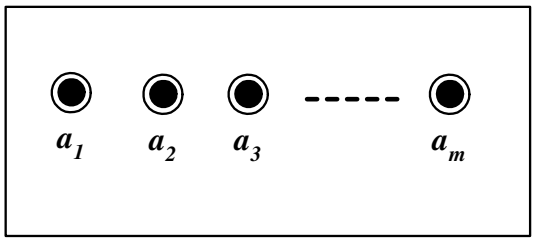


Figure 2.4: A testing machine

the specification of a system and a second process represents its implementation, then one expects equivalence between the two. A variety of equivalence notions between probabilistic processes have been proposed in the literature with different discriminating power: two processes could be equivalent according to a notion while they are non-equivalent according to a stronger notion. In this section, we present equivalence notions for probabilistic processes and we discuss their testing and logical characterization. But before, let us introduce testing and logical characterizations.

Testing a process can be viewed as interacting with a testing machine such as the one in Figure 2.4. This testing machine is a black box equipped with buttons, one for each action. Executing an action can be thought of as pressing a button. The button can go down which corresponds to the acceptance of the action (noted $\checkmark$) or it can stay in the same position which corresponds to the failure of the action (noted $\times$). For the sake of more clarity, the success observation following action a is noted $a^{\checkmark}$ while its failure is noted $a^{\times}$. Hence, the execution of a test t on a process may result in an observation e out of a set of possible observations, noted O_t. Each test t defines a probability distribution over O_t. Definition 2.7 specifies how the observation set and the probability distribution on observations are defined for trace equivalence. The testing characterization of an equivalence notion requires a test grammar from which tests can be generated. Then, two processes are "equivalent" if, and only if, they yield the same probability distribution on observations for any test.

Equivalence between stochastic processes can also be characterized using a modal logic [21]. A modal logic is a language to generate formulae which manipulate modalities such as possibility, existence, necessity, etc. A formula can be seen as a property that might or might not be satisfied by a process. Given a suitable modal logic, two processes are equivalent according to a notion, if and only if, they satisfy the same formulae of the logic. What is interesting with logical characterization is that if we want to verify that two processes are not equivalent, we only have to find a formula that distinguishes them. Moreover, this formula gives information about why the processes are not equivalent.

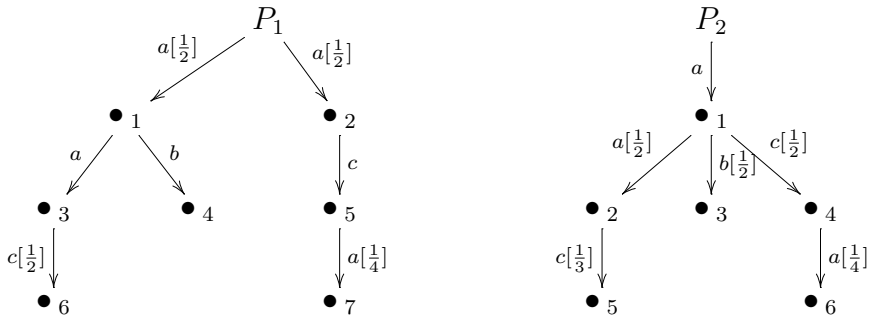


Figure 2.5: Not trace equivalent processes

2.3.1 Trace Equivalence

A trace is a sequence of actions of the form $\tau = a_1 a_2 \ldots a_n$. The empty sequence is noted ϵ. In a non-probabilistic setting, two processes are *trace equivalent* if, and only if, they accept the same set of traces. For example, making an abstraction of the probabilities values, processes P_1 and P_2 of Figure 2.5 accept the same set of traces $\{\epsilon, a, aa, ab, ac, aac, aca\}$. Probabilistic trace equivalence [23], however, requires that the probabilities (to accept each trace) are the same for both processes. To define trace equivalence, we need to extend the transition probability function to traces.

Definition 2.6. *Let $L = (\mathcal{S}, \mathrm{i}, \mathcal{A}, P)$ be an LMP, s a state in $\mathcal{S}$, $\tau = a_1 a_2 \ldots a_n$ a trace in $\mathcal{A}^*$, and X a subset of $\mathcal{S}$ ($X \subseteq \mathcal{S}$):*

$$
\begin{aligned}
P_{s \to s'}(\epsilon) &:= \begin{cases} 1 & \text{if} \quad s = s' \\ 0 & \text{otherwise} \end{cases} \\
P_{s \to s'}(\tau) &:= \sum_{t \in \mathcal{S}} P_{s \to t}(a_1 \ldots a_{n-1})\, P_{t \to s'}(a_n) \\
P_{s \to X}(\tau) &:= \sum_{s' \in X} P_{s \to s'}(\tau).
\end{aligned}
$$

Let $P^L(\tau)$ denotes $P_{\mathrm{i} \to \mathcal{S}}(\tau)$, that is, the probability to successfully perform trace τ from the initial state of L.

Two states s_1 and s_2 are trace equivalent if, and only if, $\forall\, \tau \in \mathcal{A}^*$,

$$P_{s_1 \to \mathcal{S}}(\tau) = P_{s_2 \to \mathcal{S}}(\tau).$$

The probabilities $P^{P_1}(\tau)$ and $P^{P_1}(\tau)$ to accept these traces on P_1 and P_2 of Figure 2.5 are given in Table 2.1. It is easy to see that P_1 and P_2 are not trace equivalent since they have different probabilities to accept trace aac : $P^{P_1}(aac) = \frac{1}{4}$ whereas $P^{P_2}(aac) = \frac{1}{6}$.

τ	ϵ	a	aa	ab	ac	aac	aca
$P^{P_1}(\tau)$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$
$P^{P_2}(\tau)$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{8}$

Table 2.1: Probabilities $P^{P_1}(\tau)$ and $P^{P_2}(\tau)$ in the example of Figure 2.5.

In trace equivalence, a test t is a sequence of actions of the form $t = a_1.a_2.\ldots.a_n$ where $a_1, a_2, \ldots, a_n$ are elements of $\mathcal{A}$. Testing trace equivalence consists in interacting with the testing machine in Figure 2.4. The execution of a test t on a process may result in an observation e having the form: $e = a_1^{\checkmark} a_2^{\checkmark} \ldots a_k^{\times}$ or $e = a_1^{\checkmark} a_2^{\checkmark} \ldots a_k^{\checkmark}$ where $k \leq n$. The set of all observations following a test t is noted O_t where O is an observation function. For example, $O_{a.b} = \{a^{\times}, a^{\checkmark} b^{\times}, a^{\checkmark} b^{\checkmark}\}$. Each test t defines a probability distribution over O_t.

Let $Q_t^s(e)$ be the probability of observing the observation e after running test t on state s. For example, in Figure 2.2(b), $Q_{a.a.a}^{s_0}(a^{\checkmark} a^{\checkmark} a^{\checkmark}) = \frac{3}{4}$. Given a state s, each test t defines a probability distribution Q_t^s on observations. In particular,

$$\sum_{e \in O_t} Q_t^s(e) = 1.$$

More formally, the testing characterization of trace is based on the following definition.

Definition 2.7. *The test language for trace equivalence is defined as :*

$$\mathcal{T}_{\text{trace}} \qquad t ::= \omega \mid a.t$$

where 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 observation function O is defined inductively as follows :

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

The probability distribution on observations Q_t^s is defined as :

- $Q_{\omega}^s(\omega^{\checkmark}) = 1$
- $Q_{a.t}^s(a^{\times}) = 1 - P_{s \to \mathcal{S}}(a)$
- $Q_{a.t}^s(a^{\checkmark} e) = \sum_{s' \in S} P_{s \to s'}(a) Q_t^{s'}(e) \qquad$ *where* $e \in O_t$.

For example, in Figure 2.5, $Q^{P_1}_{a.a.c.\omega}(a^\checkmark a^\checkmark c^\checkmark) = \frac{1}{4}$, whereas $Q^{P_2}_{a.a.c.\omega}(a^\checkmark a^\checkmark c^\checkmark) = \frac{1}{6}$. Although a trace and a test have the same definition (a sequence of actions), they are used differently. A trace τ is either accepted or refused whereas a test t yields a set of observations O_t. Hence, with testing we have the following theorem.

Theorem 2.8. *[30] Two LMPs L_1 and L_2 are probabilistic trace equivalent if and only if they yield the same probability distributions on observations for any test of the grammar $\mathcal{T}_{\text{trace}}$.*

As of the logical characterization, we define the modal logic $\mathcal{L}_{trace}$ and we prove that it characterizes trace equivalence.

Definition 2.9.

$$\mathcal{L}_{trace}: \qquad F ::= tt \mid \langle\tau\rangle_p$$

where $\tau \in \mathcal{A}^*$ and $p \in [0,1]$. tt is satisfied by any state and $\langle\tau\rangle_p$ is satisfied by any state that accepts trace τ with probability at least p.

Theorem 2.10. *Two states s_1 and s_2 are trace equivalent if and only if they satisfy the same formulae of $\mathcal{L}_{trace}$.*

Proof. Straightforward since a state s that accepts trace τ with probability $P_{s\to\mathcal{S}}(\tau) = p$ satisfies any formula $\langle\tau\rangle_q$ such that $q \leq p$. □

2.3.2 Bisimulation Equivalence

With trace equivalence, one can only observe the set of traces accepted from a state. However, there exist other observable properties that can further distinguish processes. For example, in Figure 2.5 the two processes P_1 and P_2 accept the same set of traces, but in P_1 after running action a the resulting state does not accept all actions a, b, and c, whereas P_2 does. Hence, an external observer can distinguish between these two processes. The main idea of bisimulation equivalence is that two processes are equivalent if and only if they exhibit the same behaviour for an external observer[1]. For this reason, it is called also observational equivalence. Larsen and Skou [30] adapted this notion to probabilistic processes and called the obtained notion *probabilistic bisimulation*.

Definition 2.11. *[30] Let $(\mathcal{S}, s_0, \mathcal{A}, P)$ be a countable LMP. An equivalence relation $\equiv$ on $\mathcal{S}$ is a bisimulation if*

[1] Bisimulation for non-probabilistic processes has been first introduced by Milner [39].

$$s \equiv s' \quad \Rightarrow \quad \forall\, a \in \mathcal{A}.\, \forall\, C \in \mathcal{S}/\equiv . \quad P_{s \to C}(a) = P_{s' \to C}(a)$$

where $\mathcal{S}/\equiv$ is the set of bisimulation equivalence classes and $P_{s \to C}(a)$ is the probability to run action a on state s and then end-up in a state of the set C.

Two states of $\mathcal{S}$ are bisimilar if there exists a bisimulation that relates them.

Larsen and Skou showed that probabilistic bisimulation can be characterized by a testing scenario [30]. Their test language has the following syntax:

Definition 2.12. *[30] The grammar of bisimulation testing is:*

$$\mathcal{T}_{\mathrm{LS}} : \qquad t ::= \omega \mid a.t \mid (t_1, \ldots, t_n)$$

where tests ω and $a.t$ are the same as in $\mathcal{T}_{\mathrm{trace}}$ while test $(t_1, \ldots, t_n)$ consists in making n copies of the current state and then executing test t_i on the i^{th} copy for $i = 1, \ldots, n$. The observation function O is recursively defined as follows :

- $O_\omega = \{\omega^{\checkmark}\}$
- $O_{a.t} = \{a^{\times}\} \cup \{a^{\checkmark} e \mid \ e \in O_t\}$
- $O_{(t_1, \ldots, t_n)} = O_{t_1} \times \ldots \times O_{t_n}$.

The probability distribution on observations is :

- $Q^s_\omega(\omega^{\checkmark}) = 1$
- $Q^s_{a.t}(a^{\times}) = 1 - P_{s \to \mathcal{S}}(a)$
- $Q^s_{a.t}(a^{\checkmark} e) = \sum_{s' \in S} P_{s \to s'}(a) Q^{s'}_t(e) \qquad$ *where* $e \in O_t$
- $Q^s_{(t_1, \ldots, t_n)}((e_1, \ldots, e_n)) = \prod_{i=1}^{n} Q^s_{t_i}(e_i) \qquad$ *where* $e_i \in O_{t_i}$.

Theorem 2.13. *[30] Two processes L_1 and L_2 are probabilistic bisimilar if and only if they yield the same probability distributions on observations for any test of $\mathcal{T}_{\mathrm{LS}}$.*

This test grammar corresponds to interacting with the testing machine of Figure 2.6 equipped with a button for each action and a replication button. When pressed, the replication button creates a new copy of the current state.

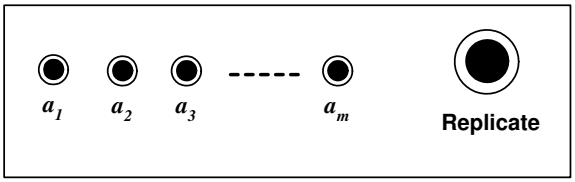


Figure 2.6: Testing machine for bisimulation

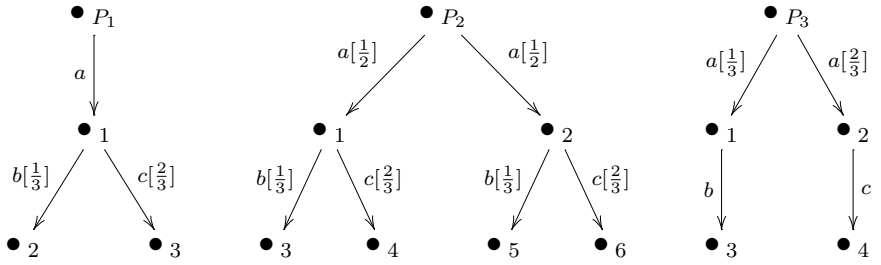


Figure 2.7: Labelled transition systems

In the context of labelled transition systems, Hennessey and Milner [17] proposed a modal logic, called *HML* which characterizes (non-probabilistic) bisimulation. The syntax of Hennessey-Milner logic is:

$$HML: \qquad F ::= \quad tt \mid \neg F \mid \ \wedge_{i\in\mathbb{N}}\ F_i \mid \langle a\rangle F$$

where $\langle a\rangle F$ is satisfied by a process that can make an a-transition to a process satisfying F, $\neg F$ represents negation, and $\wedge_{i\in\mathbb{N}} F_i$ represents conjunction. Probabilistic bisimulation for LMPs, surprisingly, is characterized by a simpler logic ($\mathcal{L}_0$) due to Desharnais et al. [8]. The syntax of $\mathcal{L}_0$ is:

$$\mathcal{L}_0: \qquad F ::= tt \mid F_1 \wedge F_2 \mid \langle a\rangle_p\ F$$

where a state satisfying $\langle a\rangle_p\ F$ means that it can jump to a state satisfying F with a probability at least p.

Processes P_1 and P_2 of Figure 2.7 are bisimilar and neither of them is bisimilar to P_3. Indeed, P_1 and P_2 satisfy the formula $\langle a\rangle_1(\langle b\rangle_{\frac{1}{3}}\, tt \ \wedge\ \langle c\rangle_{\frac{2}{3}}\, tt)$ whereas P_3 does not satisfy it. On the other hand, the test $t = a.(b.\omega, c.\omega)$ of test grammar $\mathcal{T}_{LS}$ will lead to different probability distributions on observations between P_1 and P_2, on one side, and P_3 on the other side. Indeed, $Q_t^{P_1}(a^{\checkmark}(b^{\checkmark}, c^{\checkmark})) = Q_t^{P_2}(a^{\checkmark}(b^{\checkmark}, c^{\checkmark})) = \frac{2}{9}$, whereas $Q_t^{P_3}(a^{\checkmark}(b^{\checkmark}, c^{\checkmark})) = 0$.

Reasoning about traces in the context of process algebras has been first introduced by Hoare [18]. Hoare described also other semantics based on traces such as *failure*

and *readiness*. These semantics suggest other equivalence notions that are finer than trace equivalence and can be directly defined using traces. In the following sections, we present probabilistic versions of these notions, namely, ready equivalence, failure equivalence, Barb acceptance equivalence, and Barb failure equivalence. In order to be consistent in our illustration, we define these notions in the same way as trace and bisimulation; that is, we try to view them from the logical and testing point of view in addition to their main definitions. Most of the material we present in the following sections is the result of our own investigation.

2.3.3 Ready Equivalence

Ready equivalence [18] is a finer equivalence than trace. In a non-probabilistic setting, two processes are ready equivalent if and only if they can run trace τ and then end up in a state which accepts the same set of actions $\{a, b, c\}$. For example, process P_1 in Figure 2.7 can perform trace $\tau = a$ and then end in a state which accepts the set $\{b, c, d\}$. For probabilistic processes, ready equivalence [23] requires that both processes perform trace τ with the same probability and then end up in a state accepting the same set of actions.

Definition 2.14. *Let s be a state in $\mathcal{S}$. The set of actions enabled in s is defined by:*

$$Can(s) := \{a \in \mathcal{A} \mid \exists s' \in \mathcal{S} \wedge P_{s \to s'}(a) > 0\}.$$

Definition 2.15. *Let $R : \mathcal{S} \times \mathcal{A}^* \times \mathcal{P}(\mathcal{A}) \to [0, 1]$ be the ready function which, given a state s, a sequence of actions τ and a set of actions $X \in \mathcal{P}(\mathcal{A})$[2], returns the probability that a state s performs trace τ successfully and ends up in a state which accepts the set of actions X:*

$$R(s, \tau, X) := \sum_{\{s' \in \mathcal{S} \mid Can(s') = X\}} P_{s \to s'}(\tau).$$

Two states s_1 and s_2 are ready equivalent if and only if $\forall\, \tau \in \mathcal{A}^*, \forall\, X \in \mathcal{P}(\mathcal{A})$,

$$R(s_1, \tau, X) = R(s_2, \tau, X).$$

For example, processes P_1 and P_2 of Figure 2.5 are not ready equivalent since they differ in several ready function values (Table 2.2).

Ready equivalence can be characterized by the following modal logic (the proof follows).

[2] $\mathcal{P}(\mathcal{A})$ represents the powerset of the set of actions $\mathcal{A}$.

Definition 2.16.

$$\begin{aligned} \mathcal{L}_{ready}: \qquad F &:= \langle\tau\rangle_p\ \phi\ tt \\ \phi &:= \wedge_{i\in\mathbb{N}} a_i \end{aligned}$$

where $\langle\tau\rangle_p\ \phi\ tt$ is satisfied by any state which accepts trace τ with probability at least p and then ends in a state satisfying ϕ while $a_1 \wedge a_2 \wedge \ldots \wedge a_n$ is satisfied by a state s if, and only if, $\{a_1, a_2, \ldots a_n\} = Can(s)$.

Theorem 2.17. *Two states s_1 and s_2 are ready equivalent if and only if they satisfy the same formulae of $\mathcal{L}_{ready}$.*

Proof. $\Rightarrow$. The proof is by contradiction. Suppose that s_1 and s_2 do not satisfy the same set of formulae of $\mathcal{L}_{ready}$, then it is sufficient to find a trace τ and a set of actions X such that $R(s_1, \tau, X) \neq R(s_2, \tau, X)$. Let $f = \langle\tau\rangle_p\ a_1 \wedge a_2 \wedge \ldots \wedge a_k\ tt$ a formula of $\mathcal{L}_{ready}$ and suppose, w.l.o.g., that it is satisfied by s_1 but not by s_2. Let $X = \{a_1, \ldots, a_k\}$. By Definition 2.16, we have

$$\sum_{\{s'\in\mathcal{S}\,|\,Can(s')=X\}} P_{s_1\to s'}(\tau) \geq p$$

which by Definition 2.15 implies that $R(s_1, \tau, X) \geq p$. On the other hand, $s_2 \not\models \langle\tau\rangle_p\ a_1 \wedge a_2 \wedge \ldots \wedge a_k\ tt$ implies that

$$\sum_{\{s'\in\mathcal{S}\,|\,Can(s')=X\}} P_{s_2\to s'}(\tau) < p.$$

Hence, $R(s_1, \tau, X) > R(s_2, \tau, X)$ which implies that s_1 and s_2 are not ready equivalent. $\Leftarrow$. The proof is by contradiction. Suppose that s_1 and s_2 are not ready equivalent, then it is sufficient to prove that s_1 and s_2 do not satisfy the same set of formulae of $\mathcal{L}_{ready}$. Let τ be a trace and $X = \{a_1, \ldots, a_k\}$ a set of actions such that $R(s_1, \tau, X) \neq R(s_2, \tau, X)$ and assume, w.l.o.g., that $R(s_1, \tau, X) = p > R(s_2, \tau, X)$. Then, the formula $\langle\tau\rangle_p\ a_1 \wedge a_2 \wedge \ldots \wedge a_k\ tt$ is satisfied by s_1 but not by s_2. □

Let us discuss the testing characterization of ready equivalence. As illustrated in the previous section, the property that a state s performs trace τ with probability p can be tested. Unfortunately, the property that the set of actions $\{a_1, a_2, \ldots, a_n\}$ are enabled in a state s cannot be tested. Indeed, the property that action a is enabled in a state (regardless of the probability) is not testable. Take, for example two states: one state accepting action a with probability 1 and the other accepting a with probability 10^{-80}. Action a is enabled in both states. However, in the latter state, it is almost impossible to figure it out through testing. This indicates that ready equivalence as defined above does not have a testing characterization. In Chapter 6, we present a variant of ready equivalence which is testable.

(τ, X)	$(\epsilon, \{a\})$	$(a, \{a, b\})$	$(a, \{c\})$	$(aa, \{c\})$	$(ab, \emptyset)$	$(ac, \{a\})$
$R(P_1, \tau, X)$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$R(P_2, \tau, X)$	1	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
(τ, X)	$(aac, \emptyset)$	$(aca, \emptyset)$	$(a, \{a, b, c\})$			
$R(P_1, \tau, X)$	$\frac{1}{4}$	$\frac{1}{8}$	0			
$R(P_2, \tau, X)$	$\frac{1}{6}$	$\frac{1}{8}$	1			

Table 2.2: Ready function values for processes P_1 and P_2 of the example of Figure 2.5 (zero probabilities are discarded).

2.3.4 Failure Equivalence

Failure equivalence [23] is very similar to ready equivalence, but instead of considering the set of actions accepted after a given trace, it considers the set of actions refused.

Definition 2.18. *Let $F : \mathcal{S} \times \mathcal{A}^* \times \mathcal{P}(\mathcal{A}) \to [0, 1]$ be the failure function which given a state s, a sequence of actions τ and a set of actions X, returns the probability that a state s performs trace τ successfully and ends up in a state where all actions in X are not enabled:*

$$F(s, \tau, X) := \sum_{\{s' \in \mathcal{S} \,|\, Can(s') \cap X = \emptyset\}} P_{s \to s'}(\tau).$$

Two states s_1 and s_2 are failure equivalent if and only if $\forall\, \tau \in \mathcal{A}^*, \forall\, X \in \mathcal{P}(\mathcal{A})$,

$$F(s_1, \tau, X) = F(s_2, \tau, X).$$

The values of $F(P_1, \tau, X)$ and $F(P_2, \tau, X)$ for any $\tau \in \mathcal{A}^*$ and $X \in \mathcal{P}(\mathcal{A})$ of Figure 2.5 are given in Table 2.3.

Failure equivalence can be characterized by the following modal logic.

Definition 2.19.

$$\mathcal{L}_{failure}: \qquad \begin{aligned} F &:= \langle \tau \rangle_p \ \phi \ tt \\ \phi &:= \wedge_{i \in \mathbb{N}} \neg a_i \end{aligned}$$

where $\langle \tau \rangle_p \ \phi$ is satisfied by any state which accepts trace τ with probability at least p and then end up in a state satisfying ϕ while $\neg a_1 \wedge \neg a_2 \wedge \ldots \wedge \neg a_n$ is satisfied by a state s if and only if $\{a_1, a_2, \ldots a_n\} \cap Can(s) = \emptyset$.

Theorem 2.20. *Two states s_1 and s_2 are failure equivalent if and only if they satisfy the same formulae of $\mathcal{L}_{failure}$.*

(τ, A)	$(\epsilon, \emptyset)$	$(\epsilon, \{b\})$	$(\epsilon, \{c\})$	$(\epsilon, \{b, c\})$	
$F(P_1, \tau, A)$	1	1	1	1	
$F(P_2, \tau, A)$	1	1	1	1	
(τ, A)	$(a, \emptyset)$	$(a, \{a\})$	$(a, \{b\})$	$(a, \{c\})$	$(a, \{a, b\})$
$F(P_1, \tau, A)$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$F(P_2, \tau, A)$	1	0	0	0	0
(τ, A)	$(aa, \emptyset)$	$(aa, \{a\})$	$(aa, \{b\})$	$(aa, \{a, b\})$	
$F(P_1, \tau, A)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	
$F(P_2, \tau, A)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	
(τ, A)	$(ac, \emptyset)$	$(ac, \{b\})$	$(ac, \{c\})$	$(ac, \{b, c\})$	
$F(P_1, \tau, A)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	
$F(P_2, \tau, A)$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	
(τ, A)	$(ab, -)$	$(aac, -)$	$(aca, -)$		
$F(P_1, \tau, A)$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$		
$F(P_2, \tau, A)$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{8}$		

Table 2.3: Failure function values for processes P_1 and P_2 of the example of Figure 2.5 (zero probabilities are discarded).

The proof is very similar to the proof of Theorem 2.17.

In reactive processes, ready and failure equivalences have the same distinguishing capabilities. Indeed, it is easy to show that if $R(s_1, \tau, X) \neq R(s_2, \tau, X)$ then there exists $Y = \mathcal{S} \backslash \{X\}$ such that $F(s_1, \tau, Y) \neq F(s_2, \tau, Y)$.

2.3.5 Barb Acceptance Equivalence

As described above, ready and failure equivalences consider the set of actions accepted or refused after the end of a trace. Barb equivalences [35], however, consider information about acceptance and refusal at intermediate states.

Definition 2.21. *A Barb trace* $Btr \in (\mathcal{A}^* \times \mathcal{P}(\mathcal{A}^*))$ *has the form*

$$\big((a_1, a_2, \ldots, a_n), (X_1, X_2, \ldots, X_{n+1})\big)$$

where $a_i \in \mathcal{A}$ *and* $X_i \in \mathcal{P}(\mathcal{A})\ \forall\, i$.

In Barb acceptance equivalence, a state s accepts a Barb trace $Btr = \big((a_1, a_2, \ldots, a_n), (X_1, X_2, \ldots, X_{n+1})\big)$ with probability p it can perform trace $\tau = a_1, a_2, \ldots, a_n$ with probability p such that in intermediate states it has respectively acceptance sets $X_1, X_2, \ldots, X_{n+1}$.

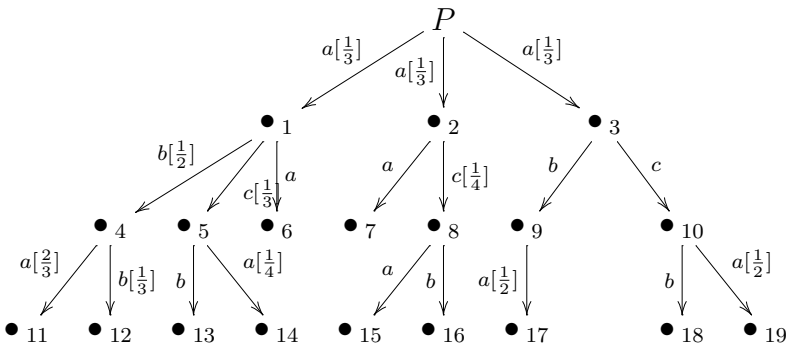


Figure 2.8: Process P

Btr	$(acb, \{a\}\{a,b,c\}\{a,b\}\emptyset)$	$(acb, \{a\}\{a,c\}\{a,b\}\emptyset)$	$(acb, \{a\}\{b,c\}\{a,b\}\emptyset)$
$BarbAcc(P, Btr)$	$\frac{1}{9}$	$\frac{1}{12}$	$\frac{1}{3}$

Table 2.4: Some Barb traces and their corresponding *BarbAcc* probability values in process P of Figure 2.8.

It is important to note that an action a_i must belong to the acceptance set X_i. Two processes are Barb acceptance equivalent if and only if they accept all Barb traces with the same probability.

Definition 2.22. *Let* $BarbAcc : \mathcal{S} \times (\mathcal{A}^* \times \mathcal{P}(\mathcal{A})^*) \to [0,1]$ *be the Barb acceptance function which, given a state* s *and a Barb trace* $Btr = \big((a_1, a_2, \ldots, a_n), (X_1, X_2, \ldots, X_{n+1})\big)$*, returns the probability that state* s *performs the Barb trace* Btr *with success:*

$$BarbAcc(s, Btr) := \sum_{\{s' \in \mathcal{S} \,|\, Can(s') = X_{n+1}\}} P_{s \to s'}^{BarbA}(a_1 \ldots a_n)$$

where $P_{s \to s'}^{BarbA}(a_1 \ldots a_n)$ *is defined inductively as*

$$P_{s \to s'}^{BarbA}(a_1 \ldots a_n) = \sum_{\{t \in \mathcal{S} \,|\, Can(t) = X_n\}} P_{s \to t}^{BarbA}(a_1 \ldots a_{n-1}) P_{t \to s'}(a_n).$$

Two states s_1 *and* s_2 *are Barb acceptance equivalent if and only if* $\forall\, Btr \in (\mathcal{A}^* \times \mathcal{P}(\mathcal{A})^*)$,

$$BarbAcc(s_1, Btr) = BarbAcc(s_2, Btr).$$

Table 2.4 shows some Barb traces and the corresponding *BarbAcc* probability values in process P of Figure 2.8.

With a similar argument as the one presented for ready equivalence, Barb acceptance equivalence cannot be tested.

Btr	$(acb, \{b,c\}\emptyset\{c\}\{a,b,c\})$	$(aca, \{b\}\{b,a\}\{c\}\{a,b\})$	$(aa, \{b,c\}\{b\}\{a,b,c\})$
$BarbRef(P, Btr)$	$\frac{19}{36} = (\frac{1}{9} + \frac{1}{12} + \frac{1}{3})$	0	$\frac{1}{3}$

Table 2.5: Some Barb traces and their corresponding $BarbRef$ probability values in process P of Figure 2.8.

2.3.6 Barb Refusal Equivalence

Barb refusal equivalence [35] considers refusal information at intermediate states. In Barb refusal a Barb trace $Btr = \big((a_1, a_2, \ldots, a_n), (X_1, X_2, \ldots, X_{n+1})\big)$ is interpreted as follows: a state s accepts Btr with probability p if s can perform trace $\tau = a_1, a_2, \ldots, a_n$ with probability p such that in intermediate states it has respectively refusal sets $X_1, X_2, \ldots, X_{n+1}$. Note that an action a_i must not be part of the refusal set X_i.

Definition 2.23. *Let $BarbRef : \mathcal{S} \times (\mathcal{A}^* \times \mathcal{P}(\mathcal{A})^*) \to [0,1]$ be the Barb refusal function which given a state s and a Barb trace $Btr = \big((a_1, a_2, \ldots, a_n), (X_1, X_2, \ldots, X_{n+1})\big)$, returns the probability that state s accepts the Barb trace Btr:*

$$BarbRef(s, Btr) := \sum_{\{s' \in \mathcal{S} \,|\, Can(s') \cap X_{n+1} = \emptyset\}} P_{s \to s'}^{BarbR}(a_1 \ldots a_n)$$

where $P_{s \to s'}^{BarbR}(a_1 \ldots a_n)$ is defined inductively as

$$P_{s \to s'}^{BarbR}(a_1 \ldots a_n) = \sum_{\{t \in \mathcal{S} \,|\, Can(t) \cap X_n = \emptyset\}} P_{s \to t}^{BarbR}(a_1 \ldots a_{n-1}) P_{t \to s'}(a_n).$$

Two states s_1 and s_2 are Barb refusal equivalent if and only if $\forall\, Btr \in (\mathcal{A}^* \times \mathcal{P}(\mathcal{A})^*)$,

$$BarbRef(s_1, Btr) = BarbRef(s_2, Btr).$$

Table 2.5 shows some Barb traces and the corresponding $BarbRef$ probability values in process P of Figure 2.8.

2.4 Metrics for Stochastic Systems

All equivalence notions for probabilistic processes presented so far are sensitive to the exact values of transition probabilities. That is, a slight difference in a single transition probability between two equivalent processes is sufficient to conclude that they are no longer equivalent. Knowing that generally the probabilities come from approximations,

these slight differences should not necessarily be interpreted as non equivalence. Instead, a more suitable approach would be to consider a metric between stochastic processes in order to quantify how far apart the processes are. A metric d is a function that yields a real number (distance) for each pair of processes. In order to be a metric function, d should satisfy the following properties:

1. $d(P, Q) = 0$ implies that $P = Q$
2. $d(P, Q) = d(Q, P)$ (symmetry)
3. $d(P, R) \leq d(P, Q) + d(Q, R)$ (triangle inequality).

The first property requires that the zero distance coincides with an exact matching between the two processes. This is too restrictive. Instead, it would be more interesting to generate a different metric function for each equivalence notion such that the equality $P = Q$ in the first property is replaced by P and Q being equivalent with respect to that equivalence notion. By doing so, the function is no longer a metric and is more accurately called a *pseudo-metric*. However, we will continue to use the term "metric" in this thesis to refer to these pseudo-metrics. Since each one of the equivalence notions presented so far focuses on a particular aspect of processes, this will result in a collection of metric functions with different capabilities.

Generally, in metrics, the actual numerical values are not very significant. Instead, properties such as the significance of zero distance and relative distance of processes are more relevant. This section is a survey of the main approaches in the literature focusing on metrics and distances for stochastic processes. It starts by presenting fundamental concepts which can be very useful to compare stochastic processes, namely, *entropy*, *cross entropy*, and in particular *relative entropy* which is a measure of the difference between two probability distributions.

Definition 2.24. *A random variable $X : S \to R$ is a measurable function from a sample space S to the measurable space (sometimes called alphabet) A of possible values of the variable.*

Example 2.25. *Consider the experiment of rolling a die and suppose that we are concerned about whether the outcome is even or odd. The sample space is obviously the set $\{1, 2, 3, 4, 5, 6\}$: the set of possible outcomes of the experiment. The measurable space can be the set $\{0, 1\}$ where 0 designates odd and 1 even. Then, the random variable X can be defined as:*

$$X(\omega) := \begin{cases} 0 & \text{if } \omega = 1, 3, \text{ or } 5 \\ 1 & \text{if } \omega = 2, 4, \text{ or } 6. \end{cases}$$

A random variable is usually equipped with a probability distribution which is a probability measure defined over the measurable space. Let X be a random variable with a probability distribution p. Then

$$p(x) = Pr(X = x)$$

for each x in the alphabet of X. If the die of the above example is uniform then $p(0) = p(1) = \frac{1}{2}$.

2.4.1 Measures of Entropy

The entropy of a random variable is a measure of its uncertainty. It measures the amount of information required on the average to describe the random variable.

Definition 2.26. *Let X be a random variable with an alphabet A and equipped with the probability distribution p. The entropy of X, $H(X)$ (sometimes noted $H(p)$), is defined as:*

$$H(X) = -\sum_{x \in \mathrm{A}} p(x) \log_2 p(x).$$

The value of the entropy is expressed in bits. To clarify the notion of entropy and the bits unit, consider the following example.

Example 2.27. *Let X be a random variable with alphabet $\mathrm{A} = \{a, b, c, d\}$ and probability distribution p such that: $p(X = a) = \frac{1}{4}$, $p(X = b) = \frac{1}{3}$, $p(X = c) = \frac{5}{24}$, and $p(X = d) = \frac{5}{24}$. Suppose that we wish to determine the value of X with the minimum number of binary questions. The best first question is "Is $X = b$?". This has probability $\frac{1}{3}$ that the answer is yes. If, however, the answer is no, the best second question would be: "Is $X = a$? ". If in turn the answer is no, the next question can be "Is $X = c$?". The entropy of X corresponds in this example to the average number of binary questions:*

$$H(X) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{5}{24} \log_2 \frac{5}{24} - \frac{5}{24} \log_2 \frac{5}{24} = 1.96 \text{ bits.}$$

The cross entropy is another measure which indicates to which extent a probability distribution matches a second probability distribution.

Definition 2.28. *Let X be a random variable with alphabet A and p and q two probability distributions over the domain of X. The cross entropy between p and q is defined as:*

$$H(p \,\|\, q) = -\sum_{x \in \mathrm{A}} p(x) \log_2(q(x)).$$

Intuitively, consider that the first distribution p is the target distribution and that q is some estimation of p. Cross entropy measures how well the distribution q fits the target distribution p. Note that, if $p = q$ then the cross entropy is simply the entropy of p ($H(p, q) = H(p)$).

The relative entropy or the Kullback-Leibler divergence[3] is a measure of the difference between two probability distributions.

Definition 2.29. *Let X be a random variable with alphabet* A *and p and q two probability distributions of X. The relative entropy between p and q is defined as:*

$$KL(p \,\|\, q) = \sum_{x \in \mathrm{A}} p(x) \, \log_2 \frac{p(x)}{q(x)}.$$

The relative entropy $KL(p \,\|\, q)$ is the extra cost, in bits, of assuming that the distribution is q when the true distribution is p. $KL(p \,\|\, q)$ cannot have negative values (see [7] for a detailed proof) and is zero if and only if $p = q$. Since it is neither symmetric nor satisfying the triangle inequality, KL is called a divergence.

Example 2.30. *Consider the random variable X of the die experiment defined in Example 2.25. Let p and q two probability distributions such that:*

$$\begin{array}{lll} p(0) = \frac{1}{2} & & q(0) = \frac{3}{4} \\ p(1) = \frac{1}{2} & \textit{and} & q(1) = \frac{1}{4}. \end{array}$$

Then,

$$\begin{array}{ll} H(p) = 1 & H(q) = 0.81125 \\ H(p \,\|\, q) = 1,2075 & H(q \,\|\, p) = 1 \end{array}$$

and,

$$\begin{array}{rcl} KL(p \,\|\, q) & = & \frac{1}{2} \log_2 \frac{\frac{1}{2}}{\frac{3}{4}} + \frac{1}{2} \log_2 \frac{\frac{1}{2}}{\frac{1}{4}} = 1 - \frac{1}{2} \log_2 3 = 0.2075 \ \textit{bits} \\ KL(q \,\|\, p) & = & 0.1887 \ \textit{bits}. \end{array}$$

By expanding the definition of relative entropy, one can recover an interesting rela-

[3] The relative entropy is known under a variety of other names including information divergence, information for discrimination, etc.

tion between entropy, cross entropy and the relative entropy.

$$\begin{aligned} KL(p \,\|\, q) &= \sum_{x \in \mathrm{A}} p(x)\ \log_2 \frac{p(x)}{q(x)} \\ &= \sum_{x \in \mathrm{A}} p(x)\,(\log_2 p(x) - \log_2 q(x)) \\ &= \sum_{x \in \mathrm{A}} p(x) \log_2 p(x) - \sum_{x \in \mathrm{A}} p(x) \log_2 q(x) \\ &= -\sum_{x \in \mathrm{A}} p(x) \log_2 q(x) + \sum_{x \in \mathrm{A}} p(x) \log_2 p(x) \\ &= H(p \,\|\, q) - H(p). \end{aligned} \tag{2.1}$$

Putting all this together, we can summarize this section by describing the three entropy measures as follows:

- the entropy $H(p)$ measures the minimum number of bits needed to identify an event x in the alphabet of X if the probability distribution p is used,
- the cross entropy $H(p \,\|\, q)$ measures the average number of bits needed to identify an event x in the alphabet of X if a probability distribution q is used rather than the "true" distribution p,
- the relative entropy $KL(p \,\|\, q)$ measures the number of *extra* bits needed to identify an event x in the alphabet of X if a probability distribution q is used rather than the "true" distribution p.

We will see later how the KL divergence is used to define a stochastic game that is behind the main contribution of this thesis.

2.4.2 Distance Measure for Hidden Markov Models

The first metric for stochastic processes that we present measures distances between Hidden Markov Models (HMMs). An HMM is a stochastic model where the state space is not observable: one cannot see the current state at a given moment. A detailed definition of HMMs will be given in Chapter 7. But for the sake of presenting two distance measures for HMMs, it is sufficient to regard a HMM as a transition system with a single action and where each visited state generates an observation out of a set of possible observations, governed by a probability distribution. More formally a HMM λ with N states and M observations is a tuple (μ, A, B) where:

- μ is the initial state probability distribution
- A is an $N \times N$ state transition probability matrix, and
- B is an $N \times M$ observation probability matrix.

Example 2.31. *Let $\lambda_1 = (\mu_1, A_1, B_1)$ be a HMM with 4 states and 3 observations:*

$$A_1 = \begin{pmatrix} 0 & 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.4 & 0.1 \\ 0.2 & 0 & 0.5 & 0.3 \\ 0 & 0.2 & 0.2 & 0.6 \end{pmatrix} \qquad B_1 = \begin{pmatrix} 0.5 & 0.2 & 0 \\ 0.1 & 0.3 & 1 \\ 0.4 & 0 & 0 \\ 0 & 0.5 & 0 \end{pmatrix}$$

$$\mu_1 = \begin{pmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}$$

$a_{23} = 0.4$ in the matrix A_1 represents the probability to make a transition from state 2 to state 3. $b_{12} = 0.1$ in matrix B_1 represents the probability that observation 2 is generated when the model is in state 1.

Rabiner et al. [41] used HMMs to model words in an automatic speech recognition system[4]. The HMM parameters (μ, A, and B) can be estimated iteratively from a set of observation sequences: at each iteration, the current estimation is improved and a new HMM is generated. This loop stops when the distance between two successive HMMs becomes negligeable. Rabiner et al. developed two distance measures for HMMs: one is based on a state permutation function [33] and the other is inspired by relative entropy [24]. We now briefly present these distances.

The first distance measure is given by the following equation. Let $\lambda = (\mu, A, B)$ and $\hat{\lambda} = (\hat{\mu}, \hat{A}, \hat{B})$ be two HMMs with N states and M observations. Then,

$$d(\lambda\,,\,\hat{\lambda}) \;=\; \|B - \hat{B}\| \;=\; \Big(\frac{1}{M\,N}\sum_{j=1}^{N}\sum_{k=1}^{M}[\,b_{jk} - \hat{b}_{f(j)k}\,]^2\Big)^{\frac{1}{2}} \tag{2.2}$$

where b_{jk} is the probability to get observation k form state j in HMM λ and $f(j)$ is the state permutation that minimizes the measure of the equation. The optimal state permutation is determined by using a technique of minimum bipartite matching (Appendix B of [33]). It is interesting to notice that the distance measure of Equation (2.2) depends only on the observation matrix B. This can be explained by the fact that for HMMs, the matrix B has significantly more impact on the distance than the other parameters μ and A. Indeed, HMM model parameters are most of the time estimated on the basis of the observed events. At the same time, this characteristic constitutes an argument

[4]This application is detailed in Appendix B.

against this measure since it would be more obvious to include the other parameters. In addition, the evaluation of this measure requires a great deal of computation.

The second distance measure is inspired by relative entropy KL. Let $\lambda = (\mu, A, B)$ be a HMM and let $O_T = o_1 o_2 \ldots o_T$ be a sequence of observations of length T and $S_T = s_0 s_1 \ldots s_T$ the corresponding hidden state sequence. Then, $\nu(O_T \,|\, \lambda)$ represents the probability that HMM λ accepts the observation sequence O_T and is defined as:

$$\nu(O_T \,|\, \lambda) = \sum_{all\ S_T = s_0 s_1 \ldots s_T} \mu(s_0) \prod_{t=1}^{T} a_{s_{t\text{-}1} s_t}\, b_{s_t o_t}.$$

Let $\lambda = (\mu, A, B)$ and $\hat{\lambda} = (\hat{\mu}, \hat{A}, \hat{B})$ be two HMMs. Then, the distance measure $D(\lambda\,,\,\hat{\lambda})$ is defined as:

$$D(\lambda\,,\,\hat{\lambda}) = \lim_{T \to \infty} \frac{1}{T} \left[\, \log_2\, \nu(O_T \,|\, \lambda) - \log_2\, \nu(O_T \,|\, \hat{\lambda}) \right]$$

where O_T is an observation sequence generated according to the distribution $\nu(\,.\,|\,\lambda)$. It is important to mention that the distance measure $D(\lambda\,,\,\hat{\lambda})$ requires that the pair of HMMs being compared both be *ergodic*. An ergodic model is a model where every state can be reached from any other state in a finite number of steps. The distance measure $D(\lambda\,,\,\hat{\lambda})$ is not symmetric. A symmetrized version can be simply generated:

$$D_S(\lambda\,,\,\hat{\lambda}) = \frac{1}{2} \left[\, D(\lambda\,,\,\hat{\lambda}) + D(\hat{\lambda}\,,\,\lambda) \,\right].$$

These two distances proved their efficiency in practice, especially in automatic speech recognition systems, but it is important to mention that both distances rely on the knowledge of the HMM models. The two distances have been presented to the sake of comparison with the technique we propose. Indeed, like the second metric presented above, the technique we propose in this thesis to estimate the divergence between processes is based also on the relative entropy KL. The difference, however, is that our technique does not require knowledge of the models.

2.4.3 A Metric for Labelled Markov Processes

The metric we describe in this section is due to Desharnais et al. [9] and is inspired by the logical characterization of bisimulation (Section 2.3.2). Recall that to show the difference between two LMPs, it is sufficient to come up with a formula that is satisfied in one process but not in the other. More complex distinguishing formulas indicate closer processes. Indeed, if the formula is very large, then only a long sequence

of observations will distinguish the processes. Capturing this intuition with logical formulas is not possible since the formulas yield truth values in $\{0, 1\}$. The idea is then to propose a quantitative analogue of logical formulas which are called functional expressions. A functional expression is interpreted on a state of the process and yields a numerical value in the interval $[0, 1]$. This value corresponds to a quantitative measure of the extent to which the state satisfies a logical formula. Recall that the logical characterization of bisimulation is given by the logic $\mathcal{L}_0$:

$$F ::= tt \mid F_1 \wedge F_2 \mid \langle a \rangle_p \, F.$$

For each $c \in (0, 1]$, $\mathcal{F}^c$ represents a family of functional expressions generated by the following grammar:

$$f := 1 \mid 1 - f \mid \langle a \rangle \, f \mid min(f_1, f_2) \mid \sup_{i \in \mathbb{N}} f_i \mid f \ominus q$$

where q is rational. A functional expression is not a function until it is interpreted in a process.

Let $L = (\mathcal{S}, s_0, \mathcal{A}, P)$ be an LMP. Then, the interpretation of the functional expression f in L yields the function $f_L : \mathcal{S} \longrightarrow [0, 1]$ (the subscript is discarded when no confusion can arise) such that, for $s \in \mathcal{S}$,

- $1(s) = 1$
- $(1 - f)(s) = 1 - f(s)$
- $\langle a \rangle \, f(s) = c \sum_{t \in \mathcal{S}} P(s, a, t) \, f(t)$
- $(f \ominus q)(s) = max(f(s) - q, 0)$

c appears only in the interpretation of $\langle a \rangle \, f(s)$ and its role is to discount the effect of actions that happen later in the future. If $c = 1$, then all actions have the same effect regardless of their position in the future.

To see intuitively the relationship between the syntax of $\mathcal{F}^c$ and the logic $\mathcal{L}_0$, let us make an informal analogy between the connectives of $\mathcal{F}^c$ and the ones of $\mathcal{L}_0$. tt maps to the functional 1, the conjunction maps to the *min* connective, and $\langle a \rangle_q$ is split up into two expressions: $\langle a \rangle \, f$, which corresponds to prefixing and, $f \ominus q$, which captures the greater than q idea.

Example 2.32. *Consider the LMPs $L_1 = (\mathcal{S}_1, s_0, \mathcal{A}, P)$ and $L_2 = (\mathcal{S}_2, t_0, \mathcal{A}, P')$ of Figure 2.9. The formula $F = \langle a \rangle_{\frac{1}{3}} (\langle a \rangle \, T \wedge \langle b \rangle \, T)$ is one of the formulas that witness*

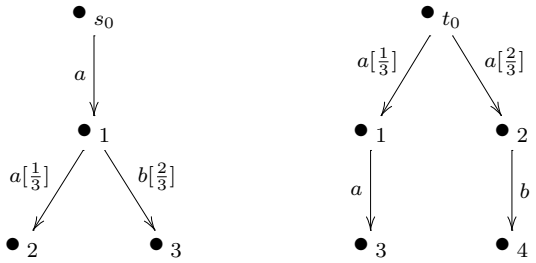


Figure 2.9: LMPs L_1 and L_2

the non bisimilarity of L_1 and L_2. Indeed, it is satisfied in L_1 but not in L_2. Let $f = \langle a \rangle \, min(\langle a \rangle \, T, \langle b \rangle \, T)$ be a functional expression inspired by F. Then,

$$f_{L_1} = f(s_0) \quad = c \; 1 \; min(c \; \tfrac{1}{3} \, , \, c \; \tfrac{2}{3}) = \tfrac{c^2}{3}$$
$$f_{L_2} = f(t_0) \quad = c \; \tfrac{1}{3} \; min(c \, , \, 0) + c \; \tfrac{2}{3} \; min(c \, , \, 0) = 0.$$

The soundness and completeness of this function set with respect to bisimulation are expressed in the following theorem.

Theorem 2.33. *[9] For any LMPs L and L' and for any $c \in (0, 1]$*

$$s \in \mathcal{S} \text{ and } s' \in \mathcal{S}' \text{ are bisimilar} \quad \Leftrightarrow \quad \forall \, f \in \mathcal{F}^c \;\; f_L(s) = f_{L'}(s').$$

Each collection $\mathcal{F}^c$ of functional expressions induces a distance measure between two LMPs L and L':

$$d^c(L \, , \, L') = \sup_{f \in \mathcal{F}^c} | \; f_L(s_0) \; - \; f_{L'}(s'_0) \; |.$$

2.4.4 ϵ-bisimulation Metric

This metric is due to Giaclone et al. [16]. It has been defined for a particular class of processes called deterministic PCCS (Probabililistic Calculus of Communicating Systems) processes. The word deterministic is used in the sense that, for any action a, a PCCS process has at most one transition labelled with a. The metric is based on a weak form of bisimulation called ϵ-bisimulation, denoted $\overset{\epsilon}{\sim}$. Two processes P and Q are ϵ-bisimilar ($P \overset{\epsilon}{\sim} Q$) if P and Q can simulate each other with a bound ϵ of deviation in probability.

Definition 2.34. *For $\epsilon \in [0, 1)$, a relation $\mathcal{R}_\epsilon$ is called ϵ-bisimulation if $(P, Q) \in \mathcal{R}_\epsilon$ implies $\forall a \in \mathcal{A}$,*

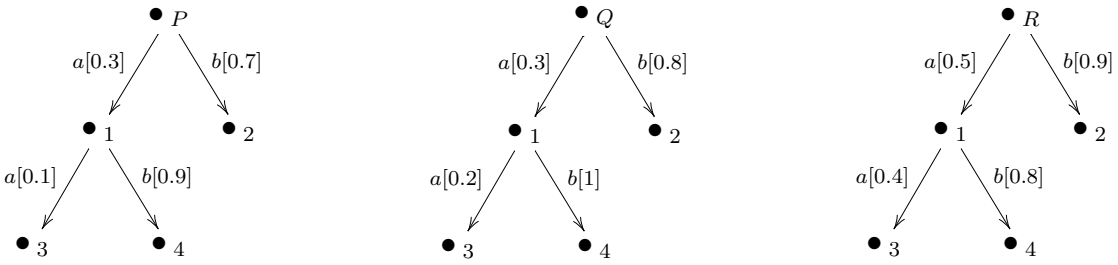


Figure 2.10: ϵ-bisimilar processes

i. if $P \xrightarrow{a[p]} P'$ *then* $\exists Q'$ *such that* $Q \xrightarrow{a[q]} Q'$, $|\, p - q \,| \leq \epsilon$, *and* $(P', Q') \in \mathcal{R}_\epsilon$

ii. if $Q \xrightarrow{a[q]} Q'$ *then* $\exists P'$ *such that* $P \xrightarrow{a[p]} P'$, $|\, p - q \,| \leq \epsilon$, *and* $(P', Q') \in \mathcal{R}_\epsilon$.

where $P \xrightarrow{a[p]} P'$ *means that process* P *can perform an* a*-labelled transition to* P' *with probability* p.

Two processes P *and* Q *are said to be* ϵ*-bisimilar if there exists an* ϵ*-bisimulation* $\mathcal{R}_\epsilon$ *such that* $(P, Q) \in \mathcal{R}_\epsilon$.

Based on ϵ-bisimulation, Giaclone et al. defined a metric for deterministic PCCS processes. Intuitively, the distance between two processes P and Q is ϵ if they are related by $\overset{\epsilon}{\sim}$ and by no other $\overset{\epsilon'}{\sim}$ such that $\epsilon' < \epsilon$. That is,

$$\rho(P\,,\,Q) := \begin{cases} \inf \;\; \epsilon & \text{such that } P \overset{\epsilon}{\sim} Q \\ 1 & \text{otherwise.} \end{cases}$$

Example 2.35. *Consider the three deterministic PCCS processes* P, Q, *and* R *of Figure 2.10. It is easy to see that* $P \overset{0.1}{\sim} Q$, $Q \overset{0.2}{\sim} R$, *and* $P \overset{0.3}{\sim} R$. *Hence,* $\rho(P\,,\,Q) = 0.1$, $\rho(Q\,,\,R) = 0.2$, *and* $\rho(P\,,\,R) = 0.3$.

This ϵ-bisimulation based metric is symmetric and does satisfy the triangle inequality (the proof is simple and can be found in [16]). However, for non-deterministic PCCS processes, the metric fails to satisfy the triangle inequality. To the best of our knowledge the ϵ-bisimulation metric has not been used in practice.

Other metrics were proposed by Van Breugel and Worrel [58, 57] and in the non-deterministic case by Kwiatkowska and Norman [27, 28].

2.5 Conclusion

In this chapter, we introduced stochastic models used in formal verification, in particular labelled Markov processes (LMP). Then, we went through the most popular probabilistic equivalence notions for stochastic processes and we discussed their testing and logical characterizations. However, we showed that with stochastic processes, it is more practical to consider distance or divergence measures. The last section of this chapter was a survey of these metrics. An interesting feature shared by most of the metrics one can find in the literature is that they require the models of the stochastic processes. Hence, if one wants to compare stochastic processes whose internal structures are completely unknown, all these metrics turn out to be useless. In Chapter 4, we present an approach to compare stochastic processes that do not rely on the knowledge of their internal structures. The idea is to use reinforcement learning, a branch of machine learning known for its efficiency when the model or the models are not completely known. The next chapter is a description of the reinforcement learning problem and its solution methods.

Chapter 3

Reinforcement Learning (RL)

3.1 Introduction

One of the main results of this thesis is to reformulate the problem of comparing stochastic systems into a problem a RL problem. RL is a relatively new scientific field which became one of the most active research areas in machine learning and artificial intelligence. It has developed strong mathematical foundations and impressive applications in robotics, control theory, simulation, etc. This chapter is a survey of the field. The first part is an introduction to the basic elements of RL including the Markov decision process (MDP) framework. The rest of the chapter is a survey of the most important techniques to solve MDPs. A reader familiar with RL notions and algorithms can safely skip this chapter.

3.2 Basic Elements

The term reinforcement learning is derived from a theory in psychology having the same name [49]. In computer science, it designates a branch of artificial intelligence and machine learning which is particularly suited for prediction and control in stochastic environments.

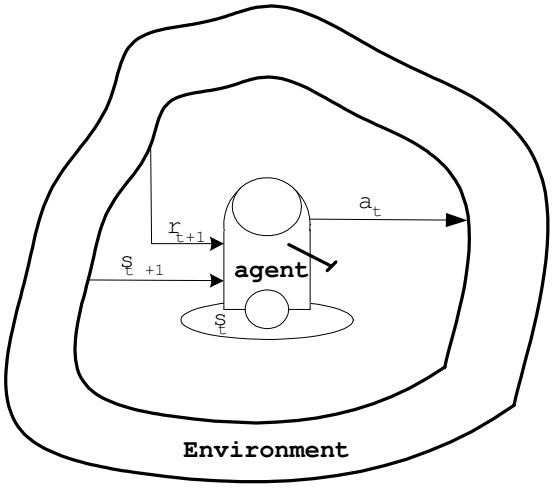


Figure 3.1: The agent-environment interaction

3.2.1 Agent-Environment Interaction

Reinforcement Learning (RL) consists in learning from interaction to achieve a goal. The entity in charge of learning is called the *RL agent* or simply, the *agent*. This agent is in constant interaction with a dynamic environment. The environment designates everything outside the agent. The protocol of interaction between the agent and the environment is depicted in Figure 3.1. At each time step t, depending on its current state s_t, the agent selects an action a_t and behaves accordingly. The environment responds to this move by issuing a reward value r_{t+1} and specifying the new state of the agent s_{t+1}. Generally, the reward value indicates to what extent the goal has been approached or achieved. Hence, the interaction between the agent and the environment starting from time step t yields a sequence of the form : $s_t\ a_t \rightarrow s_{t+1}\ r_{t+1}\ a_{t+1} \rightarrow s_{t+2}\ r_{t+2}\ a_{t+2}\ \ldots \rightarrow s_{t+n}\ r_{t+n}\ a_{t+n}$.

Example 3.1. *Consider a robot navigating in a room and trying to find the exit door. The robot can select to move in any direction. Hence, the set of actions can be the set of moving directions (e.g. left, right, forward, backward). The set of states can be the set of possible coordinates in the room. The reward can be* 0 *for all actions except the ones which end up in the goal state, that is, the exit door, which yields a reward of 1. In this simple example, the boundary between the agent and the environment, unlike one could think, does not correspond to the physical boundary of the robot. Actually, it is much closer to the agent than that. The RL agent is the decision-making entity of the robot. All the other parts including sensing hardware are considered part of the environment. More generally, the boundary between the agent and the environment is*

drawn according to what the agent can change arbitrarily. That is, anything that falls beyond the complete control of the agent is considered part of its environment.

Reinforcement learning is a goal-oriented task. The objective of the agent is then to maximize its cumulative reward, called also the *expected return*. In its simplest form, the return after a time step t, R_t, is the sum of the sequence of obtained rewards. That is,

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_{t+n}. \tag{3.1}$$

This simple way of computing the cumulative reward requires the knowledge of the number n of steps to be performed. For example, in the robot of Example 3.1, one needs to know the number of moves the robot will undertake before starting the episode. More interesting forms of returns will be discussed later.

3.2.2 Markov Property

At each time step, the agent selects an action to perform based on its current state. Hence, the state representation in a RL task is an important issue One could think that immediate sensations are all that the agent needs for decision-making. Unfortunately, in many problems, more information is needed. In particular, the agent may need information about the past history. For example, suppose that the robot of Example 3.1 has a variable speed. Then the immediate sensation at a given moment which is the current position in the room is not sufficient to make a good decision. Knowledge about the speed with which it is moving is required. In light of this, a good state representation is a one that retains relevant information about immediate and past sensations in a compact and summarized form. The crucial point is that all that is needed to make a decision in the future should be retained in the current state. Such a state representation is said to have the *Markov property*. A good example of a Markov state representation is a checkers position: everything the player needs to know in order to make the next move is summarized in the current position.

To make a formal illustration of the Markov property in the context of RL, consider a typical step of interaction between the agent and its environment. Without assuming the Markov property, the probability that, at time step $t+1$, the agent finds itself in state s' with a reward r is:

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\}$$

where $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$ represent the complete past history of the agent. By assuming the Markov property, this probability is the same as:

$$Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}.$$

3.2.3 Markov Decision Process (MDP)

A Markov Decision Process (MDP) [5, 43, 61, 62, 63] is a mathematical framework to model RL tasks satisfying the Markov property. More precisely, an MDP is defined as follows.

Definition 3.2. *An MDP is a tuple (S, A, T, R) where:*

- *S is a set of states.*
- *A is a set of actions.*
- *$T : S \times A \to \Pi(S)$ is a transition function which maps each state-action pair to a probability distribution over the set S. We write $T_{s \to s'}(a)$ for the probability to make a transition to state s' from s via action a.*
- *$R : S \times A \to \mathbb{R}$ is the reward function which maps each state-action pair (s, a) to a real number representing the average immediate reward obtained by an agent when it performs action a from state s. We write R^a_s for $R(s, a)$.*

Sometimes the reward function is defined differently. Indeed, it can be defined as $R : S \times A \times S \to \mathbb{R}$ (that is, by specifying the target state as well). In that case, $R^a_{ss'}$ refers to the reward resulting from running action a on state s and ending in state s'. Since no confusion can arise, in the rest of the thesis, both notations are used.

Example 3.3. *Consider the simple Gridworld with obstacles of Figure 3.2. The agent starts from state 1 (S) and tries to reach the goal state (G). At each state, there are 4 possible actions $A = \{l, r, u, d\}$ for the 4 possible directions left, right, up, and down. These actions do not cause transitions in the corresponding directions deterministically. With probability 0.8 the selected action causes the corresponding transition, with probability 0 it causes a move to the opposite direction (never happens), and with probabilities 0.1 a move in one of the two other directions. For instance, if the agent is in state 7 and tries action right, then with probability 0.8 it makes a transition to state 8, with probability 0.1 it makes a transition to state 11, and with probability 0.1 it makes a transition to state 2. Hence, $T_{7 \to 8}(r) = 0.8, T_{7 \to 11}(r) = 0.1, T_{7 \to 2}(r) = 0.1$. Transitions*

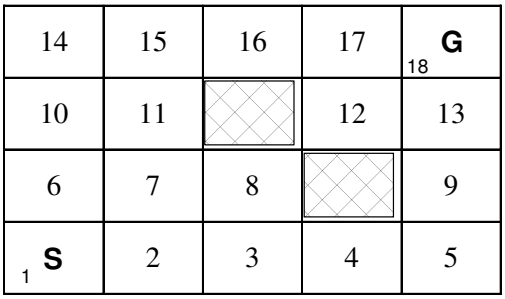


Figure 3.2: A Gridworld with obstacles

leading outside the grid or to an obstacle location are blocked, in which case the agent remains where it is. The immediate reward is 0 *for all state-action pairs except for the pairs that lead to the goal state, where the immediate reward is* 1*. Once the goal state is reached, the task is reinitialized to the initial state. For small and simple RL problems such as this one, it is very helpful to represent the corresponding MDP as a graph. The nodes of the graph represent the states, the edges represent the transitions. These are labelled with the action, the corresponding probability (between brackets) and the immediate reward. Figure 3.3 shows part of the Gridworld MDP.*

3.2.4 Finite Horizon and Infinite Horizon

Equation (3.1) states that the return R_t is simply the sum of the sequence of received immediate rewards $r_{t+1}, r_{t+2}, \ldots, r_{t+n}$. This approach is convenient when one wants to compute the return expected from the next n steps. This framework is called finite horizon. The expected return can then be formulated as:

$$E\Big(\sum_{k=0}^{n} r_{t+k+1}\Big). \tag{3.2}$$

This framework is not very useful in practice since it is rare that an appropriate n is known exactly. Instead, one may prefer a more mathematically attractive framework called infinite-horizon discounted model. In this model, the sequence of rewards is discounted as follows:

$$\begin{aligned} R_t &= r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} \ldots \\ E(R_t) &= E\Big(\sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}\Big) \end{aligned} \tag{3.3}$$

where $0 \leq \gamma \leq 1$ is called the *discount factor*. This approach to compute the return gives less weight to rewards that happen later in the future. By varying the discount

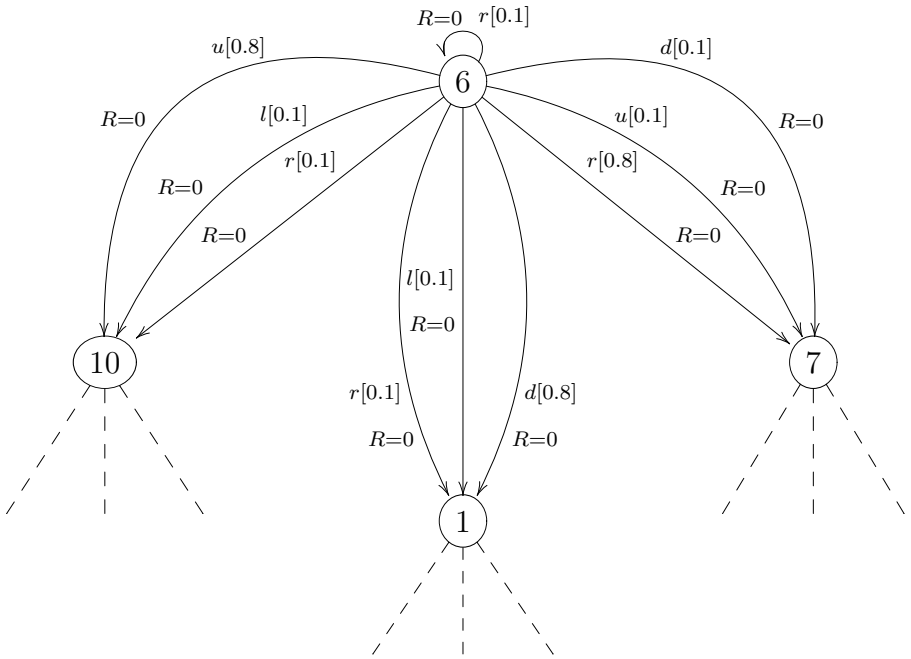


Figure 3.3: A graphical representation of part of the Gridworld MDP.

factor γ between 0 and 1, one can control the rate at which the future gets discounted. The extreme case of $\gamma = 0$ means that the agent is "myopic", concerned only by the immediate reward. The other extreme case, $\gamma = 1$, is equivalent to the non-discounted return of Equation 3.2.

The return can be computed by a third approach called the *average reward* [36]. It is an undiscounted optimality framework where the rewards obtained starting from a state are summed up without discount and then averaged over the different episodes. In this thesis, we do not consider this approach.

3.2.5 Policy

One important role of the agent is to decide which action to take at each state. The agent may prefer some action over another for a variety of reasons. The strategy that the agent uses to select actions is called a *policy*. A policy π is defined as a mapping between states and probability distributions over the set of actions ($\pi : S \rightarrow \Pi(A)$). We write $\pi(s, a)$ for the probability that the agent selects action a in state s. The policy of taking each time the action with the maximum apparent retrun is called the *greedy* policy.

By definition, a policy is stochastic since it maps each state to a probability distribution over actions. However, a policy is *deterministic* if it maps each state to a single action with probability 1.

Policies as we have defined them are called *stationary* policies, because the choice of action depends only on the current state s and is totally independent of the time step t. A *non-stationary* policy is a policy which is dependent on the time step. It is a sequence $(\pi_1, \pi_2, \pi_3, \ldots)$ of mappings between states and probability distributions over actions, one for each time step. This kind of policy is suited for the finite-horizon framework. Indeed, in this framework, the way the agent chooses its actions on the first step is generally different than the way it chooses them on the last step of its lifetime. By way of example, in the last step, the agent would prefer the action with the maximum immediate reward. However, if a certain number of steps remain in its lifetime, then it is better to select an action which maximizes the return over the remaining steps. In the infinite-horizon framework, a non-stationary policy is not necessary, since the agent has a constant expected amount of time remaining; so there is no reason to change the policy at each time step. A stationary policy is more appropriate. Most of the theory on RL focuses on the infinite-horizon discounted model and stationary policies. In this thesis, we assume these cases too.

The goal of an RL task is to maximize the expected return. This consists in finding a policy that guarantees a maximum expected return for each state. This policy is called the *optimal policy*, denoted π^*. Solving an RL task or an MDP consists in finding its optimal policy.

3.2.6 Value Function

The expected return, $E(R_t)$, is the cumulative reward the agent is expected to obtain after time step t. This return depends on two parameters: the agent's current state and the agent's policy. The expected return is then called the *state-value* function and is denoted $V^\pi(s)$. For a discounted future, the state-value function for state s and policy π is defined as:

$$V^\pi(s) = E_\pi\{R_t \,|\, s_t = s\} = E_\pi\Big\{ \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \,|\, s_t = s \Big\}. \tag{3.4}$$

where E_π is the expected value given the policy π. Another interesting and, as we will see later, very useful function is the *action-value* function, denoted $Q^\pi(s, a)$, which maps each state-action pair to a value given a policy π. $Q^\pi(s, a)$ represents the expected

return when taking action a in state s and then following policy π :

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \,|\, s_t = s, a_t = a\} = E_{\pi}\Big\{\sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} \,|\, s_t = s, a_t = a\Big\}. \tag{3.5}$$

For small problems in which the MDP model (transition and reward functions) is completely known, there exist techniques to compute exactly the state-value and action-value functions. These techniques use the following recursive property of the value functions. Let $\mathcal{M} = (S, A, T, R)$ be a finite MDP, s a state in S and π a policy. The state-value function can be expressed as :

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} T_{s \to s'}(a) \big[R^a_{ss'} + \gamma \, V^{\pi}(s')\big]. \tag{3.6}$$

This equation is known as the *Bellman equation* [3] for V^{π}. It expresses the value of a state using the values of its successor states. The first sum ($\sum_{a \in A} \pi(s,a)$) is over the set of actions A averaged by the probability that the policy π selecting each action. The second sum ($\sum_{s' \in S} T_{s \to s'}(a)$) is over the possible next states averaged by the corresponding probability $T_{s \to s'}(a)$. The remaining of the equation ($\big[R^a_{ss'} + \gamma \, V^{\pi}(s')\big]$) represents the value of the return with action a selected and next state s' taken. The action-value function $Q^{\pi}(s,a)$ can be expressed using the state-value function as follows:

$$Q^{\pi}(s,a) = \sum_{s' \in S} T_{s \to s'}(a) \big[R^a_{ss'} + \gamma \, V^{\pi}(s')\big]. \tag{3.7}$$

Using the Bellman equation to compute the value functions is a computation-intensive and demanding task since it performs an exhaustive search by averaging over all actions and all possibilities of transitions. Hence, it relies on a complete and accurate knowledge of the dynamics of the MDP which is rarely available in practice. In situations where the model of the MDP is not completely known, it is very difficult to compute exactly the state and action-value functions. In such situations, one can only estimate the functions via interaction with the environment. Estimating the value functions through interaction is the core element of RL. In the following sections, we present several known algorithms for doing it.

3.2.7 Optimality

Computing or estimating the value functions V and Q is a step towards achieving the main goal of RL, that is, finding the policy that maximizes the value functions for each state. This policy is called the *optimal policy*, is denoted π^* and satisfies the following condition:

$$V^{\pi^*}(s) \geq V^{\pi}(s) \quad \forall \pi \;\; \forall s \in S.$$

Actually, there could be several optimal policies for a problem. The value function corresponding to the optimal policies is called the optimal value function and is noted also V^*. Similarly, one can define the optimal action-value function Q^*. For a state-action pair (s, a), this function returns the expected value of selecting action a from state s and then following an optimal policy. That is,

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a).$$

Most of the RL theory is about solving MDPs, that is, identifying the optimal value functions and the corresponding optimal policies. The rest of this chapter is a survey of the most important trends and algorithms for achieving it.

3.3 Dynamic Programming

The methods to solve MDPs fall into two classes: *planning* methods and *learning* methods. Planning are methods that require a model of the environment. By a model, we mean anything that an agent can use to predict how the environment will respond to its actions. Hence, planning refers to any technique that takes a model as input and tries to figure out the optimal value function and the optimal policy. On the other hand, learning methods can be used without a model of the environment. The optimal policy is learned by interacting with the environment, that is, by using real experience.

Dynamic programming is a planning method. It refers to a "collection of algorithms that can be used to compute optimal policies given a perfect model of the environment like an MDP". Since it relies on a perfect model of the environment, dynamic programming is rarely useful in practice. However, it has been extensively studied in the literature because it is very attractive for the theoretical standpoint. In particular, it includes ideas that are essential to understand the other planning as well as learning methods. The pivotal equation of all dynamic programming algorithms is the Bellman equation (3.6) which updates the value of a state based on values of all possible successor states. This is known as *full backup*. On the other hand, *sample backup* consists in updating state values using only one (sampled) possibility of transition. This type of backup is mainly used with learning methods. The first algorithm we present is an algorithm to compute the value function V^{π} for an arbitrary policy π.

Input π
Initialize $V(s) = 0$, for all s
Repeat
  $\delta = 0$
  For each state $s \in S$
    $v = V(s)$
    $V(s) = \sum_{a \in A} \pi(s,a) \left(R_s^a + \sum_{s' \in S} T_{s \to s'}(a) \, \gamma \, V(s') \right)$
    $\delta = \max(\delta, \mid v - V(s) \mid)$
Until $\delta < \epsilon$
Return V.

Figure 3.4: Iterative policy evaluation

3.3.1 Iterative Policy Evaluation

Iterative policy evaluation is an algorithm which takes as input a policy π and returns the value function V^π. The idea is to start form an arbitrary initial value function V_0 (for example $V_0(s) = 0$, for all s) then to update it using the Bellman equation to obtain V_1. Then, the same mechanism is repeated for V_1, and so on. More formally, at each iteration k, the value function V_k is updated to obtain a new value function V_{k+1} according to the following update rule:

$$V_{k+1}(s) = \sum_{a \in A} \pi(s,a) \left(R_s^a + \sum_{s' \in S} T_{s \to s'}(a) \, \gamma \, V_k(s') \right).$$

It can be shown that the sequence $V_0, V_1, V_2, \ldots, V_k, \ldots$ converges to the exact value function V^π as $k \to \infty$. However, for practical considerations, waiting until the algorithm converges to the exact value function V^π is not a computationally good idea. Instead, the iterations can be stopped relatively long before that. The problem is that it is not obvious to decide when to stop the iterations. One possible termination criterion is to wait until the maximum difference between two successive value functions $V_k(s)$ and $V_{k+1}(s)$ over all states is less than some threshold ϵ:

$$\max_{s \in S} |V_k(s) - V_{k+1}(s)| \leq \epsilon.$$

The procedural form of iterative policy evaluation algorithm is given in Figure 3.4.

Example 3.4. *Consider the MDP of Figure 3.5 with actions set* $A = \{a, b, c\}$. *Let* π *be a policy such that, for each state* s, $\pi(s,a) = 0.5$ *and* $\pi(s,b) = 0.5$. *Let us compute the value of policy* π *using the iterative policy iteration with* $\gamma = 0.8$ *and* $\epsilon = 0.01$*:*

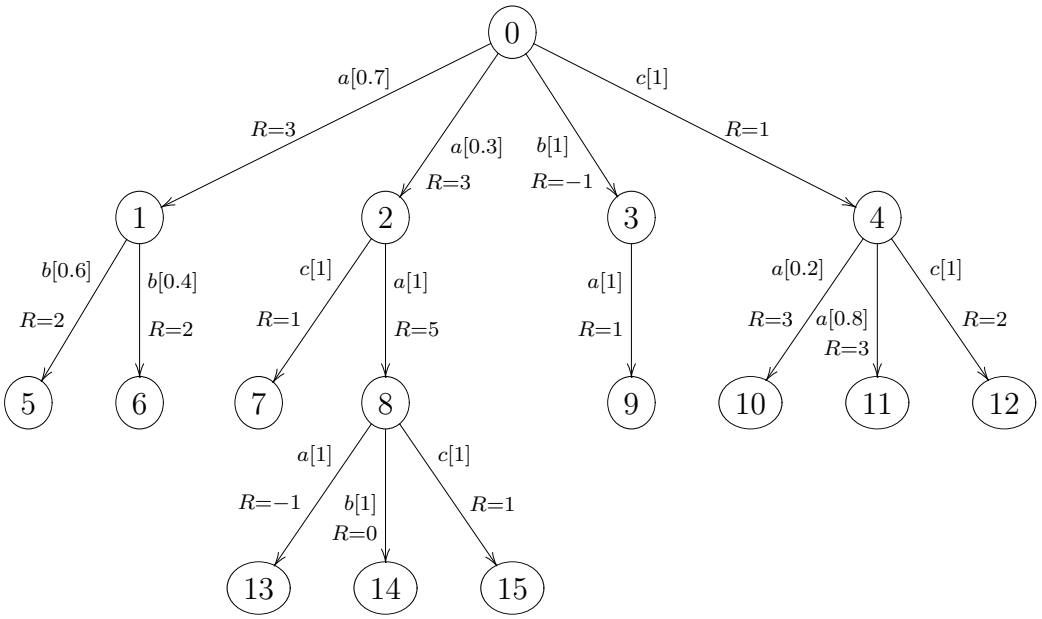


Figure 3.5: A simple MDP.

$$V_0(s) = 0 \;\; \forall\, s \in S$$

$$\begin{aligned} V_1(0) &= \pi(0,a)\ (R_0^a + T_{0\to 1}(a)\ \gamma\ V_0(1) + T_{0\to 2}(a)\ \gamma\ V_0(2)) \\ &\qquad + \pi(0,b)\ (R_0^b + T_{0\to 3}(b)\ \gamma\ V_0(3)) \\ &= 0.5\ \ (3 + 0.7\ \ 0.8\ \ 0 + 0.3\ \ 0.8\ \ 0) + 0.5\ \ (-1 + 1\ \ 0.8\ \ 0) = 1 \\ V_1(1) &= 0.5\ \ (2 + 0.6\ \ 0.8\ \ 0 + 0.4\ \ 0.8\ \ 0) = 1 \\ V_1(2) &= 0.5\ \ (5 + 1\ \ 0.8\ \ 0) = 2.5 \\ V_1(3) &= 0.5\ \ (1 + 1\ \ 0.8\ \ 0) = 0.5 \\ V_1(4) &= 0.5\ \ (3 + 0.2\ \ 0.8\ \ 0 + 0.8\ \ 0.8\ \ 0) = 1.5 \\ V_1(8) &= 0.5\ \ (-1 + 1\ \ 0.8\ \ 0) + 0.5\ \ (0 + 1\ \ 0.8\ \ 0) = -0.5 \end{aligned}$$

$$\begin{aligned} V_2(0) &= 0.5\ \ (3 + 0.7\ \ 0.8\ \ 1 + 0.3\ \ 0.8\ \ 2.5) + 0.5\ \ (-1 + 1\ \ 0.8\ \ 0.5) = 1.705 \\ V_2(2) &= 0.5\ \ (5 + 1\ \ 0.8\ \ (-0.5)) = 2.3 \end{aligned}$$

$$V_3(0) = 0.5(3 + 0.7\ \ 0.8\ \ 1 + 0.3\ \ 0.8\ \ 2.3) + 0.5\ \ (-1 + 1\ \ 0.8\ \ 0.5) = 1.756.$$

The exact state values according to policy π are then: $V^\pi(0) = 1.756$, $V^\pi(1) = 1$, $V^\pi(2) = 2.3$, $V^\pi(4) = 1.5$, $V^\pi(8) = -0.5$. The value of the remaining states is 0.

3.3.2 Policy Iteration

Policy iteration is a control algorithm, that is, it focuses on looking for the optimal policy. If follows a process called *policy improvement*. This process consists in constructing, at each iteration, a new and better policy based on an original policy. Consider a deterministic policy π which selects at state s action a ($\pi(s) = a$). If selecting action a'

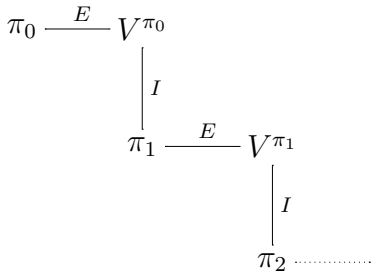


Figure 3.6: Policy evaluation and policy improvement

from s instead of a produces a better result, then, the policy π' which behaves exactly as π except on state s, where $\pi'(s) = a'$, will yield a better value function:

$$V^{\pi'}(s) \geq V^{\pi}(s) \quad \forall s \in S.$$

Moreover, if this improvement is performed on all states at each iteration, then a faster rate of improvement can be achieved. The approach of policy iteration algorithm is to iterate over this process so as to obtain at each iteration a better policy. Between two successive improvements, the algorithm evaluates the current policy. This evaluation consists in applying the policy evaluation algorithm of the previous section. Hence the policy iteration algorithm can be graphically represented as moving in a two dimensions space: one evaluation step, then one improvement step, and so on (Figure 3.6).

The complete algorithm is given in a procedural form in Figure 3.7. The expression $arg\max_{a \in A}(f)$ returns the action that maximizes f.

3.3.3 Value Iteration

Policy iteration has a serious drawback in that each iteration involves a complete run of policy evaluation algorithm. The latter, described in Section 3.3.1, consists of several sweeps over the set of states. To improve this, one can alleviate the iterations of the algorithm by stopping the policy evaluation before reaching convergence. Hence, the policy evaluation step can be stopped after a certain number of sweeps over states. In particular, the extreme case of stopping policy evaluation after one sweep is of special interest. The resulting algorithm is called *value iteration*. The update rule of value iteration combines both policy improvement and policy evaluation steps:

$$V_{k+1}(s) = \max_{a \in A} \left(R_s^a + \sum_{s' \in S} T_{s \to s'}(a)\, \gamma\, V_k(s') \right).$$

1. Select a random policy π
   Initialize V arbitrarily (e.g. $V(s) = 0$)
2. Repeat
   $\delta = 0$
   For each state $s \in S$
      $v = V(s)$
      $V(s) = \sum_{a \in A} \pi(s,a) \left(R_s^a + \sum_{s' \in S} T_{s \to s'}(a) \, \gamma \, V(s') \right)$
      $\delta = \max(\delta, \mid v - V(s) \mid)$
   Until $\delta < \epsilon$
3. policyStable = true
   For each state $s \in S$
      $b = \pi(s)$
      $\pi(s) = arg \max_{a \in A} \left(R_s^a + \sum_{s' \in S} T_{s \to s'}(a) \, \gamma \, V(s') \right)$
      if $b \neq \pi(s)$ then policyStable = false
   if policyStable then stop
      else goto 2
4. Return V and π.

Figure 3.7: Policy iteration

As of the termination criterion, Williams and Baird [64] showed that if the maximum difference between two successive values functions is less than ϵ, that is,

$$\max_{s \in S} \mid V_k(s) - V_{k+1}(s) \mid \leq \epsilon$$

then the value function of the greedy policy (the policy which consists in taking at each state the action with the maximum value using the current estimate of the value function) differs from the value function of the optimal policy by no more than $\frac{2\epsilon\gamma}{1-\gamma}$. In the light of this result, one can tune the algorithm to produce a policy with a desired closeness to the optimal policy. The complete value iteration algorithm is illustrated in Figure 3.8.

3.4 Exploration versus Exploitation

Unlike planning methods, learning methods do not need a model in input to solve an MDP. They directly learn the optimal value function and policy by interacting with the environment. In a typical learning algorithm, the agent selects at each step a sample action that he thinks will yield the best return on the long-run. If the agent maintains,

Initialize V arbitrarily (e.g. $V(s) = 0$)
Repeat
  $\delta = 0$
  For each state $s \in S$
    $v = V(s)$
    $V(s) = \max_{a \in A} \left(R_s^a + \sum_{s' \in S} T_{s \to s'}(a)\, \gamma\, V(s') \right)$
    $\delta = \max(\delta, \mid v - V(s) \mid)$
Until $\delta < \epsilon$
For each state $s \in S$, $\pi(s) = arg \max_{a \in A} \left(R_s^a + \sum_{s' \in S} T_{s \to s'}(a)\, \gamma\, V(s') \right)$
Return V and π.

Figure 3.8: Value iteration

for each state, an estimate of the action-value for each action ($Q(s, a)$), then it may think that choosing the action with the highest action-value, called the greedy action, will guarantee the highest return at the long-run. This is, most of the time, not the case. Indeed, recall that the return following an action is the cumulative reward and that the action-values maintained at each step are only estimates of that return. Hence, an action with an apparently low action-value may get an important reward after some steps. Similarly, an apparently greedy action may turn out to not be that good after some steps in the future. By choosing a greedy action, the agent is said to *exploit* its current knowledge to maximize the return on the long-run, whereas by choosing one of the other non-greedy actions, the agent is said to *explore* its environment to discover better actions that it can then exploit. Since by a single action selection it is not possible to both exploit and explore, the agent should tradeoff between exploitation and exploration. This tradeoff is a crucial aspect of RL theory. The methods to select actions from states so as to guarantee a good compromise between exploration and exploitation are called *action selection* methods. The most known action selection methods are *ϵ-greedy* [61] and *Softmax* [6].

3.4.1 ϵ-greedy

At each state, there is always one or several actions with a maximum estimated action-value: the greedy actions. Selecting at any state the greedy action prevents any exploration of the other possible actions. This can lead to very poor value functions since the algorithm will converge quickly to a sub-optimal policy. One interesting alternative to this 100% greedy approach is to behave most of the time greedily and once in a while

select randomly among the non-greedy actions. Hence, using a suitable ϵ ($0 < \epsilon < 1$), select the greedy action with probability $1-\epsilon$ and with a probability ϵ selects randomly among all actions, including the greedy action. More precisely, in presence of a set of actions A, the greedy action is selected with probability $1-\epsilon+\frac{\epsilon}{|A|}$ and any other action is selected with probability $\frac{\epsilon}{|A|}$. This action selection method is called ϵ-greedy. The mechanism of letting all actions possible from a state is called *softening* and it is mainly used for policies. It is important to mention that, generally, better convergence results are obtained by letting the ϵ value decrease progressively over time.

3.4.2 Softmax

One important downside of the ϵ-greedy method is that when it behaves non-greedily, it will select an action randomly from the set of actions regardless of their action-values. In situations where there are strong differences between actions, the convergence to the optimal policy may be strongly affected. To overcome this problem, one possible solution is to weight the probability of selecting each action according to its current estimated action-value. This method is called Softmax action selection. Several mathematical formulas can be used to rank the actions according to their $Q-$values. The most popular is the Gibbs distribution, which chooses each action a among n actions with probability:

$$\frac{e^{Q(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q(s,b)/\tau}}$$

where τ is a positive number called the temperature. The temperature τ controls to which extent a difference in action-values will transpose into difference in probabilities. The bigger is τ, the more equiprobable the actions will be. Hence, a low temperature τ will yield to accentuated probabilities for actions with different action-values.

It is not clear which one of ϵ-greedy and Softmax is better. Most of the time this depends mainly on the nature of the task.

3.5 Monte Carlo

The *Monte Carlo* (MC) expression refers typically to estimation techniques involving an important random component. In the context of RL, it refers to methods for solving MDPs by averaging complete sample returns [38].

Initialize V arbitrarily (e.g. $V(s) = 0$)
$Returns(s)$ = empty list, for each $s \in S$
Repeat m times (m = the number of episodes)
    Generate an episode using π
    For each state s in the episode
        R = return following s
        Append R to $Returns(s)$
        $V(s) = Average(Returns(s))$
Return V.

Figure 3.9: First-visit MC policy evaluation

With Monte Carlo, the RL task is assumed to break up into episodes. Hence, whatever actions are selected, an episode terminates after a finite number of steps. Monte Carlo is a learning method whose main idea is to maintain the returns following each state, then the value is obtained by averaging all experienced returns. One important aspect of Monte Carlo is that the agent must wait until the end of the episode before adding the return and updating the values of states. As for dynamic programming, we start by presenting a Monte Carlo algorithm for policy evaluation.

3.5.1 Monte Carlo Policy Evaluation

Recall that the policy evaluation algorithm takes as input a policy π and tries to estimate the corresponding value function V^π. There are two Monte Carlo methods to do it: *every-visit MC* and *first-visit MC* [2, 48]. The every-visit method averages the (discounted) returns of all visits to a given state s across all episodes. The first-visit, however, averages the returns of only the first visits in each episode. The algorithm of the first-visit MC policy evaluation is illustrated in a procedural form in Figure 3.9. Note that generating an episode using policy π means that the action selection at each state is guided by the policy π.

3.5.2 Monte Carlo Control

The policy evaluation algorithm above proceeds by estimating state-value function (V^π). If the objective is to approximate the optimal value and policy, one needs to work with action-value function (Q). Indeed, knowing the values of actions separately is

Initialize $Q(s,a)$ arbitrarily (e.g. $Q(s,a)=0$)
Select a random policy π
$Returns(s,a)$ = empty list
Repeat m times (m = the number of episodes)
    Generate an episode using π and exploring start
    For each pair s,a in the episode
        R = return following s,a
        Append R to $Returns(s,a)$
        $Q(s,a) = Average(Returns(s,a))$
    For each s
        $\pi(s) = arg\max_{a \in A} Q(s,a)$
Return V and π.

Figure 3.10: MC exploring start

essential to suggest better policies. A simple MC control algorithm might have the same structure as the policy iteration DP algorithm presented in Section 3.3.2. That is, it starts from an arbitrary policy π_0 then performs a complete evaluation to obtain V^{π_0}, then improves the policy to obtain policy π_1, then evaluate it, and so on. This simple algorithm suffers from two drawbacks. The first is that each iteration involves a complete policy evaluation. The second is that nothing guarantees that exploration will be maintained. The first problem can be solved in the same spirit as value iteration (Section 3.3.3). The idea is to halt the evaluation of each policy after only one episode. One simple way to solve the second problem is to let each episode start from a different state-action pair, that is, every pair has a non-zero probability to be selected in the beginning of an episode. This trick is called *exploring start*. Applying these two ideas gives rise to a MC control algorithm called MC exploring start (Figure 3.10).

3.5.3 On-policy and Off-policy Methods

The exploring start condition of the previous algorithm assumes that each episode starts by a different state-action pair. This assumption is not always applicable and in situations where it is applicable, it is not always an efficient solution to maintain exploration. In this section, we investigate two more efficient and attractive techniques to guarantee continual exploration called *on-policy* and *off-policy* methods[1].

[1]On-policy and off-policy methods are not specific to Monte Carlo approach. They are used with other kinds of algorithms such as temporal difference.

Initialize $Q(s,a)$ arbitrarily (e.g. $Q(s,a)=0$)
Select a random ϵ-soft policy π
$Returns(s,a)$ = empty list
Repeat m times (m = the number of episodes)
    Generate an episode using π
    For each pair s,a in the episode
        R = return following s,a
        Append R to $Returns(s,a)$
        $Q(s,a) = Average(Returns(s,a))$
    For each state s
        $a^* = arg\max_{a\in A} Q(s,a)$
        For each action a

$$\pi(s,a) = \begin{cases} 1-\epsilon+\frac{\epsilon}{|A(s)|} & \text{if } a=a^* \\ \frac{\epsilon}{|A(s)|} & \text{if } a\neq a^* \end{cases}$$

Return V and π.

Figure 3.11: ϵ-soft on-policy MC

The idea of on-policy methods is to use the same policy for making decision and for policy improvement. In order to ensure a continual exploration, this same policy is generally soft, that is, for each state, it keeps a non-zero probability to select any action ($\forall s \in S, \forall a \in A, \pi(s,a) > 0$). In particular, for a given ϵ ($0 < \epsilon < 1$), a soft policy which guarantees that each action has a probability at least $\frac{\epsilon}{|A|}$ to be selected is called *ϵ-soft* policy ($\forall\, s \in S, \forall\, a \in A, \pi(s,a) \geq \frac{\epsilon}{|A|}$). The sample algorithm we present as an example of on-policy MC is called ϵ-soft on-policy MC algorithm. This algorithm starts from an ϵ-soft policy and attempts to improve it gradually by making it move towards an ϵ-greedy policy. Since an ϵ-greedy is by nature a soft policy, exploration will always be guaranteed. The ϵ-soft on-policy MC algorithm is given in a procedural form in Figure 3.11.

Off-policy methods, on the other hand, follow a different approach. They behave according to one policy while estimating and improving another policy. Hence exploration is maintained through the first policy while estimation of the optimal policy is guaranteed via the second policy. A very famous algorithm with an off-policy approach, called Q-learning, is presented in the next section.

Initialize V arbitrarily (e.g. $V(s) = 0$)
π = policy to be evaluated
Repeat for each episode
    Initialize s
    Repeat for each step in the episode
        $a = \pi(s)$
        Take a, observe r', s'
        $V(s) = V(s) + \alpha\,[r' + \gamma\,V(s') - V(s)]$
        $s = s'$
Return V.

Figure 3.12: $TD(0)$ policy evaluation

3.6 Temporal Difference

Temporal difference [46, 19] is a set of learning algorithms combining the ideas of dynamic programming and Monte Carlo. As dynamic programming, temporal difference (TD) updates the estimates based on the estimates of successor states (backup). As Monte Carlo, temporal difference can learn directly from raw experience by simply interacting with the environment. Temporal difference differs from Monte Carlo in that they it does not wait until the end of the episode to get the final return and to update the value function. Instead, it can choose to perform the update after one step, two steps, etc. Of particular interest is the case where the updates are performed after only one step. The first three algorithms we present in this section, namely Temporal Difference (0) ($TD(0)$), Sarsa, and Q-learning, belong to this category.

3.6.1 $TD(0)$

The simplest policy evaluation TD algorithm is called $TD(0)$ [52]. It uses the following equation to update its value function.

$$V(s_t) := V(s_t) + \alpha\,[r_{t+1} + \gamma\,V(s_{t+1}) - V(s_t)] \tag{3.8}$$

where $0 < \alpha < 1$ is called the *step-size* parameter or the *learning rate* factor. A careful reading of this equation shows how $TD(0)$ combines dynamic programming and Monte Carlo ideas. Indeed, on one hand, the rule uses an estimation of the next state $V(s_{t+1})$ to update $V(s_t)$. On the other hand, it uses a sample return, r_{t+1}, instead of the expected reward ($\sum_{a \in A} \pi(s,a) R^a_s$). The amount between brackets in Equation (3.8) represents

Initialize $Q(s,a)$ arbitrarily (e.g. $Q(s,a)=0$)
Repeat for each episode
    Initialize s
    Choose action a from s using an action selection method (e.g. ϵ-greedy)
    Repeat for each step in the episode
        Take action a, observe r', s'
        Choose action a' from s' using an action selection method (e.g. ϵ-greedy)
        $Q(s,a) = Q(s,a) + \alpha\,[r' + \gamma\, Q(s',a') - Q(s,a)]$.
        $s = s'$
        $a = a'$
For each state $s \in S$, $\pi(s) = arg \max_{a \in A} Q(s,a)$
Return V and π.

Figure 3.13: Sarsa

the error experienced during the current step. Hence, the value of the current state s_t is updated by adding this error to the old value weighted by the step size parameter α. The step size parameter controls the rate by which the learning happens. Very often, the step size parameter should not be constant. Indeed, it is more convenient to consider a variable α. A well-known result in stochastic approximation theory states that, in order to obtain convergence, two conditions are required for α:

$$\sum_{k=1}^{\infty} \alpha_k(s,a) = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \alpha_k^2(s,a) < \infty$$

where $\alpha_k(s,a)$ is the value of α when action a is selected from state s for the k^{th} time. Figure 3.12 illustrates $TD(0)$ algorithm for estimating V^{π}.

3.6.2 Sarsa

$TD(0)$ is a policy evaluation algorithm that learns values of states $V^{\pi}(s)\ \forall\, s \in S$. As usual, if one wants to find the optimal policy, the best approach is to concentrate on learning action-values. The Sarsa algorithm [44] uses the same update rule of $TD(0)$ (Equation (3.8)) but applied to state-action pairs :

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\,[r_{t+1} + \gamma\, Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]. \tag{3.9}$$

For the purpose of moving toward the optimal policy, Sarsa algorithm combines at each episode policy improving and policy evaluation. The approach it uses to tradeoff

Initialize $Q(s, a)$ arbitrarily (e.g. $Q(s, a) = 0$)
Repeat for each episode
    Initialize s
    Repeat for each step in the episode
        Choose action a from s using an action selection method (e.g. ϵ-greedy)
        Take action a, observe r, s'
        $Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)\right]$
        s = s'
For each state $s \in S$, $\pi(s) = arg \max_{a \in A} Q(s, a)$
Return V and π.

Figure 3.14: Q-learning

exploration and exploitation is an on-policy method. Hence Sarsa is an on-policy one-step temporal difference control algorithm (Figure 3.13). The on-policy aspect of Sarsa is exhibited through the fact that the same action is used for behavior ($a = a'$) and for updating.

3.6.3 Q-learning

Q-learning [61] is an off-policy one-step TD control algorithm. Separate policies are used for making decisions and for estimating the optimal policy. The action-value function update rule is slightly different from that of Sarsa:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a' \in A} Q(s_{t+1}, a') - Q(s_t, a_t)\right]. \tag{3.10}$$

The main difference is that the Q-learning updates the value of the current state by the action with the maximum estimated value in the next state. A different action may be selected from that next state in the next step. This is the manifestation of the off-policy approach of Q-learning. The complete Q-learning algorithm is given in Figure 3.14.

Q-learning algorithm was introduced by Watkins [61]. Later on, Rummery and Niranjan [44] proposed a modified version of the same algorithm they called *modified Q-learning* that Sutton [55] named Sarsa.

Initialize V arbitrarily (e.g. $V(s) = 0$)
Initialize $e(s) = 0$ for all states
Repeat for each episode
    Initialize s
    Repeat for each step in the episode
        $a = \pi(s)$
        Take action a, observe r, s'
        $\delta = r + \gamma\, V(s') - V(s)$
        $e(s) = e(s) + 1$
        For all states
          $V(s) = V(s) + \alpha\, \delta\, e(s)$
          $e(s) = e(s) + \gamma\, \lambda\, e(s)$
        $s = s'$
Return V and π.

Figure 3.15: $TD(\lambda)$

3.6.4 $TD(\lambda)$

The algorithms presented so far in this section are one-step temporal difference. These are located in one extreme case where the backup is performed after only one step. The other extreme case is when the backup is performed at the end of the episode, which corresponds to Monte Carlo methods. Between these extreme cases, one can consider backups after two steps, three steps, etc. In general, the n-step return, noted $R_t^{(n)}$, is defined as:

$$R_t^{(n)} = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \ldots + \gamma^{n-1}\, r_{t+n} + \gamma^n\, V_t(s_{t+n}).$$

The idea of $TD(\lambda)$ policy evaluation algorithm [61] is to average different n-step returns :$R_t^{(1)}, R_t^{(2)}$, etc. Each n-step return is weighted proportionally to λ^{n-1}, where $0 \leq \lambda \leq 1$. Hence the average return, called λ-return, is defined as:

$$\begin{aligned} R_t^{\lambda} &= (1-\lambda) R_t^{(1)} + (1-\lambda)\, \lambda\, R_t^{(2)} + (1-\lambda)\, \lambda^2\, R_t^{(3)} + \ldots \\ &= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1}\, R_t^{(n)} \end{aligned} \tag{3.11}$$

where $(1-\lambda)$ is a normalization factor which ensures that weights sum up to 1[2].

[2]Recall that $1 + \lambda + \lambda^2 + \lambda^3 + \ldots = \frac{1}{1-\lambda}$.

This way of viewing $TD(\lambda)$ (Equation (3.11)) is called the forward view [22]. Unfortunately, this view is not directly implementable since at each step it needs knowledge about what will happen many steps in the future. Instead, there is another view, called the backward view [26, 52], which is computationally more attractive. In a nutshell, the idea of the backward view is that instead of using future rewards to perform the update, the changes experienced in the current state are propagated backwardly to the previously visited states. For concreteness, an additional memory variable, called the *eligibility trace* and noted $e_t(s)$, is maintained for each state. Intuitively, since the backward view is based on propagating a change backwards, the eligibility trace indicates to which extent a state is eligible for performing an update. The more recently a state has been visited, the more eligible it will be for update. The eligibility traces are initialized to zero and then will be updated as follows:

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s) + 1 & if \quad s = s_t \\ \gamma \lambda e_{t-1}(s) & if \quad s \neq s_t. \end{cases}$$

That is, in each step, the eligibility trace of the visited state s_t will be updated by 1, whereas the eligibility traces of all the remaining states will decay by $\gamma \lambda$. The procedural form of the $TD(\lambda)$ policy evaluation algorithm using eligibility traces is given in Figure 3.15.

Finally, we note that eligibility traces can be used to design control algorithms such as $Q(\lambda)$ [61] and $Sarsa(\lambda)$ [45].

3.7 Integrating Planning and Learning

Direct RL techniques such as Q-learning, Sarsa, and Monte Carlo learn optimal policies without requiring a model of the MDP. The downside of these techniques, however, is that they make an extremely inefficient use of the data they gather by interaction. In applications where real world experience is costly, these techniques may produce poor results. In this section, we still assume that the model of the MDP is not available but we examine algorithms that proceed by learning the model. These techniques, generally, combine planning and learning methods.

Repeat
Select a state s and an action a at random
$s', r = model(s, a)$
$Q(s, a) = Q(s, a) + \alpha \left[r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a) \right]$

Figure 3.16: Main loop of Q-planning

3.7.1 Q-planning

Q-planning is an algorithm which requires an estimated model of the MDP on which it applies Q-learning algorithm. The Q-learning runs on simulated experience (*model*) exactly as it does on real experience. Hence, *model* can be seen as a stochastic function in $S \times A \rightarrow \Pi(S \times \mathbb{R})$ that accepts as input a state s and an action a and returns a new state s' and a reward r. The main loop of Q-planning is illustrated in Figure 3.16. The algorithm is more accurately called random-sample one-step Q-planning. This straightforward method to solve MDPs: first learn the model by exploring the environment and next compute an optimal policy using one of the known algorithms (DP, MC, or TD) has three serious problems. First, it makes an arbitrary separation between the model learning phase and the optimal policy approximation phase. Second, it is not clear how should the model learning phase gather data about the environment; random exploration can be very inefficient. Third, since there is separation in the phases, if some changes occur in the environment while the second phase is underway, the results will not be consistent with the environment.

3.7.2 Dyna-Q

Dyna-Q [53, 54] is an algorithm which exemplifies the integration of planning and learning methods. It shows how apparently distinct methods can be programmed to work together. In particular, it combines model learning, direct RL, and planning methods at each episode. Figure 3.17 shows the Dyna-Q algorithm in a procedural form. Note that the action-value function ($Q(s, a)$) is updated several times in an episode: one time by Q-learning and N times by Q-planning. Hence, both direct RL and planning phases use the same algorithm: Q-learning. Whereas, the model learning proceeds in the background.

Initialize $Q(s,a)$ and $model(s,a)$ arbitrarily
Repeat for each episode
    Initialize s
    Repeat for each step in the episode
        Select action a using ϵ-greedy
        Take action a, observe r, s'
        $Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)\right]$
        Update $model(s,a)$ with s' and r
        Repeat N times:
            s = random state (already observed)
            a = random action (previously taken in s)
            $s', r = model(s,a)$
            $Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)\right]$

Figure 3.17: Dyna-Q

3.8 Conclusion

Among all methods presented in this chapter, TD are today the most used RL techniques due to their simplicity of use. Indeed, most of TD algorithms involve a simple and single equation that can be implemented by small computer programs and they rely only on experience generated from interaction. In particular, Q-learning algorithm introduced by Watkins [61] is considered as the most important breakthrough in RL.

The most attractive aspect of RL is the capability to learn the best strategy of behaving (optimal policy) directly from real experience, that is, without a model of the environment. This makes RL applicable in problems where very little information is known about the environment but where interaction is possible. The problem we are considering in this thesis belongs to this category. The next chapter shows how RL is a suitable and interesting approach to quantify the divergence between stochastic systems.

Chapter 4

Trace Equivalence Divergence through Reinforcement Learning

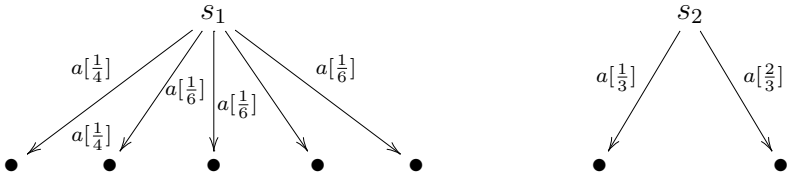
4.1 Introduction

The principal contribution of this thesis is a new approach to estimate the difference between stochastic systems. This chapter introduces the key ideas of this approach. For simplicity and clarity, these ideas will be illustrated using the simplest equivalence notion, namely, trace equivalence. Also, for concreteness, we suppose that we are facing a formal verification problem where we want to check whether an implementation of a system satisfies its pre-established formal specification. The implementation can be a physical device, a program, a protocol, etc. It is noted simply "Impl". The specification refers to a representation of the perfect behavior of the system and is noted simply "Spec". We suppose that the internal structure of both "Impl" and "Spec" are LMPs but the models of these LMPs are completely unknown. That is, we treat them as black-boxes available only for interaction. The objective of this chapter is then to define a "trace equivalence divergence" between "Impl" and "Spec".

4.2 Stochastic Game

The approach we propose is based on RL. The idea is to let the divergence come as a solution of an MDP. This MDP is defined in such a way that the optimal value can be interpreted as the divergence between "Impl" and "Spec" and the optimal policy

as tests that witness that divergence. In this section, we expose our approach in the form of a one-player stochastic game, the player being the personification of the learning algorithm, that is, the RL agent. Since the models of the processes are not available, the player proceeds by interacting with the processes. This interaction consists basically in selecting an action a, running it on both processes and then observe the result. An action that yields different behaviors in "Impl" and "Spec" indicates a possible divergence between them whereas an action that yields the same behavior indicates a possible similarity. Since the player's goal is to detect differences between the two processes, the game should give him low reward when the processes behave the same and high reward when they behave differently. However, we must be careful because the processes are probabilistic and consequently the same 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. Intuitively, high entropy in a given state means several different transitions are possible after an action. For example, consider the two states s_1 and s_2:



We assume that, for action a, both define a probability distribution on next-states ($\sum_{s' \in \mathcal{S}} P_{s_1 \to s'}(a) = \sum_{s' \in \mathcal{S}} P_{s_2 \to s'}(a) = 1$). It is easy to see that s_1 has more entropy than s_2. Indeed,

$$\begin{aligned} H(P^a_{s_1}) &= 2\,\frac{1}{4}\log_2\frac{1}{4} + 3\,\frac{1}{6}\log_2\frac{1}{6} = 2.29\,bits \\ H(P^a_{s_2}) &= \frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3} = 0.918\,bits. \end{aligned}$$

To compensate this uncertainty, we introduce a third process, called "Clone" which is simply a copy of the specification (see Figure 4.1) and we include it in the game. As an action yielding different behaviors in "Impl" and "Spec" indicates a possible divergence between them, an action yielding different behaviors between "Spec" and "Clone" is a manifestation of the entropy of process "Spec". Consequently, the player will get a high reward if "Impl" and "Spec" behave differently for some action, but this reward could be cancelled if "Spec" and "Clone" also do.

These ideas are captured in the following stochastic game, which is graphically represented in Figure 4.2 :

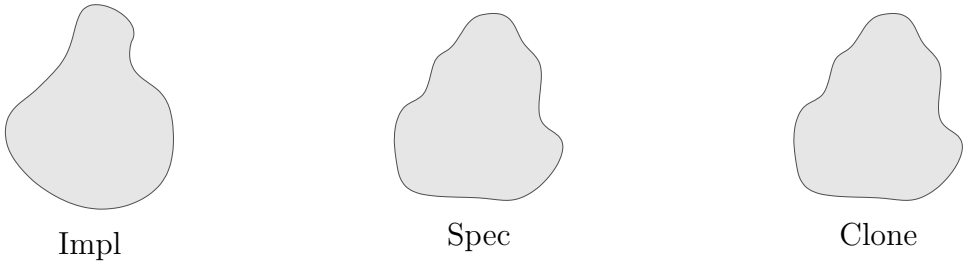


Figure 4.1: Implementation, Specification, and Clone

Game$_0$: Put the three systems ("Impl", "Spec", and "Clone") in their initial states; then

Step 1 : The player chooses an action a.

Step 2 : Action a is run on "Impl", "Spec", and "Clone".

Step 3 : If a succeeds on the three processes; go to Step 1. Else the game ends and the reward is computed as follows: a $(+1)$ reward is given for different observations between "Impl" and "Spec" 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, the immediate reward is equal to:

$$R := (Obs.I \neq Obs.Spec) \; - \; (Obs.Spec \neq Obs.C) \tag{4.1}$$

where 0 and 1 are used as both truth values and numbers and $Obs.P \in \{\checkmark, \times\}$ designates the observation obtained in process P.

Step 4 : Repeat until the episode ends on one of the three processes.

For example, if action a is executed on the three processes and observation $\langle a^{\times}, a^{\checkmark}, a^{\checkmark} \rangle$ is obtained (i.e., Failure in "Impl", Successes in "Spec" and "Clone"), then an immediate reward of $(1 - 0) = (+1)$ is given. Notice that once an action fails in one of the three processes, the game (episode) ends. Hence, the only scenario allowing to move ahead is: $\langle a^{\checkmark}, a^{\checkmark}, a^{\checkmark} \rangle$ which yields a null reward. It is easy to see that a non-null reward can happen only at the end of the episode. As we will see in Chapter 6, for more complex equivalence notions (trace equivalence being the simplest one), the game will be modified to allow non-null rewards in the middle of the episode. However, we will continue to reveal these rewards at the end of the episode in order to preserve the markov property of the MDP.

This game has been inspired by the Kullback-Leibler divergence defined in Section 2.4.1. Recall that the starting idea is that two processes are trace equivalent via testing if, and only if, they yield the same probability distributions on observations

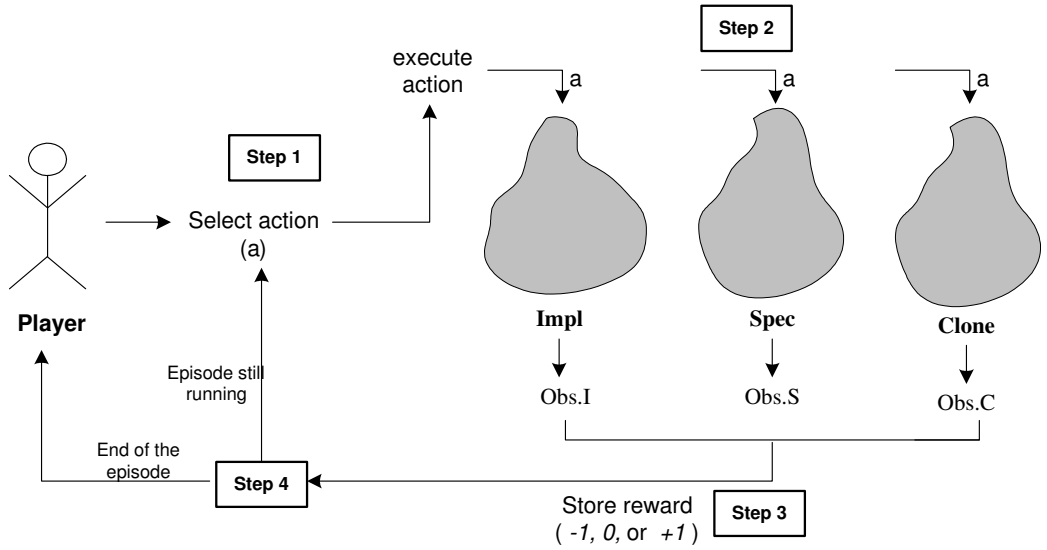


Figure 4.2: The Stochastic Game

for any test generated from the given test grammar. Hence, a divergence between two processes could be defined with the help of a divergence between the *probability distributions* on test observations (e.g. by taking the maximum divergence over all tests). Unfortunately, because of the high number of possible tests (on huge systems), the maximum value over all Kullback-Leibler divergences is not tractable. The stochastic game ($\mathbf{Game}_0$) described earlier is actually a simulation of the tradeoff of the KL divergence. To fix ideas, let us recall the KL divergence definition and then describe the analogy with $\mathbf{Game}_0$. Let p and q be two probability distributions of a random variable with alphabet A, then,

$$KL(p \,\|\, q) = H(p \,\|\, q) - H(p). \tag{4.2}$$

The cross entropy between p and q, $H(p, q)$, can be seen as how likely we can obtain different observations when interacting (via some test t) with both "Impl" and "Spec". On the other hand, the entropy of the distribution p, $H(p)$ 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 a first indication that one can derive from it the notion of divergence we are looking for. Unfortunately, this "simple" form of the game does not lead to a suitable notion of divergence. In particular, two kinds of problems may happen: the undesirable zero value and the negative value problems.

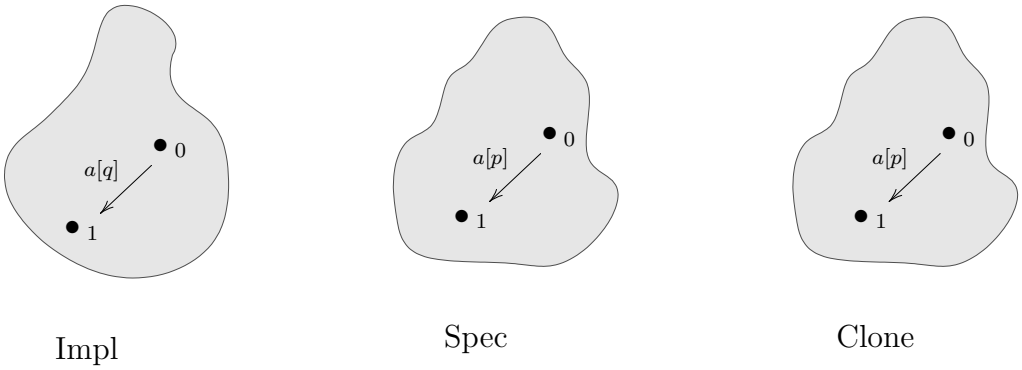


Figure 4.3: A generic example.

4.2.1 Symmetry Problem

In order to be consistent with a notion of trace equivalence divergence, $\mathbf{Game}_0$ should satisfy the two following properties:

1. if "Impl" and "Spec" are trace equivalent, then for any policy the expected reward should be zero.
2. if "Impl" and "Spec" are not trace equivalent, then the expected reward of the optimal policy should be strictly greater than zero.

$\mathbf{Game}_0$ does satisfy the first property but not the second. Consider the simple generic example of Figure 4.3. The only possible policy is $\pi = a$. Following this policy may yield eight possible observations:

$$O = \{\langle a^{\checkmark}, a^{\checkmark}, a^{\checkmark}\rangle, \langle a^{\times}, a^{\checkmark}, a^{\checkmark}\rangle, \langle a^{\times}, a^{\times}, a^{\checkmark}\rangle, \langle a^{\times}, a^{\checkmark}, a^{\times}\rangle, \langle a^{\times}, a^{\times}, a^{\times}\rangle, \langle a^{\checkmark}, a^{\times}, a^{\checkmark}\rangle, \langle a^{\checkmark}, a^{\checkmark}, a^{\times}\rangle, \langle a^{\checkmark}, a^{\times}, a^{\times}\rangle\}.$$

Hence, the expected reward according to π can be expressed as:

$$E_{\pi}(Reward) = \sum_{obs \in O} P(obs)R(obs) \tag{4.3}$$

where $P(obs)$ is the probability to observe obs and $R(obs)$ is the corresponding reward. Since the observations $\langle a^{\checkmark}, a^{\checkmark}, a^{\checkmark}\rangle$, $\langle a^{\times}, a^{\times}, a^{\times}\rangle$, $\langle a^{\checkmark}, a^{\times}, a^{\checkmark}\rangle$, and $\langle a^{\times}, a^{\checkmark}, a^{\times}\rangle$ yield a

reward of 0, we have

$$\begin{aligned} E_\pi(Reward) &= P(\langle a^\checkmark, a^\times, a^\times\rangle)R(\langle a^\checkmark, a^\times, a^\times\rangle) \\ &\quad + P(\langle a^\times, a^\checkmark, a^\checkmark\rangle)R(\langle a^\times, a^\checkmark, a^\checkmark\rangle) \\ &\quad + P(\langle a^\checkmark, a^\checkmark, a^\times\rangle)R(\langle a^\checkmark, a^\checkmark, a^\times\rangle) \\ &\quad + P(\langle a^\times, a^\times, a^\checkmark\rangle)R(\langle a^\times, a^\times, a^\checkmark\rangle) \qquad (4.4) \\ &= q\,(1-p)\,(1-p)\,(+1) + (1-q)\,p\,p\,(+1) \\ &\qquad\qquad + q\,p\,(1-p)\,(-1) + (1-q)\,(1-p)\,p\,(-1) \\ &= 2q^2 - 2pq - q + p \\ &= (p-q)(1-2p). \qquad (4.5) \end{aligned}$$

Undesirable zero value According to Equation (4.5), the expected reward of policy π is equal to 0 if $p = q$ or $1 - 2p = 0$. The first situation means that "Impl" and "Spec" are trace equivalent which is the situation we expect. However, the second situation implies that if $p = \frac{1}{2}$, then the expected reward will be 0 regardless of the value of q (it can be different from p). Hence, "Impl" and "Spec" can be not trace equivalent while the expected reward of π is zero, which is a non expected case. Intuitively, $p = \frac{1}{2}$ means that the probabilities to obtain $a^\checkmark$ and $a^\times$ in "Spec" are equal. If we turn to Equation (4.4), this implies that $P(\langle a^\checkmark, a^\checkmark, a^\times\rangle) = P(\langle a^\checkmark, a^\times, a^\times\rangle)$ and that $P(\langle a^\times, a^\checkmark, a^\checkmark\rangle) = P(\langle a^\times, a^\times, a^\checkmark\rangle)$. Hence, the probabilities to obtain reward (-1) and reward $(+1)$ are equal which leads to an expected reward of 0. Since these kinds of situations exhibit a form of symmetry, we call them *symmetric* cases[1].

Negative Value Problem If "Impl" and "Spec" are not trace equivalent, the value of the optimal policy should not only be different than zero but strictly positive. However, as defined above, $\mathbf{Game}_0$ may yield to optimal policies having negative values. Let us return to the example of Figure 4.3. According to Equation (4.5), the expected reward for $\pi = a$, the only possible policy, may be negative in two situations. The first is when $q > \frac{1}{2}$ and $p > q$ and the second is when $q < \frac{1}{2}$ and $p < q$. Intuitively, this negative value problem happens when the likelihood to obtain different observations between "Spec" and "Clone", which is due to the entropy of "Spec", is more important than the likelihood to obtain different observations between "Impl" and "Spec".

[1]The symmetry problem is not specific to trace equivalence and can be observed with other equivalence notions (see Chapter 6).

4.2.2 Prediction

Because of these two problems, it is clear that 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 will allow to get rid of these two problems and will lead to a suitable notion of divergence.

Running action a on process "Spec" yields one of two possible observations $a^{\checkmark}$ or $a^{\times}$. The idea is then to split the action a into two variants depending on its outcome on process "Spec". These two variants are noted simply $a^{\checkmark}$ and $a^{\times}$. Hence, a new modified game, $\mathbf{Game}_{\text{trace}}$, is defined based on $\mathbf{Game}_0$ as follows:

Definition 4.1. $\mathbf{Game}_{\text{trace}}$ is $\mathbf{Game}_0$ with *Step 1* replaced by

Step 1' : The player chooses an action a and makes a prediction on its success or failure on "Spec"($\mathtt{Pred} \in \{a^{\checkmark}, a^{\times}\}$)

and the reward function is replaced by

$$R := \big(Obs.Spec = \mathtt{Pred}\big)\big((Obs.I \neq Obs.Spec) - (Obs.Spec \neq Obs.C)\big). \tag{4.6}$$

For example, if $a^{\checkmark}$ is selected ($\mathtt{Pred} = a^{\checkmark}$) and the observation is $\langle a^{\times}, a^{\checkmark}, a^{\checkmark}\rangle$ 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 $\langle a^{\times}, a^{\times}, a^{\checkmark}\rangle$, we obtain $0\ (0-1) = 0$.

As we will see in the next sections, splitting the action a, on one hand, will break the symmetry of symmetric situations and, on the other hand, will guarantee that there is a policy with a positive value in case the processes are different. To fix ideas, let us reconsider the example of Figure 4.3. Since each action is split into two variants, we will have two possible policies $\pi = a^{\checkmark}$ and $\pi = a^{\times}$. The expected reward values for these policies are:

$$\begin{aligned} E_{\pi=a^{\checkmark}}(Reward) &= P(\langle a^{\times}, a^{\checkmark}, a^{\checkmark}\rangle)R(\langle a^{\times}, a^{\checkmark}, a^{\checkmark}\rangle) + P(\langle a^{\checkmark}, a^{\checkmark}, a^{\times}\rangle)R(\langle a^{\checkmark}, a^{\checkmark}, a^{\times}\rangle) \\ &= (1-q)\,p^2 - q\,p\,(1-p) \\ &= p\,(p-q). \end{aligned}$$

$$\begin{aligned} E_{\pi=a^{\times}}(Reward) &= P(\langle a^{\checkmark}, a^{\times}, a^{\times}\rangle)R(\langle a^{\checkmark}, a^{\times}, a^{\times}\rangle) + P(\langle a^{\times}, a^{\times}, a^{\checkmark}\rangle)R(\langle a^{\times}, a^{\times}, a^{\checkmark}\rangle) \\ &= q\,(1-p)\,(1-p) - (1-q)\,(1-p)\,p \\ &= (1-p)\,(q-p). \end{aligned}$$

It is easy to see that the symmetric case cannot happen anymore because

$$E_{\pi=a^{\checkmark}}(Reward) = E_{\pi=a^{\times}}(Reward) = 0$$

is obtained only when $p = q$. Also, if $E_{\pi=a^{\checkmark}}(Reward) < 0$ then $E_{\pi=a^{\times}}(Reward) > 0$ and vice-versa which guarantees that the optimal policy cannot have a negative value. This modification, however, comes with a side effect which is the doubling of the number of actions. Nevertheless, the impact of this duplication on the game (MDP) is not very important since, as we will see later, both variants $a^{\checkmark}$ and $a^{\times}$ will make a transition to the same next state of the MDP. Hence, unlike the number of actions, the number of states remains the same.

4.3 Constructing the MDP $\mathcal{M}$

In reinforcement learning, the problems are typically formalized using MDPs. The MDP on which the divergence between two LMPs ("Spec" and "Impl") will be computed can be straightforwardly obtained from $\mathbf{Game}_{\text{trace}}$. In this section, we give the definition of this MDP in terms of the conditional probabilities $P^{Spec}(\,.\,|\,.\,)$ and $P^{I}(\,.\,|\,.\,)$. $P^{Spec}(a^{\checkmark}|\tau)$ (resp. $P^{I}(a^{\checkmark}|\tau)$) is the conditional probability of observing the success of an action a given a successfully executed trace τ in "Spec" (resp. in "Impl")[2]. Using $P^{Spec}(\,.\,|\,.\,)$ and $P^{I}(\,.\,|\,.\,)$, we can give an alternative definition of trace equivalence between LMPs.

Definition 4.2. *Two LMPs "Spec" and "Impl" with the same action set $\mathcal{A}$ are trace equivalent if, and only if, $\forall\, a \in \mathcal{A}, \forall\, \tau \in \mathcal{A}^*$,*

$$P^{Spec}(a^{\checkmark}|\tau) = P^{I}(a^{\checkmark}|\tau) \quad \textit{and} \quad P^{Spec}(a^{\times}|\tau) = P^{I}(a^{\times}|\tau).$$

Since the models of "Spec" and "Impl" are not available, the conditional probabilities $P^{Spec}(\,.\,|\,.\,)$ and $P^{I}(\,.\,|\,.\,)$ are initially unknown. It is by interaction that the learning algorithm (the player in $\mathbf{Game}_{\text{trace}}$) will "learn" their values and consequently learn the MDP.

Definition 4.3. Let "Impl", "Spec", and "Clone" be three LMPs with the same set of actions $\mathcal{A}$. The MDP $\mathcal{M}$ induced by the three LMPs is a tuple $(S, \mathsf{i}, Act, Pr^{\mathcal{M}}, R)$. The state representation S is the set of accepted traces in "Spec" along with a *Dead* state:

$$S := \{\tau : \mathcal{A}^* \mid P^{\text{Spec}}(\tau) > 0\} \cup \{Dead\}.$$

[2]We write $P^{C}(a^{\checkmark}|\tau)$ and $P^{C}(a^{\times}|\tau)$ for the conditional probabilities in "Clone". This is for readability and is no additional information since $P^{C}(a^{\checkmark}|\tau) = P^{Spec}(a^{\checkmark}|\tau)$.

The initial state is $\mathsf{i} = \epsilon$ (the empty sequence). The set of actions is:

$$Act := \mathcal{A} \times \{ {}^{\checkmark}, {}^{\times} \}.$$

The next-state probability distribution $Pr^{\mathcal{M}}$ is the same for $a^{\checkmark}$ and $a^{\times}$ (which we represent generically with $a^{\text{-}}$); it is defined below, followed by the definition of the immediate reward. Let $s = \tau$ the state corresponding to the trace τ, then,

$$Pr^{\mathcal{M}}_{s \to s'}(a^{\text{-}}) \;:=\; \begin{cases} P^{Spec}(a^{\checkmark}|s)\; P^{I}(a^{\checkmark}|s)\; P^{C}(a^{\checkmark}|s) & \text{if } s' = \tau\, a \\ 1 - P^{Spec}(a^{\checkmark}|s)\; P^{I}(a^{\checkmark}|s)\; P^{C}(a^{\checkmark}|s) & \text{if } s' = \mathit{Dead} \\ 0 & \text{otherwise.} \end{cases}$$

$$R^{a^{\text{-}}}_{s} := P^{Spec}(a^{\text{-}}|s)\; \Delta^{a^{\text{-}}}_{s} \tag{4.7}$$

where
- $\Delta^{a^{\checkmark}}_{\tau} := P^{C}(a^{\checkmark}|\tau) - P^{I}(a^{\checkmark}|\tau)$, and
- $\Delta^{a^{\times}}_{\tau} := -\Delta^{a^{\checkmark}}_{\tau}$.

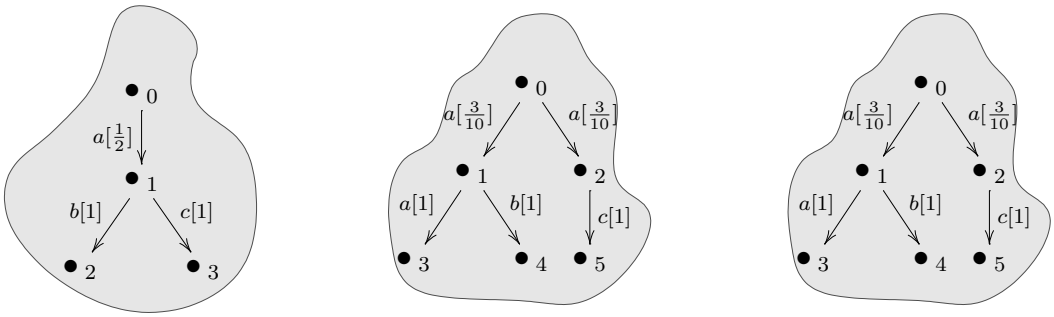
Whether it is $a^{\checkmark}$ which has been selected or $a^{\times}$ (Step 1 of $\mathbf{Game}_{\text{trace}}$), moving to the next state $s' = \tau\, a$ is possible only if the action a is accepted in the three processes. However, refusing the action a on at least one of the three processes causes a transition to the *Dead* state. Hence the *Dead* state indicates the end of an episode. This is formulated in the next state probability distribution $Pr^{\mathcal{M}}_{s \to s'}(a^{\text{-}})$ and $Pr^{\mathcal{M}}_{s \to Dead}(a^{\text{-}})$ in Definition 4.3. The definition of $R^{a^{\text{-}}}_{s}$ is obtained from $\mathbf{Game}_{\text{trace}}$ as follows. In the case where $a^{\text{-}} = a^{\checkmark}$ is chosen, then the reward is computed only if a succeeds in "Spec", that is, we get a $(+1)$ reward on observation $\langle a^{\times}, a^{\checkmark}, a^{\checkmark} \rangle$, a (-1) on $\langle a^{\checkmark}, a^{\checkmark}, a^{\times} \rangle$, and $(+0)$ on observations $\langle a^{\checkmark}, a^{\checkmark}, a^{\checkmark} \rangle$ or $\langle a^{\times}, a^{\checkmark}, a^{\times} \rangle$. If a fails, the reward is $(+0)$. Thus,

$$\begin{aligned} R^{a^{\checkmark}}_{s} &= \underbrace{P^{I}(a^{\times}|s)\; P^{Spec}(a^{\checkmark}|s)\; P^{C}(a^{\checkmark}|s)}_{\langle \times, \checkmark, \checkmark \rangle \mapsto (+1)} - \underbrace{P^{I}(a^{\checkmark}|s)\; P^{Spec}(a^{\checkmark}|s)\; P^{C}(a^{\times}|s)}_{\langle \checkmark, \checkmark, \times \rangle \mapsto (-1)} \\ &= P^{Spec}(a^{\checkmark}|s)\; \big(P^{I}(a^{\times}|s)\; P^{C}(a^{\checkmark}|s) - P^{I}(a^{\checkmark}|s)\; P^{C}(a^{\times}|s) \big) \\ &= P^{Spec}(a^{\checkmark}|s)\; \big(P^{C}(a^{\checkmark}|s) - P^{I}(a^{\checkmark}|s) \big) \\ &= P^{Spec}(a^{\checkmark}|s)\; \Delta^{a^{\checkmark}}_{s}. \end{aligned}$$

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 $\langle a^{\checkmark}, a^{\times}, a^{\times} \rangle$ $(+1)$ and $\langle a^{\times}, a^{\times}, a^{\checkmark} \rangle$ (-1) are relevant. Thus,

$$R^{a^{\times}}_{s} = P^{Spec}(a^{\times}|s)\; \Delta^{a^{\times}}_{s}.$$

We therefore obtain Equation (4.7).



Impl Spec Clone

Figure 4.4: The "Impl", "Spec", and "Clone" processes of a simple example.

Example 4.4. *Consider the three processes of Figure 4.4. The MDP $\mathcal{M}$ induced by these processes is illustrated in Figure 4.5. Each node represents a state of the MDP $\mathcal{M}$ and is labelled with the corresponding trace. Each edge represents a transition from one state s to the next s' and is labelled with the probability $Pr^{\mathcal{M}}_{s\to s'}(a^{\text{-}})$ and with the expected immediate reward $R^{a^{\text{-}}}_{s}$. For example, if the agent is in state $s_1 = a$ and chooses action $c^{\times}$, then with probability $\frac{9}{100}$ it will make a transition to state $s_2 = ac$ and obtain an immediate reward of $\frac{1}{3}$. To keep the Figure less complex, all transitions to the Dead state with probability 1 and reward 0 have been omitted (e.g. $Pr^{\mathcal{M}}_{\mathsf{i}\to Dead}(b^{\checkmark}) = 1$ and $R^{b^{\checkmark}}_{\mathsf{i}} = 0$).*

4.4 Main Theorem and Definition of $\mathbf{div}_{\mathbf{trace}}(\,.\,\|\,.\,)$

The key idea of the proposed approach is that the MDP is constructed in such a way that the optimal value coincides with the actual divergence between LMPs "Spec" and "Impl".

Theorem 4.5. *Let $\mathcal{M}$ be the MDP induced by "Impl", "Spec", and "Clone". If the discount factor $\gamma < 1$ or the size of the MDP $|\mathcal{M}| < \infty$ then the optimal value $V^{\star}(\mathsf{i}) \geq 0$, and $V^{\star}(\mathsf{i}) = 0$ if, and only if, "Impl" and "Spec" are trace equivalent.*

The proof of Theorem 4.5 is given in Section 4.5. In the light of this theorem, the trace equivalence divergence between LMPs is defined as:

Definition 4.6. Let "Impl" and "Spec" be two LMPs and $\mathcal{M}$ their induced MDP. We define their *trace equivalence divergence* as

$$\mathrm{div}_{\mathrm{trace}}(\text{“Spec”}\,\|\,\text{“Impl”}) := V^{\star}(\mathsf{i}).$$

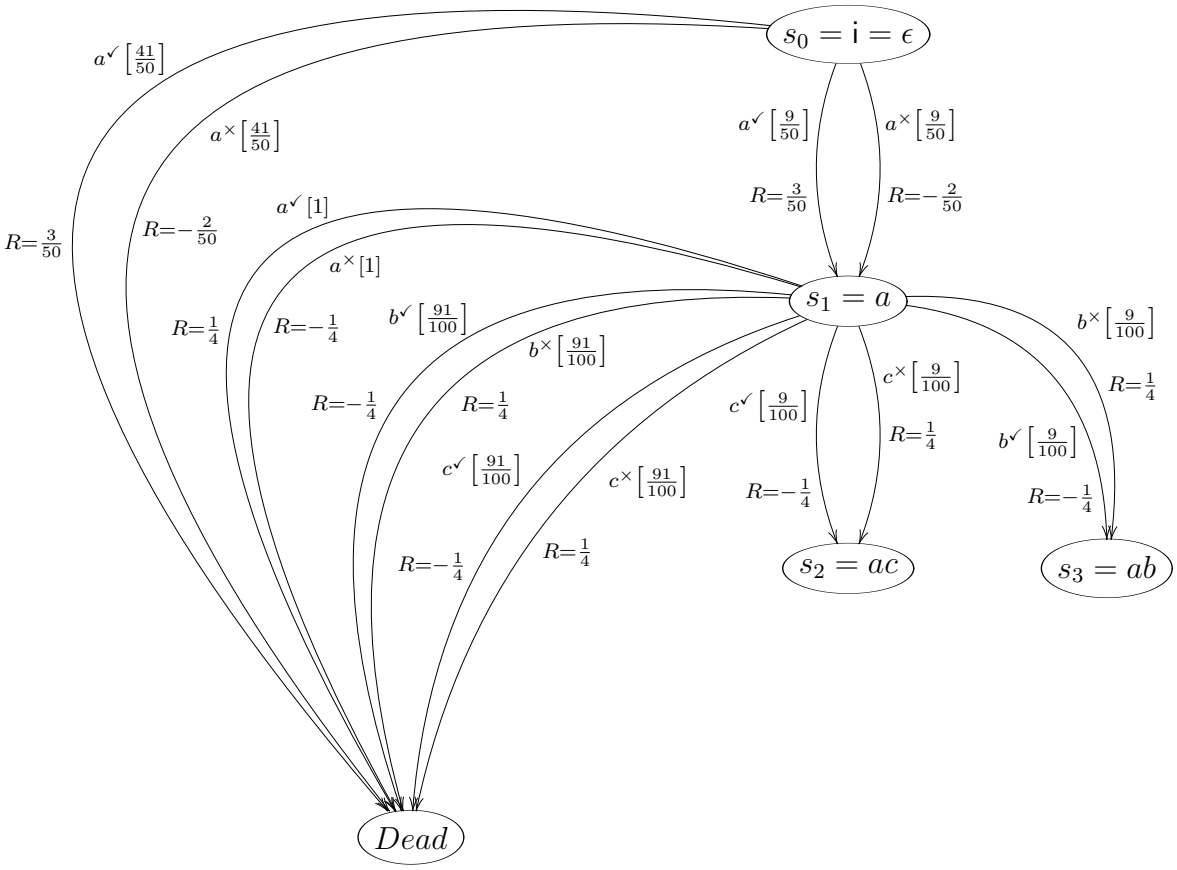


Figure 4.5: MDP induced by the Example 4.4 (all the transitions to the *Dead* state with probability 1 and reward 0 are omitted).

4.5 Proofs

Theorem 4.5 states that the optimal policy of the MDP $\mathcal{M}$ coincides with the divergence between LMPs. Hence, the first step towards proving the theorem is to figure out the formula of evaluating a policy π.

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

Let π be a policy. In order to define a formula for evaluating π on any state s, we start from the Bellman Equation (Section 3.2.6). Recall that $a^{\text{-}}$ denotes generically $a^{\checkmark}$ and $a^{\times}$.

$$\begin{aligned} V^{\pi}(s) &= \sum_{a^{\text{-}} \in Act} \pi(s, a^{\text{-}}) \sum_{s' \in S} Pr^{\mathcal{M}}_{s \to s'}(a^{\text{-}}) \, (R^{a^{\text{-}}}_{s} + \gamma \, V^{\pi}(s')) \\ &= \sum_{a^{\text{-}} \in Act} \pi(s, a^{\text{-}}) \, (R^{a^{\text{-}}}_{s} + Pr^{\mathcal{M}}_{s \to s.a}(a^{\text{-}}) \, \gamma \, V^{\pi}(s.a)). \end{aligned} \tag{4.8}$$

In the context of MDP $\mathcal{M}$, we will focus on a particular type of policy that we call test-policy. A test-policy is a deterministic (not stochastic) policy which has the form $\pi = a^{\text{-}}_1 a^{\text{-}}_2 \ldots a^{\text{-}}_n$. That is, it consists in choosing action $a^{\text{-}}_1$ at state $\mathsf{i} = \epsilon$, action $a^{\text{-}}_2$ at state $s = a_1$, action $a^{\text{-}}_3$ at state $s = a_1 a_2$, etc. Note that there is an abuse of language when we say that a test-policy π is a policy of the MDP $\mathcal{M}$ because it specifies which action to take only in states $\epsilon, a_1, a_1 a_2, \ldots, a_1 \ldots a_n$. However, if we assume that there exists an action "*abort*" in the MDP which transits with probability 1 to the *Dead* state and yields a reward of 0. Then, the test-policy π can be extended to all the states of the MDP such that for all states $s \notin \{\epsilon, a_1, a_1 a_2, \ldots, a_1 a_2 \ldots a_n\}$ it consists in choosing the action "*abort*". Since there is no "*abort*" action in the MDP, one should take into consideration this aspect of test-policy in the rest of the proof.

The value of the test-policy $\pi = a^{\text{-}}_1 a^{\text{-}}_2 \ldots a^{\text{-}}_n$ on i is thus determined as follows from Equation (4.8):

$$\begin{aligned} V^{\pi}(\mathsf{i}) &= \sum_{a^{\text{-}} \in Act} \pi(\mathsf{i}, a^{\text{-}}) \, (R^{a^{\text{-}}}_{\mathsf{i}} + Pr^{\mathcal{M}}_{\mathsf{i} \to \mathsf{i}.a}(a^{\text{-}}) \, \gamma \, V^{\pi}(\mathsf{i}.a)) \\ &= R^{a^{\text{-}}_1}_{\epsilon} + Pr^{\mathcal{M}}_{\epsilon \to a_1}(a_1) \, \gamma \, \left(R^{a^{\text{-}}_2}_{a_1} + Pr^{\mathcal{M}}_{a_1 \to a_1 a_2}(a_2) \, \gamma \, V^{\pi}(a_1 a_2) \right) \\ &= R^{a^{\text{-}}_1}_{\epsilon} + Pr^{\mathcal{M}}(a_1) \, \gamma \, R^{a^{\text{-}}_2}_{a_1} + Pr^{\mathcal{M}}(a_1 a_2) \, \gamma^2 \, R^{a^{\text{-}}_2}_{a_1 a_2} + \ldots \\ &= \sum_{i=0}^{n-1} Pr^{\mathcal{M}}(a_0 \ldots a_i) \, \gamma^i \, R^{a^{\text{-}}_{i+1}}_{a_0 \ldots a_i} \end{aligned} \tag{4.9}$$

where $Pr^{\mathcal{M}}(\tau)$ is the probability to reach state $s = \tau$ from the initial state i, and a_0 denotes ϵ.

4.5.2 Proof of Theorem 4.5

We prove in this section that solving the MDP induced by two LMPs "Impl" and "Spec" yields an equivalence divergence between them (i.e., which is positive and has value 0 if and only if the two processes are trace equivalent). Let us first enounce this easy lemma which will be useful in this section.

Lemma 4.7. *The following are equivalent.*

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

The proof of this lemma is straightforwardly obtained from the definition of $\Delta_\tau^{a^-}$ in Definition 4.3.

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

Theorem 4.8. *The following are equivalent:*

(i) *"Impl" and "Spec" are trace equivalent.*

(ii) $\forall a^- \in Act, \tau \in S\backslash\{\text{Dead}\}, \quad R_\tau^{a^-} = 0$

(iii) $\forall$ *test-policy* π, $\quad V^\pi(\mathsf{i}) = 0$.

Proof. $(i) \Rightarrow (ii)$. By (i) and Definition 4.2, we have $P^{Spec}(\,.\,|\,.\,) = P^I(\,.\,|\,.\,)$. Since τ is a trace $\forall\, s \neq Dead$, and since $P^{Spec}(\,.\,|\,.\,) = P^C(\,.\,|\,.\,)$,

$$\begin{aligned} P^{Spec}(\,.\,|\,.\,) = P^I(\,.\,|\,.\,) \quad &\Rightarrow \quad \forall a^- \in Act, s \in S\backslash\{Dead\}, \quad P^C(a^-|\tau) = P^I(a^-|\tau) \\ &\Leftrightarrow \quad \forall a^- \in Act, \tau \in S\backslash\{Dead\}, \quad \Delta_\tau^{a^-} = 0 \qquad \text{(by Lemma 4.7)} \\ &\Rightarrow \quad \forall a^- \in Act, \tau \in S\backslash\{Dead\}, \quad R_\tau^{a^-} = 0. \quad \text{(by Equation (4.7))} \end{aligned}$$

$(ii) \Rightarrow (i)$. Let $\tau \in S\backslash\{Dead\}$ be a state with trace τ. Let $a^- \in Act$. Then by (ii) and Equation (4.7), we have $R_\tau^{a^-} = P^{Spec}(a^-|\tau)\,\Delta_\tau^{a^-} = 0$. This implies either $\Delta_\tau^{a^-} = 0$ or $P^{Spec}(a^-|\tau) = 0$. By Lemma 4.7 and Definition 4.2, we know that the first case implies the result. The second case requires to see the opposite action. Indeed, $P^{Spec}(a^-|\tau) = 0$ implies that $P^{Spec}(\overline{a^-}|\tau) = 1$. By (ii), $R_s^{\overline{a^-}} = 0$. By the same argument, we can deduce that either $\Delta_\tau^{\overline{a^-}} = 0$ or $P^{Spec}(\overline{a^-}|\tau) = 0$, and therefore that $\Delta_\tau^{\overline{a^-}} = 0$. Since $\overline{\overline{a^-}} = a^-$, by Lemma 4.7 (replacing a^- by $\overline{a^-}$), we have $P^I(a^-|\tau) = P^{Spec}(a^-|\tau)$ which by Definition 4.2 implies the result.
$(ii) \Rightarrow (iii)$. Follows from Equation (4.9).
$(iii) \Rightarrow (ii)$. Fix $a^- \in Act$ and $\tau \in S\backslash\{Dead\}$ such that $a_1 \dots a_n = \tau$. Now, define the test-policy $\pi = a_1^- \dots a_n^-$. By Equation (4.9), we have $V^{\pi a^-}(\mathsf{i}) - V^\pi(\mathsf{i}) = Pr^{\mathcal{M}}(\tau)\ \gamma^n\ R_\tau^{a^-}$. By (iii) we will have $Pr^{\mathcal{M}}(\tau)\ \gamma^n\ R_\tau^{a^-} = 0$. Since $\gamma > 0$, if $Pr^{\mathcal{M}}(\tau) \neq 0$ then Equation (4.7) will imply the result. So, by way of contradiction suppose $Pr^{\mathcal{M}}(\tau) = 0$. This implies that $P^{\mathrm{I}}(\tau) = 0$ because, by definition of the MDP $\mathcal{M}$, $P^{\mathrm{Spec}}(\tau) \neq 0$ and because $P^{\mathrm{Spec}}(\tau) = P^{\mathrm{C}}(\tau)$. Now let $a_1 a_2 \dots a_k$ be the smallest subsequence of τ such that $P^{\mathrm{I}}(a_1 a_2 \dots a_k) = 0$. Since $P^{\mathrm{I}}(\epsilon) = 1$, $k \geq 1$. Fix $\tau' = a_1 a_2 \dots a_{k-1}$, and observe that both τ' and $\tau' a_k$ are states of the MDP $\mathcal{M}$ because $P^{\mathrm{Spec}}(\tau') \neq 0$ and $P^{\mathrm{Spec}}(\tau' a_k) \neq 0$. By the hypothesis and Equation (4.9), we have $0 - 0 = V^{a_1^- a_2^- \dots a_k^-}(\mathsf{i}) - V^{a_1^- a_2^- \dots a_{k-1}^-}(\mathsf{i}) = Pr^{\mathcal{M}}(\tau')\ \gamma^n\ R_{\tau'}^{a_k^-}$. Since, by construction, $Pr^{\mathcal{M}}(\tau') \neq 0$, we therefore have that $R_{\tau'}^{a_k^-} = 0$. It then follows from Equation (4.7) that $\Delta_{\tau'}^{a_k^-} = 0$, and hence that $P^I(a_k^-|\tau') = P^C(a_k^-|\tau')$. On an other hand, $P^I(a_k^-|\tau') = 0$, because $P^{\mathrm{I}}(\tau') \neq 0$ and $P^{\mathrm{I}}(\tau' a_k) = 0$. Thus, $P^C(a_k^-|\tau') = 0$, which in turn implies that $P^{\mathrm{C}}(\tau' a_k) = 0$, and therefore that $P^{\mathrm{Spec}}(\tau' a_k) = 0$. A contradiction. □

Theorem 4.9. *Two LMPs are not trace equivalent if and only if $V^\pi(\mathsf{i}) > 0$ for some test-policy π of the MDP $\mathcal{M}$.*

Proof. $(\Rightarrow)$: by Theorem 4.8, we have that $V^\pi(\mathsf{i}) \neq 0$ for some test-policy $\pi = a_1^- a_2^- \dots a_n^-$. Define the set of indices J as follows:

$$J := \{j \in \{1, \dots n\} \mid R_{a_1 \dots a_{j-1}}^{a_j^-} = P^{Spec}(a_j^-|a_1 \dots a_{j-1})\ \Delta_{a_1 \dots a_{j-1}}^{a_j^-} < 0\},$$

and note that $P^{Spec}(a_j^-|a_1 \dots a_{j-1}) > 0$ and $\Delta_{a_1 \dots a_{j-1}}^{a_j^-} < 0$ for any $j \in J$. Thus, for any such j, $\Delta_{a_1 \dots a_{j-1}}^{\overline{a_j^-}} > 0$. Let π_1 be the policy obtained from π by replacing each action a_j^- where $j \in J$ by $\overline{a_j^-}$. Then, by Equation (4.9) and Equation (4.7), $V^{\pi_1}(\mathsf{i}) > 0$, as desired.
$(\Leftarrow)$: follows from Theorem 4.8. □

Lemma 4.10. *For every test-policy π, for every $a^- \in Act$,*

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

Proof. As for Theorem 4.9, the result follows from Equation (4.9) and the fact that $\Delta^{\overline{a^-}}_{\tau_\pi} = -\Delta^{a^-}_{\tau_\pi}$ where τ_π is the trace corresponding to test-policy π (if $\pi = a_1^- \dots a_n^-$ then $\tau_\pi = a_1 \dots a_n$). □

Theorem 4.11. *If the discount factor $\gamma < 1$ or the size of MDP $|\mathcal{M}| < \infty$ then $V^\star(\mathsf{i}) \geq V^\pi(\mathsf{i})$ for any test-policy π.*

Proof. If $|\mathcal{M}| < \infty$, since $\mathcal{M}$ has a tree structure, the result is a direct consequence of Lemma 4.10. Otherwise, it is sufficient to show that

$$\forall \epsilon > 0 \ \forall \text{ test-policy } \pi \ \exists \text{policy } \pi' \ \text{ such that } \ V^{\pi'}(\mathsf{i}) < V^\pi(\mathsf{i}) - \epsilon.$$

Let $\epsilon > 0$ and $\pi = a_1^- \dots a_n^-$ be a test-policy. Because of Lemma 4.10, 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 the states $\epsilon, a_1, a_1a_2, \dots, a_1a_2 \dots a_n$ will have the desired property.

□

If "Impl" and "Spec" are trace equivalent, Theorem 4.8 implies that $V^\star(\mathsf{i}) = 0$. If they are not trace equivalent, Theorem 4.9 implies that there exists at least a test-policy π such that $V^\pi(\mathsf{i}) > 0$. By Theorem 4.11, we can conclude that $V^\star(\mathsf{i}) > 0$.

Theorem 4.5. *Let $\mathcal{M}$ be the MDP induced by "Impl" and "Spec". If the discount factor $\gamma < 1$ or the size of MDP $|\mathcal{M}| < \infty$ then the optimal value $V^\star(\mathsf{i}) \geq 0$, and $V^\star(\mathsf{i}) = 0$ if, and only if, "Impl" and "Spec" are trace equivalent.*

4.6 PAC Guarantee and Experimental Results

Reinforcement learning algorithms are based on sampling. Hence, the precision of the result depends on the number of times the task is repeated, that is, the number of episodes. In this section, we discuss the guarantees we can provide for the precision of the returned result along with the empirical analysis we performed on different LMPs.

4.6.1 PAC Guarantee

As described in the previous sections, the key idea of our approach is to define an MDP out of the processes to be tested and to interpret the optimal value of this MDP as a divergence between them. If the model of the MDP is completely known and its size is tractable, then one can use dynamic programming to obtain the exact value of the divergence. However, if the MDP model is either huge or not available, then it might be impossible to obtain such exact value. In such situations, a reinforcement learning (RL) algorithm based on sampling such as Q-learning can still be tractable. Q-learning is a temporal difference control algorithm which directly approximates $V^*(\mathsf{i})$. Since it is based on sampling, the Q-learning algorithm returns an estimation of the actual optimal value ($V^{\hat{\pi}}(\mathsf{i})$). The precision of this estimation depends on the number of episodes the algorithm performs. Hence, from that perspective, one needs a guarantee on the precision of the returned result such as PAC guarantee.

Definition 4.12. *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 $f(\mathcal{M}_K, \epsilon, \delta)$ such that if the number of episodes is greater than $f(\mathcal{M}_K, \epsilon, \delta)$, then*

$$Prob\{|\overline{V^{\hat{\pi}}(\mathsf{i})} - V^{\star}(\mathsf{i})| \leq \epsilon\} \geq 1 - \delta \tag{4.10}$$

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

Q-learning has been proven to converge to the optimal value but unfortunately, unless we wait until infinity, there are few results on the precision of the solution (PAC guarantee). Indeed, one can compute a lower bound for the optimal policy of the MDP $\mathcal{M}$ while the upper bound, as far as it concerns our setting, remains intractable.

This implies that if the two processes are trace equivalent, the Q-learning algorithm is not the best strategy to compute the divergence. On the other hand, if the two processes are not trace equivalent, an RL algorithm such as Q-learning will be among the most efficient algorithms to find a policy whose value is greater than zero. A lower bound can then easily be obtained using Hoeffding inequality based on the following idea. Let $\hat{\pi}$ be the policy returned by the RL algorithm. Let $\overline{V^{\hat{\pi}}(\mathsf{i})}$ be the estimation of $V^{\hat{\pi}}(\mathsf{i})$ using a Monte Carlo [51] 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

$$Prob\{|\overline{V^{\hat{\pi}}(\mathsf{i})} - V^{\hat{\pi}}(\mathsf{i})| \leq \epsilon\} \geq 1 - \delta. \tag{4.11}$$

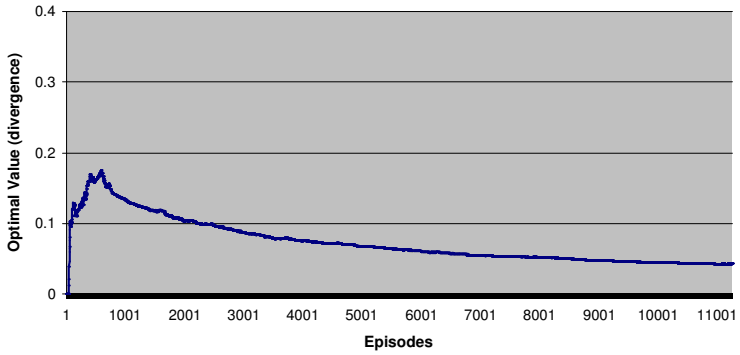


Figure 4.6: Optimal value convergence for trace equivalent processes

Since $V^{\hat{\pi}}(\mathsf{i})$ never exceeds the optimal value $V^{\star}(\mathsf{i})$, we have the following PAC lower bound:

$$Prob\{V^{\star}(\mathsf{i}) \geq \overline{V^{\hat{\pi}}(\mathsf{i})} - \epsilon\} \geq 1 - \delta. \tag{4.12}$$

Thus, at any time during the execution of the RL algorithm, one can extract its current (sub-optimal) policy $\hat{\pi}$ and, via Equation (4.12), obtain a lower bound on $V^{\star}(\mathsf{i})$ via Monte Carlo. If this lower bound is strictly positive, then $\hat{\pi}$ witnesses the difference between "Impl" and "Spec". This witness is important. For example, if "Impl" is some approximation of "Spec", then $\hat{\pi}$ shows to the user "where to look" if he wants to construct a better approximation of $\hat{\pi}$.

Finally, observe that if $\overline{V^{\hat{\pi}}(\mathsf{i})}$ is too close to zero, or exactly zero, then the Hoeffding inequality gives no guarantee on the trace equivalence or non equivalence.

4.6.2 Experimental Results

The approach described so far has been implemented with the Q-learning algorithm. The program takes as input two files representing "Impl" and "Spec" LMPs and approximates their trace equivalence divergence by only interacting with them (their internal structure is used only to interact with them). In order to guarantee a good performance of the Q-learning algorithm, we performed a tuning of its parameters. The tuning process is summarized in Appendix A. The discount factor γ is fixed to 0.8. Two action selection methods have been experimented: ϵ-greedy and SoftMax. For both methods, we tried several functions to vary the ϵ and the τ parameters. The combination that produced the best results is SoftMax such that the temperature τ is decreasing from 5 to 0.01 according to the function : $\tau = \frac{k}{\text{currentEpisod}+l}$ where k and

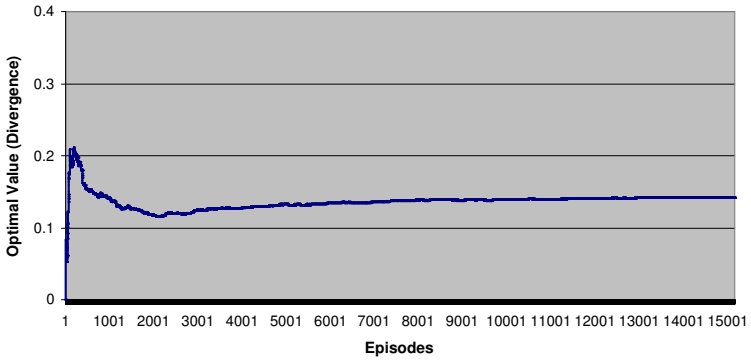


Figure 4.7: Optimal value convergence for non trace equivalent processes

	♯ of states	♯ of actions	♯ of actions/state	♯ of transitions/state
Process 1	17	5	1 - 5	1 - 6
Process 2	24	5	1 - 5	2 - 10
Process 3	32	7	1 - 7	2 - 12
Process 4	97	9	1 - 9	2 - 15
Process 5	166	10	2 - 3	10 - 20
Process 6	200	11	6 - 10	15 - 30
Process 7	207	11	4 - 9	8 - 13
Process 8	519	13	1 - 13	2 - 25
Process 9	944	13	1 - 10	1 - 30

Table 4.1: Characteristics of the randomly generated LMPs.

l are constants. As mentioned in Section 3.6, the learning rate α must decrease in order to assure convergence of the Q-learning algorithm. We tried several decreasing functions and the best convergence results were obtained with the function $\frac{1}{x}$ where x is the number of times the state-action pair has been visited.

As mentioned above, for our application, the RL algorithm (Q-learning) is initially in charge of finding a policy whose value is greater than zero and, if the MDP $\mathcal{M}$ is not too big, to estimate with high precision the optimal value, that is, our divergence notion. Figure 4.6 and Figure 4.7 show how the Q-learning algorithm converges to the optimal policy value on a randomly generated LMP (Process 1 in Table 4.1) when running the algorithm for 50 000 episodes. The convergence happens smoothly both in the first case, when P_1 and P_2 are exactly the Process 1 and in the second case, when P_2 has been slightly modified from the original LMP.

	Episodes	Precision ϵ	Value of the approximation of the optimal policy ($\overline{V^{\hat{\pi}}(\mathsf{i})}$)	Bottom bound ($\overline{V^{\hat{\pi}}(\mathsf{i})} - \epsilon$)
Process 1	$5 * 10^4$	0.005	0.17869	0.17369
Process 2	10^6	0.005	0.00156	-0.00344
Process 3	10^7	0.005	0.07625	0.07125
Process 4	$2 * 10^7$	0.001	0.00720	0.00620
Process 5	$2 * 10^7$	0.0005	0.00105	0.00055
Process 6	$3 * 10^6$	0.001	0.01979	0.01879
Process 7	$5 * 10^6$	0.001	0.01990	0.01890
Process 8	10^7	0.001	0.00263	0.00163
Process 9	10^8	0.0005	0.00014	-0.00036

Table 4.2: The results of running our divergence algorithm on slightly different LMPs. $\hat{\pi}$ is the optimal policy returned by the Q-learning algorithm. $\overline{V^{\hat{\pi}}(\mathsf{i})}$ is the Monte Carlo estimation of $V^{\hat{\pi}}(\mathsf{i})$.

To check if, in practice, our algorithm can rapidly find differences between slightly different LMPs, we made a series of experiments on randomly generated LMPs. The tool we used to randomly generate LMPs [65] is very flexible and it accepts several parameters, namely, the number of states, the number of actions, a range for the number of actions leaving a state, and a range of the number of transitions leaving a state. Table 4.1 shows the values of these parameters for the 9 processes used in our experimentation.

The experimentation consists in comparing each process with a slightly modified version of the same process. The modification consists in the following. We choose at random a state s and a transition leaving that state to another state, let's say, s'. Then, we replace the state s' in the transition with a third state s'' selected randomly. Table 4.2 shows the result of this experimentation. For each process (Column 1), Column 2 indicates the number of episodes used by the algorithm, Column 3 the selected precision ϵ of the PAC lower bound of Equation (4.12) while the confidence value δ is fixed to 0.05 for all processes. The Monte Carlo estimation of the value of the optimal policy returned by the Q-learning algorithm ($\hat{\pi}$) is given in Column 4. The lower bound for the divergence value is given in Column 5. Note that this lower bound is positive for all processes except for 2 and 9. This tends to confirm that, in practice, the algorithm is able to detect slight differences. In the case of Process 2, the negative lower bound can be explained by the extremely small probability to reach the modified state from the initial state. For Process 9, however, the negative lower bound can be explained by the fact that the divergence was too small to be detected after 10^8 episodes.

The small values in Column 4 (the value of the approximation of the optimal policy) reflects the modifications performed on the processes. For example, in Process 8, there are 519 states, only one state has been slightly tampered with. For all these examples, except 2 and 9, the algorithm was able to identify precisely the modification we performed. Indeed, for most examples, the returned optimal policy indicates the trace to the modified state.

4.7 Conclusion

The main contribution of this thesis is a new framework to quantify the divergence between stochastic processes based on reinforcement learning. The idea is to define an MDP out of the processes to be compared in such a way that its optimal value can be interpreted as a divergence between the two. In this chapter, we defined the elements of this MDP and we provided the formal proofs asserting that its optimal value is zero if the processes are equivalent and strictly positive otherwise. The rest of the chapters can be seen as a generalization of the ideas presented in this chapter. This generalization follows two directions. On one hand, Chapter 5 and Chapter 6 show how the approach can be extended to other equivalence notions. On the other hand, Chapter 7 presents a generalization to the stochastic formalisms used in artificial intelligence such as MDPs, POMDPs, PSRs, etc.

Chapter 5

K-moment Equivalence

5.1 Introduction

For many applications, trace equivalence does not discriminate enough. Bisimulation is more powerful. However, testing bisimulation requires a very expensive form of test that needs to maintain an arbitrary number of replicas of states in memory. This is a well known argument against bisimulation which is usually considered too strong. In this chapter, we propose equivalence notions that do not require recursive replication. In particular, we come up with a new family of equivalence notions that lie between trace equivalence and bisimulation. This new family of equivalence notions constitutes a good compromise between trace (too weak) and bisimulation (too strong) and more importantly, it fits very well in the MDP framework described in the previous chapter. The chapter starts by illustrating the problem with bisimulation.

5.2 Recursive Replication

The test construct $(t_1, \ldots, t_n)$ of the test language $\mathcal{T}_{LS}$ (Definition 2.12) consists in making n copies of the current state and then execute test t_i on the i^{th} copy for $i = 1 \ldots n$. Each one of these tests may in turn need to create copies of the next states. The recursive aspect makes bisimulation very difficult to check in practice. Indeed, this construct requires to make n replicas of the current state and to execute a test on each replica. Since this construct is defined recursively on tests, there is no bound on the number of replicas that must be kept in memory. Probabilistic bisimulation cannot be

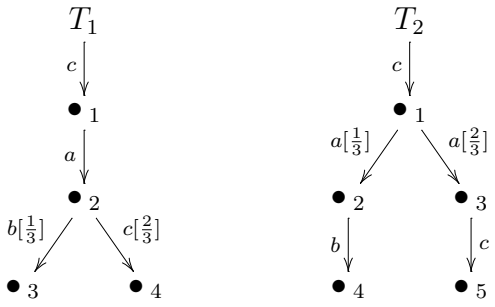


Figure 5.1: Trace equivalent but not bisimilar processes

characterized without the recursive replication construct. The example in Figure 5.1 shows intuitively why. A test distinguishing T_1 and T_2 must use the recursive replication construct to express that both b and c can be performed from the same state. Indeed, the test $c.a.(b.\omega, c.\omega)$ from $\mathcal{T}_{LS}$ distinguishes between them.

In the case of non-deterministic stochastic processes, Kwiatkowska and Norman [29] proposed a testable equivalence notion which is weaker than bisimulation. The test grammar is inspired from the one of bisimulation ($\mathcal{T}_{LS}$) where they impose an independence restriction on the tests of the form $(t_1, \ldots, t_n)$. However, the proposed test grammar still uses recursive replication.

Trace equivalence, contrarily to bisimulation, can be tested without the need to create replicas recursively, but for many applications, it does not discriminate enough. For instance, it cannot discriminate between the two processes of Figure 5.1. Indeed, T_1 and T_2 accept all traces $\{a, ca, cab, cac\}$ with the same probabilities, $1, 1, \frac{1}{3}$, and $\frac{2}{3}$ respectively. One may wonder if equivalences without recursive replication can catch this difference. Such equivalences, if they can be tested efficiently, could be a good compromise between the fact that bisimulation is very costly to test and the fact that trace equivalence does not discriminate enough.

In this chapter, we propose a family of equivalence notions, called *K-moment*, that catch these kinds of differences and make sure to limit the number of replicas. More generally, we refer to equivalences that do not create replicas recursively as recursive replication free equivalences or RRF for short. Each one of the next sections present this family of equivalences from a different point of view.

5.3 The Test Grammar $\mathcal{T}_{Kmoment^0}$

We first present our family of equivalences *K-moment* through a testing framework. The test grammar $\mathcal{T}_{Kmoment^0}$ we propose is a weakening of $\mathcal{T}_{LS}$ that yields a RRF equivalence. It comes with a replication construct that only appears at the end of a trace. We will show later on that this grammar is equivalent with an apparently stronger one ($\mathcal{T}_{Kmoment}$) where one replicates actions at every step of a trace (but no copy is kept in memory). The role of the parameter K will also be explained later.

Definition 5.1. *Let $K \in \mathbb{N}^*$. For $k \leq K$, define:*

$$\mathcal{T}_{Kmoment^0} \qquad t ::= \omega \mid a.t \mid a^k$$

where ω and $a.t$ are the same as in $\mathcal{T}_{\text{trace}}$ (Definition 2.7) whereas test a^k consists in running action a on k copies of the current state and then stop. The observation function O for $\mathcal{T}_{Kmoment^0}$ is defined as follows:

- $O_\omega = \{\omega^\checkmark\}$
- $O_{a.t} = \{a^\times\} \cup \{a^\checkmark e \mid \ e \in O_t\}$
- $O_{a^k} = \underbrace{O_a \times O_a \times \ldots \times O_a}_{k \text{ times}} = (O_a)^k$

where $O_a = \{a^\checkmark, a^\times\}$.
For $0 \leq j \leq k$, the probability distribution for observations of the test a^k is:

$$Q^s_{a^k}(\underbrace{a^\checkmark, \ldots, a^\checkmark}_{j}, \underbrace{a^\times, \ldots a^\times}_{k-j}) = P_{s \to \mathcal{S}}(a)^j \ (1 - P_{s \to \mathcal{S}}(a))^{k-j}$$

where the order of successes and failures is not significant.

Since the execution of a single action a may yield 2 different observations $a^\checkmark$ or $a^\times$, action a^k yields 2^k different observations. For example, executing a^2 has 2^2 possible observations: $\{(a^\checkmark, a^\checkmark), (a^\checkmark, a^\times), (a^\times, a^\checkmark), (a^\times, a^\times)\}$.

The observation function O yields a natural observation set for $\mathcal{T}_{Kmoment^0}$ where the outcome of all the k occurrences in a^k are considered. However, a simpler observation function (Θ) for $\mathcal{T}_{Kmoment^0}$ has the same distinguishing properties as O. The idea is to group observations of O to produce a smaller set of observations for $\mathcal{T}_{Kmoment^0}$ (grouping of observations is discussed with further details in the next chapter).

Definition 5.2. *We define an alternative observation function for the test grammar* $\mathcal{T}_{Kmoment^0}$ *as:*

- $\Theta_\omega = O_\omega = \{\omega^{\checkmark}\}$
- $\Theta_{a.t} = \{a^{\times}\} \cup \{a^{\checkmark}e \mid \ e \in \Theta_t\}$
- $\Theta_{a^k} = \{a^{k^{\checkmark}}, a^{k^{\times}}\}$

such that $a^{k^{\checkmark}}$ *is the observation that all* k *executions of* a *succeeded, and* $a^{k^{\times}}$ *is the observation that at least one execution has failed. The probability distribution on observations for* a^k*,* $Q^s_{a^k}$*, is:*

$$Q^s_{a^k}(a^{k^{\checkmark}}) = P_{s\to\mathcal{S}}(a)^k$$
$$Q^s_{a^k}(a^{k^{\times}}) = 1 - P_{s\to\mathcal{S}}(a)^k.$$

For example, if $k = 2$, the set of four observations of O_{a^2}, that is, $\{(a^{\checkmark}, a^{\checkmark}), (a^{\checkmark}, a^{\times}), (a^{\times}, a^{\checkmark}), (a^{\times}, a^{\times})\}$ will be grouped into 2 observations in $\Theta_{a^2} = \{a^{2^{\checkmark}}, a^{2^{\times}}\}$; observation $a^{2^{\checkmark}}$ corresponds to $(a^{\checkmark}, a^{\checkmark})$ whereas $a^{2^{\times}}$ corresponds to the remaining observations: $\{(a^{\checkmark}, a^{\times}), (a^{\times}, a^{\checkmark}), (a^{\times}, a^{\times})\}$.

Before proving that both observation functions define the same equivalence, we need the following simple lemma.

Lemma 5.3. *For any test* $t = \tau.a^k$ *and any observation* $obs = \tau^{\checkmark}(a^{\text{-}}, \ldots, a^{\text{-}})$*, we have*

$$Q^s_t(obs) = \sum_{s'\in\mathcal{S}} P_{s\to s'}(\tau) Q^{s'}_{a^k}(a^{\text{-}}, \ldots, a^{\text{-}})$$

where $\tau^{\checkmark}$ *stands for the observation that all actions of* τ *have succeeded.*

Proof. The proof of this lemma is a straightforward induction based on the following (induction hypothesis is applied in the second step)

$$\begin{aligned} Q^s_t(a_1^{\checkmark}\ldots a_n^{\checkmark}(a^{\text{-}}, \ldots, a^{\text{-}})) &= \sum_{s'\in\mathcal{S}} P_{s\to s'}(a) Q^{s'}_{a_2\ldots a_n.a^k}(a_2^{\checkmark}\ldots a_n^{\checkmark}(a^{\text{-}}, \ldots, a^{\text{-}})) \\ &= \sum_{s'\in\mathcal{S}} P_{s\to s'}(a) \sum_{s''\in\mathcal{S}} P_{s'\to s''}(a_2\ldots a_n) Q^{s''}_{a^k}(a^{\text{-}}, \ldots, a^{\text{-}}) \\ &= \sum_{s''\in\mathcal{S}} P_{s\to s''}(\tau) Q^{s''}_{a^k}(a^{\text{-}}, \ldots, a^{\text{-}}). \end{aligned}$$

□

Theorem 5.4. *Two LMPs yield the same probability distribution on observations for any test of $\mathcal{T}_{Kmoment^0}$ according to the observation function O if, and only if, they yield the same probability distributions on observations according to the observation function Θ.*

Proof. ($\Rightarrow$). Straightforward since the set of Θ is obtained by grouping one or more observations of O.
($\Leftarrow$). Let $L_1 = (\mathcal{S}_1, \mathcal{A}, \mathsf{i}_1, P)$ and $L_2 = (\mathcal{S}_2, \mathcal{A}, \mathsf{i}_2, P')$ be two LMPs. Let $t = \tau.a^k$ ($\tau \in \mathcal{A}^*$) be a test yielding different probability distributions on observations for L_1 and L_2 according to O. It is sufficient to prove that the same test yields different probability distributions on observations for L_1 and L_2 according to Θ. Let $obs = \tau^{\checkmark}(a^{\text{-}}, \ldots, a^{\text{-}})$ be an observation from O_t such that $Q_t^{L_1}(obs) \neq Q_t^{L_2}(obs)$ and suppose, w.l.o.g, that k is minimal, that is, for any $n < k$ and $t_n = \tau.a^n$, $Q_{t_n}^{L_1}(\tau^{\checkmark}(a^{\text{-}}, \ldots, a^{\text{-}})) = Q_{t_n}^{L_2}(\tau^{\checkmark}(a^{\text{-}}, \ldots, a^{\text{-}}))$. Let j be an integer in $0 \ldots k$. W.l.o.g, we suppose that $obs = \tau^{\checkmark}(\underbrace{a^{\checkmark}, \ldots, a^{\checkmark}}_{j}, \underbrace{a^{\times}, \ldots, a^{\times}}_{k-j})$.

Hence, by Lemma 5.3

$$\begin{aligned}
Q_t^{L_1}(obs) &= \sum_{s \in \mathcal{S}_1} P_{\mathsf{i}_1 \to s}(\tau) P_{s \to \mathcal{S}_1}(a)^j (1 - P_{s \to \mathcal{S}_1}(a))^{k-j} \\
&= \sum_{s \in \mathcal{S}_1} P_{\mathsf{i}_1 \to s}(\tau) P_{s \to \mathcal{S}_1}(a)^j - c_1 \sum_{s \in \mathcal{S}_1} P_{\mathsf{i}_1 \to s}(\tau) P_{s \to \mathcal{S}_1}(a)^{j+1} \\
&\qquad + c_2 \sum_{s \in \mathcal{S}_1} P_{\mathsf{i}_1 \to s}(\tau) P_{s \to \mathcal{S}_1}(a)^{j+2} - \\
&\qquad \vdots \\
&\qquad (-1)^{k-j} \sum_{s \in \mathcal{S}_1} P_{\mathsf{i}_1 \to s}(\tau) P_{s \to \mathcal{S}_1}(a)^k
\end{aligned}$$

where $c_1, c_2, \ldots$ are integer constants. By the minimality of k, we have :

$$\begin{aligned}
Q_t^{L_1}(obs) - Q_t^{L_2}(obs) &= (-1)^{k-j} \Big(\sum_{s \in \mathcal{S}_1} P_{\mathsf{i}_1 \to s}(\tau) P_{s \to \mathcal{S}_1}(a)^k - \sum_{s \in \mathcal{S}_2} P_{\mathsf{i}_2 \to s}(\tau) P'_{s \to \mathcal{S}_2}(a)^k \Big) \\
&= (-1)^{k-j} \big(Q_t^{L_1}(\tau^{\checkmark} a^{k^{\checkmark}}) - Q_t^{L_2}(\tau^{\checkmark} a^{k^{\checkmark}}) \big).
\end{aligned}$$

Hence, $Q_t^{L_1}(\tau^{\checkmark} a^{k^{\checkmark}}) \neq Q_t^{L_2}(\tau^{\checkmark} a^{k^{\checkmark}})$. Since $\tau^{\checkmark} a^{k^{\checkmark}} \in \Theta(\tau.a^k)$, the test $\tau.a^k$ yields different probability distributions on observations according to Θ. □

5.4 Moments Coincidence

Now that we have a testing equivalence, we are interested in providing a structural equivalence on LMPs that corresponds to it. In this section we show that the grammar

$\mathcal{T}_{Kmoment^0}$ characterizes an equivalence notion based on the coincidence of moments of a random variable, as its name suggests.

Definition 5.5. Let $(\mathcal{S}, \mathsf{i}, \mathcal{A}, P)$ be an LMP, $a \in \mathcal{A}$ and $\tau \in \mathcal{A}^*$. We define $X_{\tau,a} : \mathcal{S} \cup \{Dead\} \to [0,1]$ the random variable representing the probability to perform action a after having run trace τ. *Dead* is the outcome of the experiment of failing to perform τ. More precisely,

$$\begin{array}{rcll} \{X_{\tau,a} = p\} & = & \{s : P_{s\to\mathcal{S}}(a) = p\} & \text{for any } p \in]0,1] \\ \{X_{\tau,a} = 0\} & = & \{s : P_{s\to\mathcal{S}}(a) = 0\} \cup \{Dead\} & \end{array}$$

and

$$\begin{array}{rcll} Pr(X_{\tau,a} = p) & = & P_{\mathsf{i}\to\{s:P_{s\to\mathcal{S}}(a)=p\}}(\tau) & \text{for any } p \in]0,1] \\ Pr(X_{\tau,a} = 0) & = & P_{\mathsf{i}\to\{s:P_{s\to\mathcal{S}}(a)=0\}}(\tau) + (1 - P_{\mathsf{i}\to\mathcal{S}}(\tau)). & \end{array}$$

Remark 5.6. *Note that $Pr(X_{\tau,a}=0) = 1 - Pr(X_{\tau,a}>0)$, as wanted. This implies that the random variable $X_{\tau,a}$ is completely determined by the the values of $P_{\mathsf{i}\to\{s:P_{s\to\mathcal{S}}(a)=p\}}(\tau)$ for $p > 0$.*
On the other hand, the knowledge of the distribution of all random variables $\{X_{\tau,a}\}_{\tau\in\mathcal{A}^*,a\in\mathcal{A}}$ completely determines the values of $P_{\mathsf{i}\to\{s:P_{s\to\mathcal{S}}(a)=p\}}(\tau)$ for $p \geq 0, \tau \in \mathcal{A}^*, a \in \mathcal{A}$. To prove this assertion we only have to show that this is the case for $p = 0$. This follows from a straightforward induction, based on the facts that (1) $P_{\mathsf{i}\to\mathcal{S}}(\epsilon) = 1$ and (2) the expected value of $X_{\tau,a}$ is exactly $P_{\mathsf{i}\to\mathcal{S}}(\tau a)$.

At first sight, it is surprising to see a random variable taking probability values, but recall that we are performing tests on processes and are indeed observing the probabilities that these tests are accepted.

Recall that the i^{th} moment of a random variable X is the expected value of X^i (written $E(X^i)$) and that the expected value of a random variable X is the sum of the probabilities of each possible outcome of the experiment multiplied by its payoff ($E(X) = \sum_i x_i P(X = x_i)$). In the light of this remark, we introduce a family of equivalence notions that correspond to coincidence between the random variables ($X^{L_1}_{\tau,a}$ and $X^{L_2}_{\tau,a}$) up to the k^{th} moment. For that reason, we call them *K-moment* equivalences.

Definition 5.7. *Let $K \in \mathbb{N}^*$. Two LMPs $L_1 = (\mathcal{S}_1, \mathsf{i}_1, \mathcal{A}, P)$ and $L_2 = (\mathcal{S}_2, \mathsf{i}_2, \mathcal{A}, P')$ are K-moment equivalent, if and only if, $\forall\, \tau \in \mathcal{A}^*, \forall\, a \in \mathcal{A}$, $X^{L_1}_{\tau,a}$ and $X^{L_2}_{\tau,a}$ have exactly the same first K moments. That is,*

$$E((X^{L_1}_{\tau,a})^k) = E((X^{L_2}_{\tau,a})^k) \text{ for } k \leq K,$$

or equivalently,

$$\sum_{s\in\mathcal{S}_1} P_{\mathsf{i}_1\to s}(\tau)\ (P_{s\to\mathcal{S}_1}(a))^k = \sum_{s\in\mathcal{S}_2} P_{\mathsf{i}_2\to s}(\tau)\ (P'_{s\to\mathcal{S}_2}(a))^k$$

for $k \leq K$.

We will discuss later the relative value of K and the equivalence obtained at the limit, that is, as $K \longrightarrow \infty$. The following theorem states that *K-moment* equivalence coincides with $\mathcal{T}_{Kmoment^0}$.

Theorem 5.8. *Two LMPs are K-moment equivalent if, and only if, they yield the same probability distribution on observations of tests generated from $\mathcal{T}_{Kmoment^0}$.*

Proof. ($\Rightarrow$). By contradiction. Let $L_1 = (\mathcal{S}_1, \mathsf{i}_1, \mathcal{A}, P)$ and $L_2 = (\mathcal{S}_2, \mathsf{i}_2, \mathcal{A}, P')$ be two LMPs. Let $t = \tau.a^k$ a test of $\mathcal{T}_{Kmoment^0}$ yielding different probability distribution on observations. Suppose, w.l.o.g, that t is minimal, that is, $\forall obs \in \Theta_\tau$, $Q_\tau^{L_1}(obs) = Q_\tau^{L_2}(obs)$. Hence, we have, $Q_t^{L_1}(\tau^\checkmark a^{k^\checkmark}) \neq Q_t^{L_2}(\tau^\checkmark a^{k^\checkmark})$ which implies that

$$\sum_{s \in \mathcal{S}_1} P_{\mathsf{i}_1 \to s}(\tau)\ (P_{s \to \mathcal{S}_1}(a))^k \neq \sum_{s \in \mathcal{S}_2} P_{\mathsf{i}_2 \to s}(\tau)(P'_{s \to \mathcal{S}_2}(a))^k$$

which in turn implies $E((X_{\tau,a}^{L_1})^k) \neq E((X_{\tau,a}^{L_2})^k)$. By Definition 5.7, we can conclude that L_1 and L_2 are not *K-moment* equivalent.
($\Leftarrow$). Straightforward since $Q_t(\tau^\checkmark a^{k^\checkmark}) = E((X_{\tau,a})^k)$. □

Since $\mathcal{T}_{1\text{-}moment}$ is exactly $\mathcal{T}_{trace}$, Theorem 5.8 and Theorem 2.8 imply the following Corollary.

Corollary 5.9. 1*-moment equivalence and trace equivalence coincide.*

5.5 The Test Grammar $\mathcal{T}_{Kmoment}$

As announced above, we now introduce a second test language that characterizes *K-moment* equivalence. We present it because it has a more homogenous form than $\mathcal{T}_{Kmoment^0}$ and because it fits in the generic framework we present in the next chapter. The test language is called $\mathcal{T}_{Kmoment}$ and its definition is given with function Θ.

Definition 5.10. *Let $K \in \mathbb{N}^*$. For $k \leq K$, define:*

$$\mathcal{T}_{Kmoment} : \qquad t ::= \omega \mid a^k.t$$

where the test $a^k.t$ consists in running action a on k copies of the current state and if the last action succeeds, proceed with test t on the last copy (and delete the other copies). The observation set for $a^k.t$ according to Θ is:

$$\Theta_{a^k.t} = \{a^{k^\times}\} \cup \{a^{k^\times} e \mid e \in \Theta_t\} \cup \{a^{k^\checkmark} e \mid e \in \Theta_t\}.$$

The probability distribution on observations for $a^k.t$ is:

- $Q^s_{a^k.t}(a^{k^\times}) = 1 - P_{s\to\mathcal{S}}(a)^k$
- $Q^s_{a^k.t}(a^{k^\times}e) = (1 - P_{s\to\mathcal{S}}(a)^{k-1}) \sum_{s'\in S} P_{s\to s'}(a) Q^{s'}_t(e)$ *where* $e \in \Theta_t$
- $Q^s_{a^k.t}(a^{k^\checkmark}e) = P_{s\to\mathcal{S}}(a)^{k-1} \sum_{s'\in\mathcal{S}} P_{s\to s'}(a) Q^{s'}_t(e)$ *where* $e \in \Theta_t$.

Observation $a^{2^\times}$ means that the second occurrence of a failed (and possibly the first one also) whereas the observation $a^{2^\times}e$ means that only the second occurrence of a succeeded. For example, the set of observations resulting from test $a^2.t$ is: $\Theta_{a^2.t} = \{a^{2^\checkmark}e, a^{2^\times}, a^{2^\times}e \mid e \in O_t\}$.

$\mathcal{T}_{Kmoment}$ differs from $\mathcal{T}_{Kmoment^0}$ by allowing tests to continue after an action of the form a^k if the last occurrence of a (the execution of a on the k^{th} copy) succeeds. This last occurrence decides about the transition and is called a *transition action*. The following theorem shows that both define the same equivalence. However, $\mathcal{T}_{Kmoment}$ is more efficient in the sense that a difference between the processes can be detected before the last step of the test. This implies that the convergence of the values when computing the corresponding divergence will happen faster but also that the returned value will be different. To understand intuitively the difference, consider two processes that differ by tiny values at every step of a long trace. The fact that $\mathcal{T}_{Kmoment}$ has the ability to detect the difference at every step yields that the small values will be added up and hence return a bigger value than tests of $\mathcal{T}_{Kmoment^0}$ that will only use the last action to compute the divergence.

The following theorem states that both test grammars $\mathcal{T}_{Kmoment^0}$ and $\mathcal{T}_{Kmoment}$ characterize *K-moment* equivalence.

Theorem 5.11. *Let $K \in \mathbb{N}^*$. Two LMPs yield the same probability distribution on observations of tests generated from $\mathcal{T}_{Kmoment^0}$ if, and only if, they yield the same probability distributions on observations of tests generated from $\mathcal{T}_{Kmoment}$.*

Proof. ($\Rightarrow$). By contradiction. Let $L_1 = (\mathcal{S}_1, \mathsf{i}_1, \mathcal{A}, P)$ and $L_2 = (\mathcal{S}_2, \mathsf{i}_2, \mathcal{A}, P')$ be two LMPs. Let t be a test of $\mathcal{T}_{Kmoment}$ ($t = a_1^{k_1}.a_2^{k_2}.\ \ldots\ .a_n^{k_n}$) which yields different probability distributions on observations for L_1 and L_2, with n as small as possible. The proof consists in showing that there exists a test t of $\mathcal{T}_{Kmoment^0}$ that yields different probability distributions over observations. Let $Pr^{L_1}(t)$ be the probability that t is accepted in L_1. That is,

$$Pr^{L_1}(t) = Q^{L_1}_t(a_1^{k_1^\checkmark}\, a_2^{k_2^\checkmark}\ \ldots\ a_n^{k_n^\checkmark}).$$

W.l.o.g., we may suppose that

$$Pr^{L_1}(a_1^{k_1^\checkmark}\, a_2^{k_2^\checkmark}\ \ldots\ a_n^{k_n^\checkmark}) \neq Pr^{L_2}(a_1^{k_1^\checkmark}\, a_2^{k_2^\checkmark}\ \ldots\ a_n^{k_n^\checkmark}).$$

Thus,

$$\begin{aligned} &Pr^{L_1}(a_1^{k_1^\checkmark})Pr^{L_1}(a_2^{k_2^\checkmark}|a_1^{k_1^\checkmark})\ldots Pr^{L_1}(a_n^{k_n^\checkmark}|a_1^{k_1^\checkmark}\, a_2^{k_2^\checkmark}\ \ldots\ a_{n-1}^{k_{n-1}^\checkmark}) \\ &\quad \neq Pr^{L_2}(a_1^{k_1^\checkmark})Pr^{L_2}(a_2^{k_2^\checkmark}|a_1^{k_1^\checkmark})\ldots Pr^{L_2}(a_n^{k_n^\checkmark}|a_1^{k_1^\checkmark}\, a_2^{k_2^\checkmark}\ \ldots\ a_{n-1}^{k_{n-1}^\checkmark}). \end{aligned}$$

By the minimality of n, we therefore have

$$Pr^{L_1}(a_n^{k_n^\checkmark}|a_1^{k_1^\checkmark}\, a_2^{k_2^\checkmark}\ \ldots\ a_{n-1}^{k_{n-1}^\checkmark}) \neq Pr^{L_2}(a_n^{k_n^\checkmark}|a_1^{k_1^\checkmark}\, a_2^{k_2^\checkmark}\ \ldots\ a_{n-1}^{k_{n-1}^\checkmark}).$$

Which, by the definition of $a_i^{k_i^\checkmark}$, implies

$$\begin{aligned} &Pr^{L_1}(a_n^{k_n^\checkmark}|a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark) \neq Pr^{L_2}(a_n^{k_n^\checkmark}|a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark) \\ &\Rightarrow \frac{Pr^{L_1}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark a_n^{k_n^\checkmark})}{Pr^{L_1}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark)} \neq \frac{Pr^{L_2}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark a_n^{k_n^\checkmark})}{Pr^{L_2}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark)} \\ &\Rightarrow Pr^{L_1}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark a_n^{k_n^\checkmark}) \neq Pr^{L_2}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark a_n^{k_n^\checkmark}) \\ &\Rightarrow Q_t^{L_1}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark a_n^{k_n^\checkmark}) \neq Q_t^{L_2}(a_1^\checkmark\, a_2^\checkmark\ \ldots\ a_{n-1}^\checkmark a_n^{k_n^\checkmark}). \end{aligned}$$

Since $a_1.a_2\ \ldots\ a_{n-1}.a_n^{k_n}$ is a test of $\mathcal{T}_{Kmoment^0}$, we have the result.
($\Leftarrow$). Follows from the fact that all the tests generated from $\mathcal{T}_{Kmoment^0}$ are also tests generated from $\mathcal{T}_{Kmoment}$. □

5.6 The Value of K at the Limit

Within the family of *K-moment* equivalences, the bigger is the value of K, the more discriminating is the corresponding equivalence notion. The example of Figure 5.1 shows how 2-moment is more discriminating than 1-moment. The example of Figure 5.2 shows how 3-moment is more discriminating than 2-moment. Indeed, the two processes are 1-moment (trace) equivalent, 2-moment equivalent but not 3-moment equivalent – the test aa^3 distinguishes between them.

What happens if we let K tends to the limit? By observing the test grammar of $\mathcal{T}_{Kmoment}$, we can conclude that, even if it remains RRF, it will permit unbounded numbers of replicas at a particular state. This is not practical for testing, but for theoretical purpose, it is interesting to observe that the limit equivalence has a natural

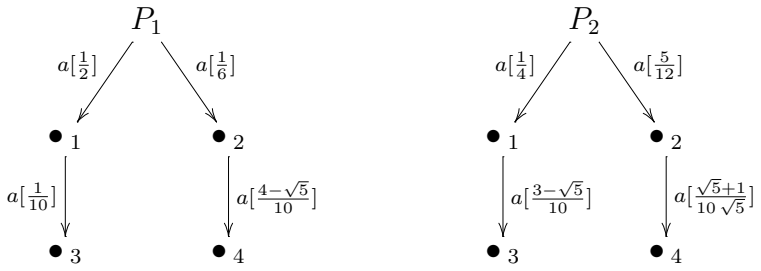


Figure 5.2: 2-moment equivalent but not 3-moment equivalent

form and that it is strictly weaker than bisimulation. In order to define this limit equivalence, we take advantage of a well known alternative definition of bisimulation.

Bisimulation can be formulated as the greatest fixed point of a particular function F which is defined as follows. Given an equivalence relation R, we say that two states s_1, s_2 are $F(R)$ equivalent if they have the same probability to jump to an equivalence class of R for every sequence of actions. More formally

$$P_{s_1 \to C}(\tau) = P_{s_2 \to C}(\tau)$$

for all R-equivalence class C and all trace τ. Bisimulation is $\cap_{i \in \mathbb{N}} F^i(\mathcal{S} \times \mathcal{S})$ and trace equivalence is simply $F(\mathcal{S} \times \mathcal{S})$. Hence, in order to naturally define an equivalence whose discrimination power is between bisimulation and trace equivalence, one could try to adjust the parameters that distinguish these two equivalences. The limit equivalence we are looking for can be defined that way.

Consider $\cap_{a \in \mathcal{A}} F(\sim_a)$ where $\sim_a$ is the equivalence that identifies states that have the same total probability to perform action a, $P_{s \to \mathcal{S}}(a)$. This equivalence discriminates between T_1 and T_2 of Figure 5.1 and any two trace equivalent processes where probabilistic choices happen at different levels. To prove that this equivalence corresponds to K-equivalence for every K, we will use the fact that moments coincidence corresponds to equality between the associated random variables.

Theorem 5.12. *Two LMPs L_1 and L_2 with the same action set $\mathcal{A}$ are $\cap_{a \in \mathcal{A}} F(\sim_a)$-equivalent if, and only if, for any trace $\tau \in \mathcal{A}^*$ and action $a \in \mathcal{A}$, the random variables $X^{L_1}_{\tau,a}$ and $X^{L_2}_{\tau,a}$ are identically distributed.*

Proof. Fix $L_1 = (\mathcal{S}_1, \mathsf{i}_1, \mathcal{A}, P_1)$, $L_2 = (\mathcal{S}_2, \mathsf{i}_2, \mathcal{A}, P_2)$, and let $C_i^{p,a} = \{s : P_{s \to \mathcal{S}_i}(a) = p\}$, $i = 1, 2$. It follows from Definition 5.5 and Remark 5.6 that *all* the random variables $X^{L_1}_{\tau,a}$ and $X^{L_2}_{\tau,a}$ are identically distributed if and only if

$$P_{\mathsf{i}_1 \to C_1^{p,a}}(\tau) = P_{\mathsf{i}_2 \to C_2^{p,a}}(\tau) \qquad \forall\, a \in \mathcal{A}, \forall \tau \in \mathcal{A}^*, \forall p \in [0,1].$$

Now, by definition of $\sim_a$, we have

$$s_1 \sim_a s_2 \quad \Leftrightarrow \quad P_{s_1 \to \mathcal{S}_1}(a) = P_{s_2 \to \mathcal{S}_2}(a).$$

Thus, by the definition of the function F, we have

$$\mathsf{i}_1 \; F(\sim_a) \; \mathsf{i}_2 \quad \Leftrightarrow \quad P_{\mathsf{i}_1 \to C_1^{p,a}}(\tau) = P_{\mathsf{i}_2 \to C_2^{p,a}}(\tau) \quad \forall \tau \in \mathcal{A}^*, \forall p \in [0,1].$$

We therefore have,

$$\begin{aligned} \mathsf{i}_1 \; \cap_{a \in \mathcal{A}} F(\sim_a) \; \mathsf{i}_2 \quad &\Leftrightarrow \quad P_{\mathsf{i}_1 \to C_1^{p,a}}(\tau) = P_{\mathsf{i}_2 \to C_1^{p,a}}(\tau) \qquad \forall\, a \in \mathcal{A}, \forall \tau \in \mathcal{A}^*, \forall p \in [0,1] \\ &\Leftrightarrow \quad Pr(X_{\tau,a}^{L_1} = p) = Pr(X_{\tau,a}^{L_2} = p) \qquad\qquad \forall \tau \in \mathcal{A}^*, \forall p \in [0,1]. \end{aligned}$$

□

We thus have a correspondence between *K-moment* equivalence and $\cap_{a \in \mathcal{A}} F(\sim_a)$, as stated in the following corollary.

Corollary 5.13. *Two LMPs are $\cap_{a \in \mathcal{A}} F(\sim_a)$ equivalent if, and only if, they are K-moment equivalent for every K.*

Proof. Since all the moments of the random variable $X_{\tau,a}^{L_1}$ are always between 0 and 1, and since we restrict to countable LMPs, the uniqueness of the Hausdorff moment problem (see [1]) implies that the random variables $X_{\tau,a}^{L_1}$ and $X_{\tau,a}^{L_2}$ are identically distributed if and only if all their moments coincide. Definition 5.5 and Theorem 5.12 imply the result. □

5.7 A Logical Characterization

Recall that a logical characterization of an equivalence notion requires the definition of a modal logic such that two processes are equivalent if, and only if, they satisfy the same formulae of the logic. The equivalence $\cap_{a \in \mathcal{A}} F(\sim_a)$ can be characterized by the following modal logic.

Definition 5.14.

$$\begin{aligned} \mathcal{L}_{k-limit}: \qquad F &:= \langle \tau \rangle_p \, \phi \, tt \\ \phi &:= \langle a \rangle_q \end{aligned}$$

where $\tau \in \mathcal{A}^*$, $a \in \mathcal{A}$, and $p, q \in [0,1]$. A formula $\langle \tau \rangle_p \, \phi \, tt$ is satisfied by any state that accepts trace τ with probability at least p then end up in a state satisfying ϕ whereas $\langle a \rangle_q$ is satisfied by any state that accepts action a with a probability equal to q.

In the example of Figure 5.1, process T_1 satisfies the formula $\langle ca\rangle_1 \langle b\rangle_{\frac{1}{3}} tt$ whereas T_2 does not.

Theorem 5.15. *Two states s_1 and s_2 are $\cap_{a\in\mathcal{A}} F(\sim_a)$-equivalent if, and only if, they satisfy exactly the same formulae of the logic $\mathcal{L}_{k-limit}$.*

Proof. ($\Rightarrow$). By contradiction. Suppose that s_1 and s_2 do not satisfy the same set of formulae of $\mathcal{L}_{k-limit}$. Let $f = \langle\tau\rangle_p \langle a\rangle_q tt$ a formula of $\mathcal{L}_{k-limit}$ and assume, w.l.o.g., that it is satisfied by s_1 but not by s_2 ($s_1 \vDash \langle\tau\rangle_p \langle a\rangle_q tt$ and $s_2 \nvDash \langle\tau\rangle_p \langle a\rangle_q tt$). By Definition 5.14 this implies:

$$P_{s_1\to\{s:P_{s\to}\mathcal{S}(a)=q\}}(\tau) \geq p > P_{s_2\to\{s:P_{s\to}\mathcal{S}(a)=q\}}(\tau).$$

By Definition 5.5, we have

$$Pr(X^{s_1}_{\tau,a} = q) \geq p > Pr(X^{s_2}_{\tau,a} = q)$$

which implies that $X^{s_1}_{\tau,a}$ and $X^{s_2}_{\tau,a}$ are not identically distributed. Theorem 5.12 implies that s_1 and s_2 are not $\cap_{a\in\mathcal{A}} F(\sim_a)$-equivalent.
($\Leftarrow$). By contradiction. Suppose that s_1 and s_2 are not $\cap_{a\in\mathcal{A}} F(\sim_a)$-equivalent. Theorem 5.12 implies that $\exists\tau \in \mathcal{A}^*, \exists a \in \mathcal{A}$, such that $X^{s_1}_{\tau,a}$ $X^{s_2}_{\tau,a}$ are not identically distributed. By Definition 5.5, we may assume, w.l.o.g., that $\exists q, p \in [0,1]$, such that

$$Pr(X^{s_1}_{\tau,a} = q) = p > Pr(X^{s_2}_{\tau,a} = q).$$

Then, it is easy to verify that the formula $f = \langle\tau\rangle_p \langle a\rangle_q tt$ of $\mathcal{L}_{k-limit}$ is satisfied by s_1 but not by s_2. □

5.8 Conclusion

K-moment is a new family of equivalence notions that constitutes a good compromise between trace equivalence and bisimulation. In this chapter, we presented this family from different point of views, namely, testing, coincidence of moments, function F, and logical. In particular, we showed that *K-moment* can be characterized by a simple testing framework which is RRF (recursive replication free). We showed also that the observations set one can obtain as a result of the execution of a test t can be grouped in a certain pattern to obtain new observation functions. In the next chapter, we will take advantage of this grouping capability to propose a new and flexible framework for defining testable equivalences.

Chapter 6

Test-Observation-Equivalence (TOE)

6.1 Introduction

Testing is an efficient tool to characterize equivalence between stochastic processes, in particular when the models of the processes are not available. Typically, when equivalence notions are defined via testing, the main element to specify is the test grammar. However, as seen in the previous chapter, there is another element that can play an important role, that is: how observations are grouped. Indeed, generally, only some aspects of observations are significant. Grouping of "similar" observations will allow to focus on important aspects of observations. In this chapter, we propose an improved technique to define equivalence notions by putting more emphasis on the observation function which indicates how observations are grouped. Hence, this technique is based on specifying two elements: a test grammar and an observation function just as we have done for the particular case of *K-moment* in the previous chapter. The resulting equivalence notion is called, therefore, test-observation-equivalence, or shortly, TOE. This chapter starts by describing the two elements of a TOE. Then, to fix ideas, we give a detailed example of a TOE called Barb Acceptance. In the second part of this chapter, we show how the RL framework presented in Chapter 4 can be generalized in such a way to match any TOE and we provide the necessary formal proof in a generic form. Finally, we use the proposed framework to give a TOE definition of the equivalence notions presented in the previous chapters and we discuss their distinguishing capabilities by organizing them in a hierarchy.

6.2 Test Grammar

A test grammar specifies the form of tests that can be generated. Then, executing tests on a process allows to figure out which properties are satisfied on the process. In particular, in presence of a system allowing to create replicas of any of its states, an interesting test grammar would take advantage of this capability. Having this feature in mind, we propose a generic test grammar for any TOE as follows. Let $L = (\mathcal{S}, \mathsf{i}, \mathcal{A}, P)$ be an LMP, then:

$$\mathcal{T}_{Gen} : \qquad t ::= \omega \mid \alpha.t$$

where ω is a dummy action which does anything and α , called a *meta-action*, has the form $\alpha = Aa$ such that A is a tuple of actions of the form $\langle a_1, a_2, \ldots a_n \rangle$ and $a \in \mathcal{A}$. The idea behind the meta-action α is that, having several replicas of the current state, perform a set of experiments via the A-part of α and then execute the action a to transit to the next state. Therefore, the a-part of α is the transition action and is noted $\boldsymbol{trans}(\alpha)$.

Example 6.1. *Consider the test grammar:*

$$t ::= \omega \mid \{a_1, \ldots, a_n\}a.t$$

where the meta-action α has the form $\{a_1, \ldots, a_n\}a$. The curly brackets in the meta-action $\{a_1, \ldots, a_n\}a$ designate as usual a set of actions. That is, $\{a_1, \ldots, a_n\}$ does not include the same action twice and the order of the actions is not significant. The meta-action $\{a_1, \ldots, a_n\}a$ consists in executing actions $a_1, \ldots, a_n$ on n replicas of the current process, then executing the transition action a on another replica. If the transition action succeeds, move on to the next state and proceed with t.

The properties that the generated tests can check depend on the form of the meta-action α. For instance, in the previous example, the meta-action $\alpha = \{a_1, \ldots, a_n\}a$ checks if the same state accepts actions $a_1, \ldots, a_n$ and a. On the other hand, a meta-action $\alpha = a^k$ checks if a state accepts action a in k successive executions.

Note finally that each form of α corresponds to a different TOE and that all TOEs are RRF.

6.3 Grouping of Observations

Executing a test on a state may yield to different possible observations. From the observer point of view, it is not the observation itself which is relevant but a certain

aspect of it. For example, in the observation function of a^k in Definition 5.2, it is not the exact observation obtained which is significant for the observer, but whether all k executions of a succeed or not. Such aspect is generally shared by several observations. The idea is then to group observations according to these aspects. This will allow first to focus on the desired properties and second to evaluate tests more efficiently since the number of observations can be considerably optimized.

Let a be an action in $\mathcal{A}$ and s a state in $\mathcal{S}$. Running action a on state s produces one of two observations: ✓ or × (noted also $a^{\checkmark}$ and $a^{\times}$). Hence $O_a = \{\checkmark, \times\}$. As defined above, the meta-action α can be composed of different actions or instances of the same action and hence, it may produce more observations. For example, $\{a_1, \ldots, a_n\}a$ in Example 6.1 may produce between 2 and 2^{n+1} observations. The number of observations depends on whether observations are grouped or not. On one hand, not grouping observations will produce the following observation set for action $\{a_1, \ldots, a_n\}a$:

$$\begin{aligned} O_{\{a_1,\ldots,a_n\}a} &= O_{a_1} \times \ldots \times O_{a_n} \times O_a \\ &= \{a_1^{\checkmark}, a_1^{\times}\} \times \ldots \times \{a_n^{\checkmark}, a_n^{\times}\} \times \{a^{\checkmark}, a^{\times}\} \\ &= \{\checkmark, \times\}^{n+1}. \end{aligned} \tag{6.1}$$

On the other hand, by extensively grouping observations we may end-up with the following observation set:

$$\begin{aligned} \Theta_{\{a_1,\ldots,a_n\}a} &= \left\{\{a_1, \ldots, a_n\}a^{\checkmark}, \{a_1, \ldots, a_n\}a^{\times}\right\} \\ &= \{\checkmark, \times\} \end{aligned} \tag{6.2}$$

where the observation $\{a_1, \ldots, a_n\}a^{\checkmark}$ means that all actions succeeded and the observation $\{a_1, \ldots, a_n\}a^{\times}$ means that at least one of the actions $a_1, \ldots, a_n, a$ has failed.

The observation functions O and Θ are both extreme cases. Indeed, the observation function O yields to maximal observation sets ($|O_{\{a_1,\ldots,a_n\}a}| = 2^{n+1}$), whereas the observation function Θ yields to minimal observation sets ($|\Theta_{\{a_1,\ldots,a_n\}a}| = 2$). However, between these two extreme cases, there are other observation functions that can be defined. For example, it is possible to group observations according to the number of succeeded actions among $a_1, \ldots, a_n, a$: observations with 0 successes, observations with 1 success, etc. This will yield to $n + 2$ possible observations.

Notice that the observation set $\Theta_{\{a_1,\ldots,a_n\}a}$ (Equation (6.2)) is a partition of the observation set $O_{\{a_1,\ldots,a_n\}a}$ (Equation (6.1)) where there are only two parts. One part, denoted $\{a_1, \ldots, a_n\}a^{\checkmark}$ or ✓, contains only one element of O_α which is $(a_1^{\checkmark}, a_2^{\checkmark}, \ldots, a_n^{\checkmark}, a^{\checkmark})$. The second part, denoted $\{a_1, \ldots, a_n\}a^{\times}$ or ×, contains the remaining elements of O_α.

Generalizing this discussion to $\mathcal{T}_{Gen}$, we define O as the observation function without grouping.

Definition 6.2. *Let $\alpha = Aa$, then*

- $O_\omega = \{\omega^\checkmark\}$
- $O_{\alpha.t} = O_A \times \{a^\times\} \cup O_A \times \{a^\checkmark\} \times O_t$

where $O_A = \{\checkmark, \times\}^{|A|}$.

Other observation functions can be defined by partitioning the sets of observations defined by O. The way of partitioning must satisfy two properties. The first is that the observations set of a given meta-action α should always be grouped according the same pattern but two different meta-actions α_1 and α_2 can correspond to two different possible groupings. The second is that the grouping of a long test $t = \alpha_1.\alpha_2 \ldots \alpha_n$ must respect the grouping of each of its meta-actions $\alpha_1, \alpha_2, \ldots, \alpha_n$. This is what is captured by the following definition of generic observation function.

Definition 6.3. *Assume that ψ_α is a partition of O_α for every meta-action α. Then the observation function ψ which assigns a set of observations to every test of $\mathcal{T}_{\text{Gen}}$ is defined as follows.*

- $\psi_\omega = \{\omega^\checkmark\}$
- $\psi_{\alpha.t} = \psi_\alpha^\times \cup (\psi_\alpha^\checkmark \times \psi_t)$ *if* $t \neq \omega$

where $\psi_\alpha^\times \cup \psi_\alpha^\checkmark$ is a refinement of ψ_α such that observations whose transition action has failed (in $\psi_\alpha^\times$) are separated from observations whose transition action has not failed (in $\psi_\alpha^\checkmark$).

The refinement of ψ_α into $\psi_\alpha^\times$ and $\psi_\alpha^\checkmark$ is just to reflect the following fact. It may happen that two observations be grouped even if different events happened on the transition action. It is the case for our $\mathcal{T}_{Kmoment}$ observation function Θ. Indeed, consider test $a^2.t$: the two observations $(a^\checkmark, a^\times)$ and $(a^\times, a^\checkmark)$ are grouped together if $t = \omega$ (it is considered a failure) but otherwise, the case where the transition action succeeds is separated from the case where it does not. This is hidden in the notation $a^{2^\times}$ and $a^{2^\times}e$ of $\Theta_{a^2.t} = \{a^{2^\checkmark}e, a^{2^\times}, a^{2^\times}e \mid e \in O_t\}$.

This definition guarantees that the observation function ψ_α is consistent for every meta-action, but it allows to not treat different meta-actions α_1 and α_2 (ψ_{α_1} and ψ_{α_2} can be different groupings). Let us see a quick example of a less structured one in the sense that meta-actions with the same length will be grouped the same way. Consider the following observation function, λ, which groups observations according to the number of successful actions.

$$
\begin{aligned}
|\alpha| = 1 \quad &\Rightarrow \quad \lambda_\alpha = \{ \underbrace{\{\checkmark\}}_{\text{1 success}} , \underbrace{\{\times\}}_{\text{0 success}} \} \\
|\alpha| = 2 \quad &\Rightarrow \quad \lambda_\alpha = \{ \underbrace{\{(\checkmark,\checkmark)\}}_{\text{2 successes}}, \underbrace{\{(\checkmark,\times),(\times,\checkmark)\}}_{\text{1 success}}, \underbrace{\{(\times,\times)\}}_{\text{0 success}} \} \\
|\alpha| = 3 \quad &\Rightarrow \quad \lambda_\alpha = \{ \underbrace{\{(\checkmark,\checkmark,\checkmark)\}}_{\text{3 successes}}, \underbrace{\{(\checkmark,\checkmark,\times),(\checkmark,\times,\checkmark),(\times,\checkmark,\checkmark)\}}_{\text{2 successes}}, \\
&\qquad \underbrace{\{(\checkmark,\times,\times),(\times,\checkmark,\times),(\times,\times,\checkmark)\}}_{\text{1 success}}, \underbrace{\{(\times,\times,\times)\}}_{\text{0 success}} \} \\
&\vdots
\end{aligned}
$$

Another observation function could group the observations of meta-actions of even length according to their number of successful actions, and, for odd length meta-actions, group observations in two parts as in Equation (6.2). One could even think also of observations functions that do not treat similarly actions of the same length. An example would be to observe the failure or success of an action a in a meta-action α only if an action b is present and successful in α.

Combining a test grammar $\mathcal{T}$ and a suitable observation function ψ produces a TOE. By suitably tuning these two elements, it is possible to define a large range of equivalence notions. Among these, one can find trace and K-*moment* equivalences. To further fix ideas, let us detail the definition of another TOE.

6.4 A TOE Example: Barb Acceptance Equivalence

In this section, we revisit Example 6.1 and we give a complete definition of the corresponding TOE. The TOE is called Barb acceptance and is a testable variant of Barb acceptance equivalence [35] defined in Section 2.3.5. Recall that two processes are Barb acceptance equivalent if, and only if, they accept all Barb traces with the same probability. A Barb trace has the form $A_0 a_0 A_1 a_1 A_2 a_2 \ldots A_n a_n$ where A_i is a set of actions.

Process L accepts this Barb trace with probability p if L initially has acceptance set A_0 and it has probability p to perform the trace $a_0 a_1 \ldots a_{n-1}$ by going through states that have, respectively, acceptance sets $A_1, A_2, \ldots, A_n$.

Definition 6.4. *The Barb acceptance TOE is a couple $(\mathcal{T}_{\text{barbAcc}}, \psi)$ such that:*

$$\mathcal{T}_{\text{barbAcc}} \qquad t ::= \omega \mid \{a_1, \ldots, a_n\}a.t$$

and the observation function ψ is defined by partitioning the observations set $O_{\{a_1,\ldots,a_n\}a}$ as follows:

- $\psi_{\{a_1,\ldots,a_n\}a} = \left\{\{a_1, \ldots, a_n\}a^{\times}, \{a_1, \ldots, a_n\}a^{\checkmark}\right\}$
- $$\begin{aligned}\psi_{\{a_1,\ldots,a_n\}a.t} \quad = \quad & \left\{\{a_1, \ldots, a_n\}a^{\times}\right\} \\ & \cup \left\{\{a_1, \ldots, a_n\}a^{\times}\ e \mid e \in \psi_t\right\} \\ & \cup \left\{\{a_1, \ldots, a_n\}a^{\checkmark}\ e \mid e \in \psi_t\right\} \qquad \text{if } t \neq \omega\end{aligned}$$

where the observation $\{a_1, \ldots, a_n\}a^{\times}$ means that at least one of the actions $a_1, \ldots, a_n, a$ has failed and the observation $\{a_1, \ldots, a_n\}a^{\checkmark}$ means that all actions succeeded.

Since a TOE is by definition RRF, the MDP framework to estimate the trace equivalence divergence between LMPs (Section 4.3) can be extended to any TOE. This is the topic of the next section.

6.5 A Generic Framework

In Chapter 4, we introduced a new approach to estimate the divergence between stochastic systems without the need to know their internal structures. The approach was illustrated for the case of trace equivalence. In this section, we show that the approach can be generalized to any TOE $(\mathcal{T}, \psi)$. To this end, we reformulate the stochastic game, the MDP model as well as the theorems in terms of the generic meta-action α and the generic observation function ψ. Then, we give the formal proof of this generic framework.

6.5.1 The Stochastic Game

The stochastic game introduced in Section 4.2 is generalized as follows. Let $(\mathcal{T}, \psi)$ be a TOE.

Game$_{Gen}$: Put the three systems ("Impl", "Spec", and "Clone") in their initial states; then

Step 1 : The player chooses an action α and makes a prediction $\mathtt{Pred}$ on the outcome it may produce on "Spec".

Step 2 : Action α is run on the three processes.

Step 3 : If $\mathit{trans}(\alpha)$ succeeds on the three processes; go to Step 1. Else the game ends. In both cases the reward is computed as follows : (writing I for "Impl", C for "Clone" and Obs for observation)

$$R := \big(Obs.Spec) = \mathtt{Pred}\big)\big((Obs.I \neq \mathtt{Pred}) - (\mathtt{Pred} \neq Obs.C)\big)$$

where 0 and 1 are used as both truth values and numbers and $\mathtt{Pred}, Obs.Spec, Obs.I$, and $Obs.C$ are elements of ψ_α. This reward will be revealed to the player at the end of the episode.

Step 4 : Repeat until the episode ends on one of the three processes.

6.5.2 MDP Construction

The generic version of the MDP is very similar to the MDP of Definition 4.3. The state of states is still a set of traces where the trace is a sequence of meta-actions $\alpha_1, \alpha_2, \ldots, \alpha_n$. An action of the MDP is a meta-action α coupled with a prediction $\mathtt{Pred}$ on its outcome on "Spec" denoted $\alpha^{\mathtt{Pred}}$. The next-state probability distribution is the same as Definition 4.3 while the reward function is slightly modified.

Definition 6.5. *Let $(\mathcal{T}, \psi)$ be a TOE. Given the LMPs "Impl", "Spec" and "Clone" sharing the same action set $\mathcal{A}$, the induced MDP $\mathcal{M}$ is a tuple $(S, \mathsf{i}, Act, Pr^{\mathcal{M}}, R)$. The set of states of the MDP $\mathcal{M}$ is:*

$$S := \{\tau : \alpha_1\alpha_2\ldots\alpha_n \mid P^{Spec}(\tau) > 0\} \cup \{Dead\} \tag{6.3}$$

where $\alpha_i = A_i a_i$, A_i is a tuple of actions of $\mathcal{A}$ and $a_i \in \mathcal{A}$. The initial state is $\mathsf{i} = \epsilon$ (the empty sequence).

The set of actions of $\mathcal{M}$ is:

$$Act := \{\alpha^{\mathtt{Pred}} \mid \alpha = Aa \in tuple(\mathcal{A}) \times \mathcal{A} \wedge \mathtt{Pred} \in \psi_\alpha\}. \tag{6.4}$$

Suppose that $s = \tau$ *and* $\textbf{\textit{trans}}(\alpha) = a$*, then the next-state probability distribution* $Pr^{\mathcal{M}}$ *and the total immediate reward* R *functions are defined as follows:*

$$Pr^{\mathcal{M}}_{s\to s'}(\alpha^{\text{Pred}}) \;:=\; \begin{cases} P^{Spec}(a^{\checkmark}|s)\; P^{I}(a^{\checkmark}|s)\; P^{C}(a^{\checkmark}|s) & \text{if } s' = \tau\,\alpha \\ 1 - P^{Spec}(a^{\checkmark}|s)\; P^{I}(a^{\checkmark}|s)\; P^{C}(a^{\checkmark}|s) & \text{if } s' = Dead \\ 0 & \text{otherwise.} \end{cases}$$

$$R^{\alpha^{\text{Pred}}}_{s} \;:=\; P^{Spec}(\alpha^{\text{Pred}}|s)\Delta^{\alpha^{\text{Pred}}}_{s} \tag{6.5}$$

where $\bullet\ \Delta^{\alpha^{\text{Pred}}}_{s} := P^{C}(\alpha^{\text{Pred}}|\tau) - P^{I}(\alpha^{\text{Pred}}|\tau)$.

The definition of $R^{\alpha^{Pred}}_{s}$ is obtained from $Game_{Gen}$ in the same spirit as Equation (4.7) in Definition 4.3 except that $a^{\checkmark}$ and $a^{\times}$ are replaced, in this generic case, by $\alpha^{Pred}, \forall \textbf{\textit{Pred}} \in \psi_{\alpha}$. Indeed, according to the reward formula in step 3 of $Game_{Gen}$, the reward is computed only if the observation in "Spec" ($Obs.Spec$) matches the prediction ($\textbf{\textit{Pred}}$). Hence we get a (+1) reward when the observation in "Impl" is different from $\textbf{\textit{Pred}}$ ($Obs.I \neq \textbf{\textit{Pred}}$ or equivalently $Obs.I = \overline{\textbf{\textit{Pred}}}$) and the observation in "Clone" is the same as $\textbf{\textit{Pred}}$ ($Obs.C = \textbf{\textit{Pred}}$). On the other hand, we get a (-1) reward when $Obs.I = \textbf{\textit{Pred}}$ and $Obs.C \neq \textbf{\textit{Pred}}$. Thus,

$$\begin{aligned} R^{\alpha^{Pred}}_{s} &= P^{I}(\alpha^{\overline{Pred}}|s)\; P^{Spec}(\alpha^{Pred}|s)\; P^{C}(\alpha^{Pred}|s) - P^{I}(\alpha^{Pred}|s)\; P^{Spec}(\alpha^{Pred}|s)\; P^{C}(\alpha^{\overline{Pred}}|s) \\ &= P^{Spec}(\alpha^{Pred}|s)(P^{I}(\alpha^{\overline{Pred}}|s)P^{C}(\alpha^{Pred}|s) - P^{I}(\alpha^{Pred}|s)P^{C}(\alpha^{\overline{Pred}}|s)) \\ &= P^{Spec}(\alpha^{Pred}|s)\; \Delta^{\alpha^{Pred}}_{s}. \end{aligned}$$

6.5.3 Main Theorem and Definition of div(. ‖ .)

As for trace equivalence, for any TOE, the divergence between LMPs corresponds to the optimal value of the induced MDP.

Definition 6.6. Let "Impl" and "Spec" be two LMPs, E a TOE, and $\mathcal{M}$ their induced MDP. We define their E *equivalence divergence* as

$$\text{div}(\text{“Impl”} \| \text{“Spec”}) := V^{\star}(\mathsf{i}).$$

This definition is based on the following theorem.

Theorem 6.7. *Let $\mathcal{M}$ be the MDP induced by "Impl", "Spec", and "Clone". Let $E = (\mathcal{T}, \psi)$ be a TOE. If $\gamma < 1$ or $|\mathcal{M}| < \infty$ then $V^\star(\mathsf{i}) \geq 0$, and $V^\star(\mathsf{i}) = 0$, if and only if, "Impl" and "Spec" are equivalent according to* E.

The proof of Theorem 6.7 is given in the next section.

6.5.4 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 [51].

$$\begin{aligned} V^\pi(s) &= \sum_{\alpha^{\mathit{Pred}} \in Act} \pi(s, \alpha^{\mathit{Pred}}) \sum_{s' \in S} Pr^{\mathcal{M}}_{s \to s'}(\alpha^{\mathit{Pred}}) \; (R_s^{\alpha^{\mathit{Pred}}} + \gamma \; V^\pi(s')) \\ &= \sum_{\alpha^{\mathit{Pred}} \in Act} \pi(s, \alpha^{\mathit{Pred}}) (R_s^{\alpha^{\mathit{Pred}}} + Pr^{\mathcal{M}}_{s \to s.\alpha}(\alpha^{\mathit{Pred}}) \; \gamma V^\pi(s.\alpha)). \end{aligned} \tag{6.6}$$

Recall that a *test-policy* a deterministic policy that has the form $\pi = \alpha_1^{\mathit{Pred}_1} \alpha_2^{\mathit{Pred}_2} \ldots \alpha_{\mathrm{n}}^{\mathit{Pred}_{\mathrm{n}}}$. The value of π on i is thus determined as follows from Equation (6.6):

$$\begin{aligned} V^\pi(\mathsf{i}) &= \sum_{\alpha^{\mathit{Pred}} \in Act} \pi(\mathsf{i}, \alpha^{\mathit{Pred}}) \; (R_{\mathsf{i}}^{\alpha^{\mathit{Pred}}} + Pr^{\mathcal{M}}_{\mathsf{i} \to \mathsf{i}.\alpha}(\alpha^{\mathit{Pred}}) \; \gamma \; V^\pi(\mathsf{i}.\alpha)) \\ &= R_\epsilon^{\alpha_1^{\mathit{Pred}_1}} + Pr^{\mathcal{M}}_{\epsilon \to \alpha_1}(\alpha_1) \; \gamma \; \left(R_{\alpha_1}^{\alpha_2^{\mathit{Pred}_2}} + Pr^{\mathcal{M}}_{\alpha_1 \to \alpha_1\alpha_2}(\alpha_2) \; \gamma \; V^\pi(\alpha_1\alpha_2) \right) \\ &= R_\epsilon^{\alpha_1^{\mathit{Pred}_1}} + Pr^{\mathcal{M}}(\alpha_1) \; \gamma \; R_{\alpha_1}^{\alpha_2^{\mathit{Pred}_2}} + Pr^{\mathcal{M}}(\alpha_1\alpha_2) \; \gamma^2 \; R_{\alpha_1\alpha_2}^{\alpha_2^{\mathit{Pred}_2}} + \ldots \\ &= \sum_{i=0}^{n-1} Pr^{\mathcal{M}}(\alpha_0 \ldots \alpha_{\mathsf{i}}) \; \gamma^i \; R_{\alpha_0 \ldots \alpha_{\mathsf{i}}}^{\alpha_{\mathsf{i}+1}^{\mathit{Pred}_{\mathsf{i}+1}}} \end{aligned} \tag{6.7}$$

where $Pr^{\mathcal{M}}(\tau)$ is the probability that trace τ is accepted in the MDP, and α_0 denotes ϵ.

6.5.5 Proof of Theorem 6.7

We prove in this section that solving the MDP induced by "Spec" and "Impl" yields an equivalence divergence between them, i.e. the value is positive and it is 0, if and only if the two processes are equivalent. Let us first define this easy lemma which will be useful in this section.

Lemma 6.8. *Let ψ be an observation function, $\forall\, s \in S \backslash \{\mathrm{Dead}\}, \forall \alpha$,*

$$\textit{for} \quad \psi_\alpha = \{\mathit{Pred}_1, \ldots, \mathit{Pred}_k\}, \qquad \sum_{\mathrm{Pred} \in \psi_\alpha} \Delta_s^{\alpha^{\mathrm{Pred}}} = 0.$$

Proof.

$$\begin{aligned}\sum_{\mathit{Pred}\in\psi_\alpha} \Delta_s^{\alpha^{\mathit{Pred}}} &= \Delta_s^{\alpha^{\mathit{Pred}\,1}} + \ldots + \Delta_s^{\alpha^{\mathit{Pred}_k}} \\ &= \big(P^C(\alpha^{\mathit{Pred}\,1}|s) - P^I(\alpha^{\mathit{Pred}\,1}|s)\big) + \ldots + \big(P^C(\alpha^{\mathit{Pred}_k}|s) - P^I(\alpha^{\mathit{Pred}_k}|s)\big) \\ &= P^C(\alpha^{\mathit{Pred}\,1}|s) + \ldots + P^C(\alpha^{\mathit{Pred}_k}|s) - \big(P^I(\alpha^{\mathit{Pred}\,1}|s) + \ldots + P^I(\alpha^{\mathit{Pred}_k}|s)\big) \\ &= 1 - 1 \\ &= 0.\end{aligned}$$

□

In the remaining of the proof, $E = (\mathcal{T}, \psi)$ represents a fixed TOE.

Theorem 6.9. *Let "Impl" and "Spec" be two LMPs. Then the following are equivalent:*

(*i*) *"Impl" and "Spec" are equivalent according to* E.

(*ii*) $\forall \alpha^{\mathit{Pred}} \in \mathit{Act}, \forall \tau \in S\backslash\{\text{Dead}\}, \quad R_\tau^{\alpha^{\mathit{Pred}}} = 0$

(*iii*) $\forall$ *test-policy* π, $\quad V^\pi(\mathsf{i}) = 0$

Proof. (*i*) $\Rightarrow$ (*ii*). Since each state τ of $\mathcal{S}$ different from *Dead* is a trace, and since $P^{\mathit{Spec}}(\,.\,|\,.\,) = P^C(\,.\,|\,.\,)$,

$$\begin{aligned} P^{\mathit{Spec}}(\,.\,|\,.\,) = P^I(\,.\,|\,.\,) \;&\Leftrightarrow\; \forall \alpha^{\mathit{Pred}} \in \mathit{Act}, \tau \in S\backslash\{\mathit{Dead}\} \\ &\qquad P^C(\alpha^{\mathit{Pred}}|\tau) = P^I(\alpha^{\mathit{Pred}}|\tau) \\ &\Leftrightarrow\; \forall \alpha^{\mathit{Pred}} \in \mathit{Act}, \tau \in S\backslash\{\mathit{Dead}\} \\ &\qquad \Delta_\tau^{\alpha^{\mathit{Pred}}} = 0 \qquad \text{(by definition of } \Delta_\tau^{\alpha^{\mathit{Pred}}}\text{)} \\ &\Rightarrow\; \forall \alpha^{\mathit{Pred}} \in \mathit{Act}, \tau \in S\backslash\{\mathit{Dead}\} \\ &\qquad R_\tau^{\alpha^{\mathit{Pred}}} = 0 \quad \text{(by Equation (6.5)).}\end{aligned}$$

(*ii*) $\Rightarrow$ (*i*). Let $\tau \in S\backslash\{\mathit{Dead}\}$ be a state of the MDP. Let $\alpha^{\mathit{Pred}} \in \mathit{Act}$. By Equation (6.5), $R_\tau^{\alpha^{\mathit{Pred}}} = P^{\mathit{Spec}}(\alpha^{\mathit{Pred}}|\tau)\Delta_\tau^{\alpha^{\mathit{Pred}}} = 0$ implies that either $P^{\mathit{Spec}}(\alpha^{\mathit{Pred}}|\tau) = 0$ or $P^C(\alpha^{\mathit{Pred}}|\tau) = P^I(\alpha^{\mathit{Pred}}|\tau)$. We have to prove that the latter is true, $\forall\, \mathit{Pred} \in \psi_\alpha$. Hence, we must prove that if $P^{\mathit{Spec}}(\alpha^{\mathit{Pred}}|\tau) = 0$, then $P^I(\alpha^{\mathit{Pred}}|\tau) = 0$ as well. Let

$$\begin{aligned}\psi_\alpha^0 &:= \{\mathit{Pred} \in \psi_\alpha \,|\, P^{\mathit{Spec}}(\alpha^{\mathit{Pred}}|\tau) = 0\} \text{ and,} \\ \psi_\alpha^{\overline{0}} &:= \{\mathit{Pred} \in \psi_\alpha \,|\, P^{\mathit{Spec}}(\alpha^{\mathit{Pred}}|\tau) > 0\}.\end{aligned}$$

Notice that, $\psi_\alpha = \psi^0_\alpha \cup \psi^{\overline{0}}_\alpha$ and that $P^C(\alpha^{Pred}|\tau) = P^I(\alpha^{Pred}|\tau)$ for $\mathit{Pred} \in \psi^{\overline{0}}_\alpha$. The following equations imply that $P^I(\alpha^{Pred}|\tau)$ must be zero, $\forall\, \mathit{Pred} \in \psi^0_\alpha$:

$$\begin{aligned}
\sum_{Pred \in \psi^0_\alpha} P^I(\alpha^{Pred}|\tau) &= \sum_{Pred \in \psi_\alpha} P^I(\alpha^{Pred}|\tau) - \sum_{Pred \in \psi^{\overline{0}}_\alpha} P^I(\alpha^{Pred}|\tau) \\
&= \sum_{Pred \in \psi_\alpha} P^I(\alpha^{Pred}|\tau) - \sum_{Pred \in \psi^{\overline{0}}_\alpha} P^{Spec}(\alpha^{Pred}|\tau) \\
&= 1 - 1 \\
&= 0.
\end{aligned}$$

$(ii) \Rightarrow (iii)$ follows from Equation (6.7).
$(iii) \Rightarrow (ii)$. Fix $\alpha^{Pred} \in Act$ and $\tau \in S \backslash \{Dead\}$ such that $Pr^{\mathcal{M}}(s) > 0$, and let $n \geq 0$ such that $\alpha_1 \ldots \alpha_n = \tau$. Now, define $\pi = \alpha_1^{Pred_1} \ldots \alpha_n^{Pred_n}$. By (iii) and Equation (6.7), we have $0 = 0 - 0 = V^{\pi\alpha^{Pred}}(\mathsf{i}) - V^{\pi}(\mathsf{i}) = Pr^{\mathcal{M}}(\tau)\, \gamma^n\, R^{\alpha^{Pred}}_{\tau}$. Since $\gamma > 0$, if $Pr^{\mathcal{M}}(\tau) \neq 0$ then Equation (6.5) will imply the result. So, by way of contradiction suppose $Pr^{\mathcal{M}}(\tau) = 0$. This implies that $P^{\mathrm{I}}(\tau) = 0$ because, by definition of the MDP $\mathcal{M}$, $P^{\mathrm{Spec}}(\tau) \neq 0$ and because $P^{\mathrm{Spec}}(\tau) = P^{\mathrm{C}}(\tau)$. Now let $\alpha_1\alpha_{2\ldots}\alpha_{\mathrm{k}}$ be the smallest subsequence of τ such that $P^{\mathrm{I}}(\alpha_1\alpha_{2\ldots}\alpha_{\mathrm{k}}) = 0$. Since $P^{\mathrm{I}}(\epsilon) = 1$, $k \geq 1$. Fix $\tau' = \alpha_1\alpha_{2\ldots}\alpha_{\mathrm{k\text{-}1}}$, and observe that both τ' and $\tau'\alpha_{\mathrm{k}}$ are states of the MDP $\mathcal{M}$ because $P^{\mathrm{Spec}}(\tau') \neq 0$ and $P^{\mathrm{Spec}}(\tau'\alpha_{\mathrm{k}}) \neq 0$. By the hypothesis and Equation (6.7), we have $0 - 0 = V^{\alpha_1^{Pred_1}\alpha_2^{Pred_2}\ldots\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}}(\mathsf{i}) - V^{\alpha_1^{Pred_1}\alpha_2^{Pred_2}\ldots\alpha_{\mathrm{k\text{-}1}}^{Pred_{\mathrm{k\text{-}1}}}}(\mathsf{i}) = Pr^{\mathcal{M}}(\tau')\, \gamma^n\, R^{\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}}_{\tau'}$. Since, by construction, $Pr^{\mathcal{M}}(\tau') \neq 0$, we therefore have that $R^{\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}}_{\tau'} = 0$. It then follows from Equation (6.5) that $\Delta^{\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}}_{\tau'} = 0$, and hence that $P^I(\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}|\tau') = P^C(\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}|\tau')$. On an other hand, $P^I(\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}|\tau') = 0$, because $P^{\mathrm{I}}(\tau') \neq 0$ and $P^{\mathrm{I}}(\tau'\alpha_{\mathrm{k}}) = 0$. Thus, $P^C(\alpha_{\mathrm{k}}^{Pred_{\mathrm{k}}}|\tau') = 0$, which in turn implies that $P^{\mathrm{C}}(\tau'\alpha_{\mathrm{k}}) = 0$, and therefore that $P^{\mathrm{Spec}}(\tau'\alpha_{\mathrm{k}}) = 0$. A contradiction. □

Theorem 6.10. *Two LMPs are not E-equivalent if and only if $V^{\pi}(\mathsf{i}) > 0$ for some test-policy π of the MDP $\mathcal{M}$.*

Proof. $(\Rightarrow)$: by Theorem 6.9, we have that $V^{\pi}(\mathsf{i}) \neq 0$ for some test-policy $\pi = \alpha_1^{Pred_1}\alpha_2^{Pred_2}\ldots\alpha_n^{Pred_n}$. Define

$$J := \{j \in \{1, \ldots n\} \mid P^{Spec}(\alpha_{\mathrm{j}}^{Pred_{\mathrm{j}}}|\alpha_1 \ldots \alpha_{\mathrm{j\text{-}1}})\, \Delta^{\alpha_{\mathrm{j}}^{Pred_{\mathrm{j}}}}_{\alpha_1\ldots\alpha_{\mathrm{j\text{-}1}}} < 0\},$$

and note that $P^{Spec}(\alpha_{\mathrm{j}}^{Pred_{\mathrm{j}}}|\alpha_1 \ldots \alpha_{\mathrm{j\text{-}1}}) > 0$ and $\Delta^{\alpha_{\mathrm{j}}^{Pred_{\mathrm{j}}}}_{\alpha_1\ldots\alpha_{\mathrm{j\text{-}1}}} < 0$ for any $j \in J$. By Lemma 6.8, we have, for any meta-action α and trace τ, $\sum_{\alpha^{Pred} \in \psi_\alpha} \Delta^{\alpha^{Pred}}_{\tau} = 0$, then

$$\exists \mathit{Pred}' \text{ such that } \Delta^{\alpha_{\mathrm{j}}^{Pred'}}_{\alpha_1\alpha_2\ldots\alpha_{\mathrm{j\text{-}1}}} > 0.$$

Let π_1 be the test-policy obtained from π by replacing each action $\alpha_{\mathrm{j}}^{Pred_{\mathrm{j}}}$ where $j \in J$ by $\alpha_{\mathrm{j}}^{Pred'}$. Then, by Equation (6.7), $V^{\pi_1}(\mathsf{i}) > 0$, as desired.
$(\Leftarrow)$: follows from Theorem 6.9. □

Lemma 6.11. *For every test-policy π, for every meta-action $\alpha \in tuple(\mathcal{A}) \times \mathcal{A}$,*

$$\exists \alpha^{\mathtt{Pred}} \in \psi_\alpha \text{ such that } V^{\pi}(\mathsf{i}) \leq V^{\pi\alpha^{\mathtt{Pred}}}(\mathsf{i}).$$

Proof. As for Theorem 6.10, the result follows from Equation (6.7) and Lemma 6.8. □

Theorem 6.12. *If $\gamma < 1$ or $|\mathcal{M}| < \infty$ then $V^{\star}(\mathsf{i}) \geq V^{\pi}(\mathsf{i})$ for any test-policy π.*

Proof. If $|\mathcal{M}| < \infty$, since $\mathcal{M}$ has a tree structure, the result is a direct consequence of Lemma 6.11. Otherwise, it is sufficient to show that

$$\forall \epsilon > 0 \ \forall \text{ test-policy } \pi \ \exists \text{policy } \pi' \text{ such that } V^{\pi'}(\mathsf{i}) < V^{\pi}(\mathsf{i}) - \epsilon.$$

Let $\epsilon > 0$ and $\pi = \alpha_1^{Pred_1} \dots \alpha_n^{Pred_n}$ be a test-policy. Because of Lemma 6.11, 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 the states $\epsilon, \alpha_1, \alpha_1\alpha_2, \dots, \alpha_1\alpha_2 \dots \alpha_n$ will have the desired property. □

If "Impl" and "Spec" are equivalent, Theorem 6.9 implies that $V^{\star}(\mathsf{i}) = 0$. If, however, they are not equivalent, Theorem 6.10 implies that there exists at least a test-policy π such that $V^{\pi}(\mathsf{i}) > 0$. By Theorem 6.12, we can conclude that $V^{\star}(\mathsf{i}) > 0$.

Theorem 6.7. *Let $\mathcal{M}$ be the MDP induced by "Impl" and "Spec". If the discount factor $\gamma < 1$ or the size of MDP $|\mathcal{M}| < \infty$ then the optimal value $V^{\star}(\mathsf{i}) \geq 0$, and $V^{\star}(\mathsf{i}) = 0$ if, and only if, "Impl" and "Spec" are equivalent according to E.*

6.6 Hierarchy of TOEs

In the two previous sections, we presented the TOE framework and we proved that any equivalence defined as a TOE fits in the RL framework to estimate the divergences. In this section, we revisit all equivalence notions discussed so far in this thesis, namely, trace, ready, Barb acceptance, *K-moment*, and bisimulation equivalences and we try to view them as TOEs. Then we try to organize them in a hierarchy based on their distinguishing capabilities. Note that not all these equivalence notions can be defined as TOEs. Bisimulation is a clear example. For such equivalences, we simply consider their testing characterization.

Trace, *K-moment*, and testable Barb acceptance (Section 6.4) equivalences can be defined as TOEs. The following table gives a TOE definition for each one of them[1].

TOE	**Test grammar**	**Observation function**
Trace	$t ::= \omega \mid a.t$	$O_\omega = \{\omega^\checkmark\}$ $O_{a.t} = \{a^\times\} \cup \{a^\checkmark e \mid \ e \in O_t\}$
K-moment	$t ::= \omega \mid a^k.t$	$\psi_\omega = \{\omega^\checkmark\}$ $\psi_{a^k.t} = \{a^{k^\times}\} \cup \{a^{k^\times} e \mid e \in \psi_t\}$ $\cup \{a^{k^\checkmark} e \mid e \in \psi_t\}$
Barb acceptance	$t ::= \omega \mid \{a_1, \ldots, a_n\}a.t$	$\psi_\omega = \{\omega^\checkmark\}$ $\psi_{\{a_1,\ldots,a_n\}a.t} = \{\{a_1, \ldots, a_n\}a^\times\}$ $\cup \{\{a_1, \ldots, a_n\}a^\times \ e \mid e \in \psi_t\}$ $\cup \{\{a_1, \ldots, a_n\}a^\checkmark \ e \mid e \in \psi_t\}$

As mentioned in Section 2.3.3, ready equivalence is not testable. In this section, we define a testable variant of ready that we use in the comparative study. This testable variant, however, cannot be defined as a TOE. The following table gives the testing characterizations of testable ready as well as bisimulation.

Equivalence	**Test grammar**	**Observation function**
Bisimulation	$t ::= \omega \mid a.t \mid (t_1, \ldots, t_n)$	$O_\omega = \{\omega^\checkmark\}$ $O_{a.t} = \{a^\times\} \cup \{a^\checkmark e \mid \ e \in O_t\}$ $O_{(t_1,\ldots,t_n)} = O_{t_1} \times \ldots \times O_{t_n}$
Ready	$t ::= \omega \mid a.t \mid \{a_1, \ldots, a_n\}$	$O_\omega = \{\omega^\checkmark\}$ $O_{a.t} = \{a^\times\} \cup \{a^\checkmark e \mid \ e \in O_t\}$ $O_{\{a_1,\ldots,a_n\}} = O_{a_1} \times \ldots \times O_{a_n}$

By comparing the distinguishing capabilities of these equivalence notions, we obtain the hierarchy of Figure 6.1. The equivalence notions are organized from the weakest (bottom) to the strongest (top). Each arrow in the figure is explained using a simple example as follows.

- Trace equivalence is the weakest notion. For example, trace cannot distinguish between P_1 and P_2 processes of Figure 6.2 while any other notion can distinguish between them: *K-moment* with $c^1.a^1.b^2.\omega$, ready with $c.a.\{b, c\}$, Barb acceptance with $\{c\}c.\{a\}a.\{b, c\}b.\omega$, and bisimulation with $c.a.(b.\omega, c.\omega)$.

[1]Generally, there exists several TOE definitions for the same equivalence notion.

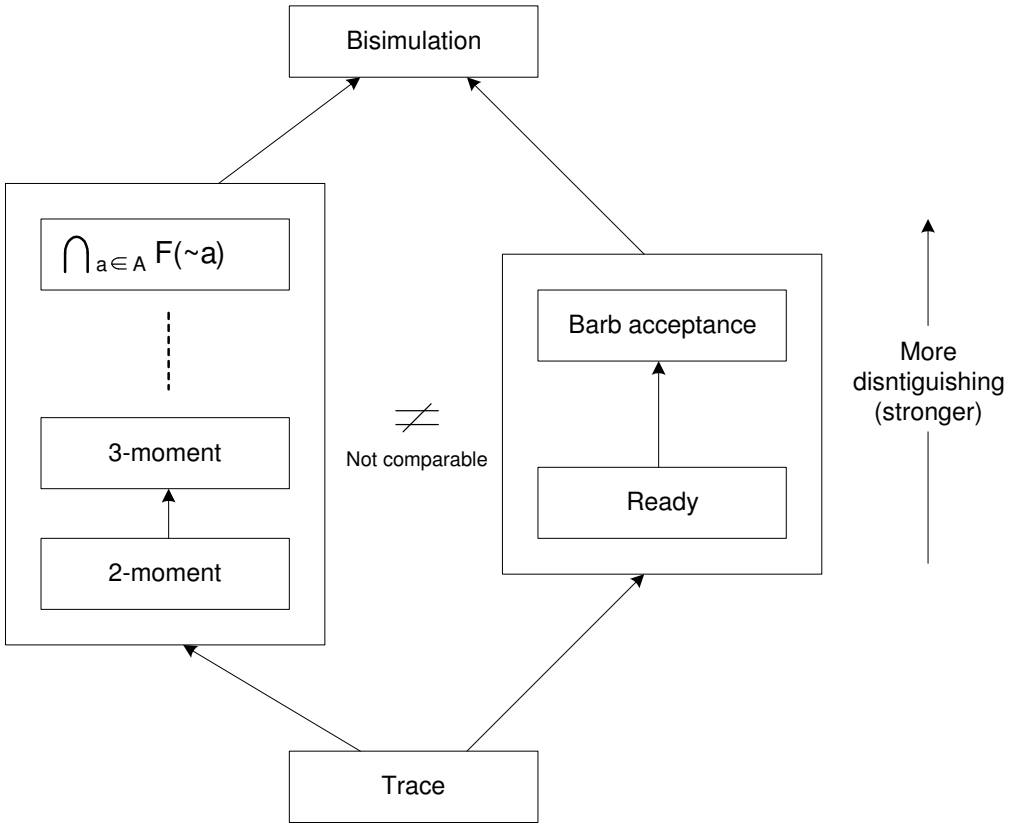


Figure 6.1: Hierarchy of TOEs and testable equivalence notions

- 3-*moment* TOE is more distinguishing than 2-*moment* TOE. P_1 and P_2 of Figure 6.3 are 1-*moment* (trace) equivalent, 2-*moment* equivalent but not 3-*moment* equivalent. Indeed, $Q^{P_1}_{a.a^2}(a^{\checkmark}a^{2^{\checkmark}}) = \frac{1}{2}(\frac{1}{10})^2 + \frac{1}{6}(\frac{4-\sqrt{5}}{10})^2 = \frac{3-\sqrt{5}}{75} = Q^{P_2}_{a.a^2}(a^{\checkmark}a^{2^{\checkmark}})$ while $Q^{P_1}_{a.a^3}(a^{\checkmark}a^{3^{\checkmark}}) \neq Q^{P_2}_{a.a^3}(a^{\checkmark}a^{3^{\checkmark}})$.

- Barb acceptance TOE is more distinguishing than testable ready. P_1 and P_2 of Figure 6.4 are trace equivalent, ready equivalent but not Barb acceptance equivalent. Indeed, test $t = \{a\}a.\{b,d\}b.\{c,d\}c$ yields different observation probabilities in P_1 and P_2: $Q^{P_1}_t(\{a\}a^{\checkmark}.\{b,d\}b^{\checkmark}.\{c,d\}c^{\checkmark}) = (1\,\frac{1}{2})(1\,\frac{1}{2}\,1)(\frac{1}{3}\,1\,\frac{1}{3}) = \frac{1}{36}$ whereas $Q^{P_1}_t(\{a\}a^{\checkmark}.\{b,d\}b^{\checkmark}.\{c,d\}c^{\checkmark}) = (1\,\frac{1}{2})(1\,\frac{1}{2}\,1)(1\,1\,1) = \frac{1}{4}$.

- K-*moment* TOEs are not comparable with testable ready equivalence and Barb acceptance TOE. T_1 and T_2 processes of Figure 6.5 are ready and Barb acceptance equivalent but not K-*moment*: the test $a.b^2$ distinguishes between them. On the other hand, Q_1 and Q_2 processes of the same figure are 2-*moment* equivalent but not ready nor Barb acceptance equivalent: the tests $a.\{b,c\}$ and $\{a\}a.\{b,c\}b$ can distinguish between them.

- Bisimulation is the strongest. Processes P_1 and P_2 of Figure 6.6 can be distin-

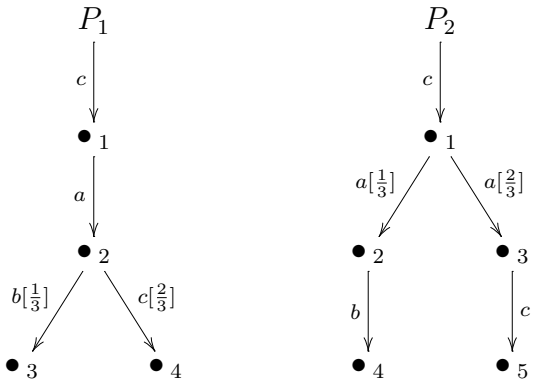


Figure 6.2: Only trace equivalence cannot distinguish between these processes.

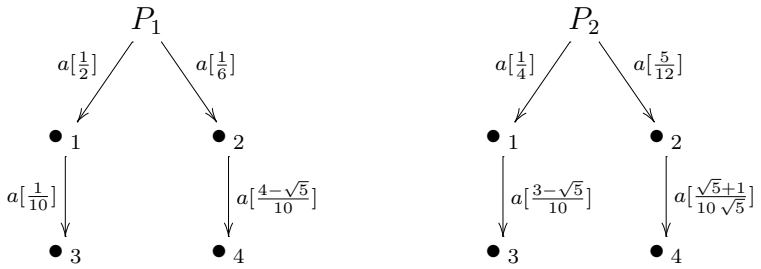


Figure 6.3: 2-moment equivalent but not 3-moment equivalent

guished only by bisimulation with test $a.(b.c.\omega, b.d.\omega)$.

6.7 Conclusion

TOE is a new framework to define equivalence notions based on testing. It refines simple testing characterization by adding more flexibility in the definition of observation functions. The idea is to group together "similar" observations which allow in one hand to focus on important aspects of observations and on the other hand to evaluate tests more efficiently since the number of observations will be considerably reduced. Since any equivalence notion defined as a TOE is recursive replication free (RRF), we showed that the RL framework of Chapter 4 to estimate divergences can be tailored to any such equivalences. Hence, this chapter is a generalization of Chapter 4 to different equivalence notions. The internal structure of the stochastic processes, however, is still assumed to be LMP. In the next chapter, we see how the approach can be generalized to other stochastic models that are mainly used in artificial intelligence.

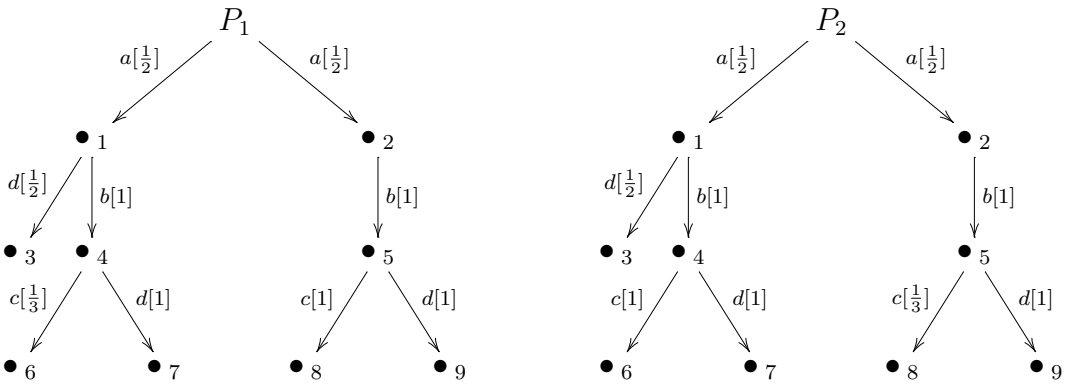


Figure 6.4: Ready equivalent but not Barb acceptance equivalent processes.

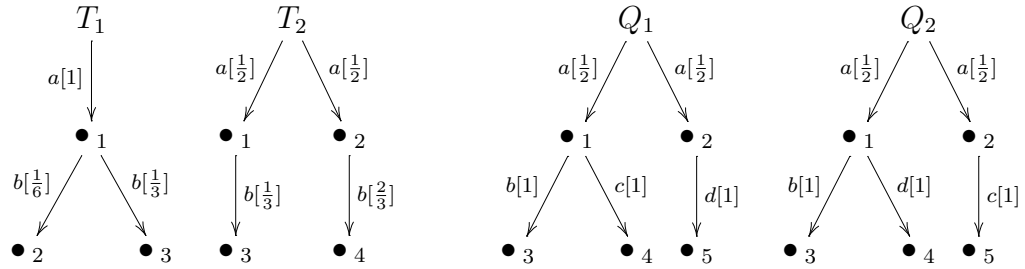


Figure 6.5: *K-moment* and Barb acceptance TOEs are not comparable

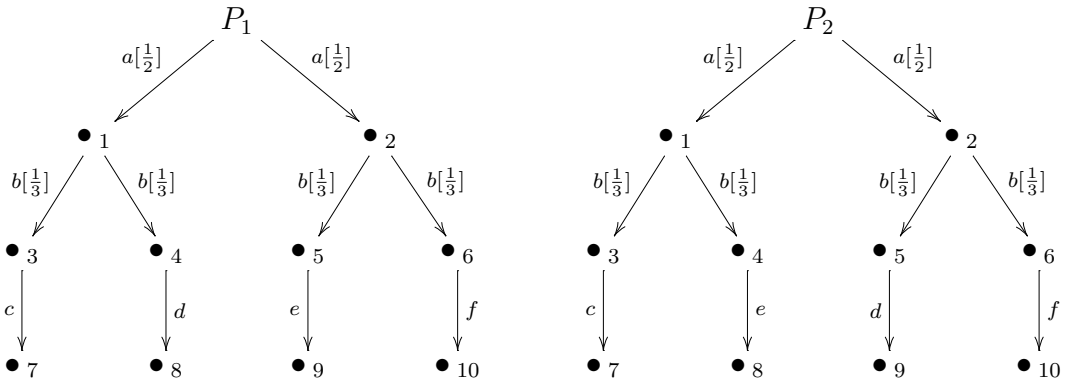


Figure 6.6: Only bisimulation can distinguish between P_1 and P_2.

Chapter 7

Divergence between Dynamical Systems

7.1 Introduction

All the results presented so far in this thesis hold for stochastic processes whose internal structure is an LMP. However, these results can be very useful in other applications and in different scientific fields. In particular, in artificial intelligence, there are several problems that define a stochastic behavior. These are called, in an artificial intelligence vocabulary, *dynamical systems*. An MDP is an example of a stochastic process that can be used to model a dynamical system. In this chapter, we discuss how the ideas presented so far in this thesis can be extended to such systems. We start by describing dynamical systems and models. Then we show how our proposed approach can be adjusted to a particular stochastic model, namely, partially observable Markov decision process (POMDP).

7.2 Dynamical Systems

Dynamical systems refer to the class of problems where the state of the environment changes over time[1]. If the system at time step t is in a particular state, at time step $t+1$, it can be in a new state. Often, the dynamics of these kinds of systems are subject to uncertainty. For example, it is not usually possible to predict deterministically the next

[1]By contrast to static systems where the state of the environment does not change.

state of the system. Modelling dynamical systems and then reasoning about them is the focus of several scientific communities, in particular, artificial intelligence, operations research, control theory, etc. Typically, in studying dynamical systems one is interested in one of two objectives: learning and controlling. Learning a dynamical system consists in estimating, more or less precisely, the parameters of the model. This will allow to understand the inner-working of the environment and unveil the patterns behind it. A good example is how human genes patterns (represented as Hidden Markov Models) can be learned from raw DNA sequences. Controlling dynamical systems, on the other hand, consists in finding an optimal strategy (or policy) of behaving given a goal task. This objective has been extensively discussed for the particular context of MDPs in Chapter 3.

In the last three decades, several models have been used to model dynamical systems. The problem to determine which model is more appropriate depends principally on the characteristics of the dynamical system. Of primary importance is the capability to identify the current state. That is, the *observability* of the system. For example, a player in a checkers game can see exactly his current state. However, a player in a Poker game does not since he has no way to know his opponent's cards. A second important element in the selection of the appropriate model is the prior knowledge of the states set, that is, the capability to enumerate the set of states of the system. For example, the set of states of a moving robot can be the set of possible locations in, let's say, a room. However, if one considers a medical diagnostic system, the set of states is not very clear since a state must capture complex information about the patient's disease and treatment history. The other aspect which may play an important role in the selection of the model is the *controllability* of the dynamical system. A controllable system assumes the existence, in some form or another, of an agent which controls the behavior of the system, typically by selecting the actions to take at each state. The moving robot is an example of a controllable dynamical system. On the other hand, an uncontrollable system evolves according to its internal dynamics without any external intervention. For example, a human gene can be viewed as DNA sequence generated from an uncontrollable system. Typically, with a controllable dynamical system one is interested in controlling whereas with uncontrollable systems one is interested in learning the model.

In the following, we classify the most used stochastic systems and models according to these criteria. We present models for observable systems, models for partially observable systems, and finally models that do not suppose the prior knowledge of the set of states, namely, history based and prediction based models.

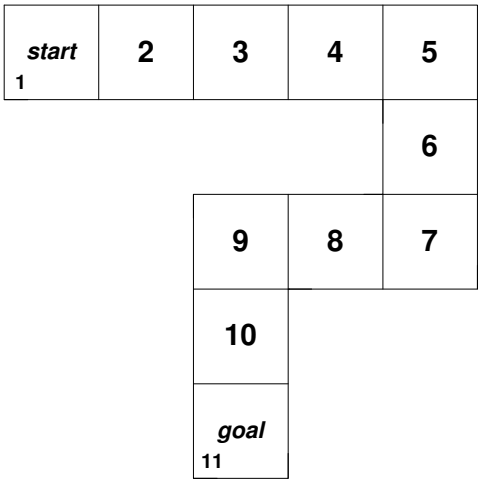


Figure 7.1: A simple Candy Land board game

7.3 Models for Observable Dynamical Systems

In an observable dynamical system, at each time step, there is a way to determine exactly the state in which the system is. In this section, we present two popular mathematical frameworks that can be used to model such systems, namely, *Markov chain* and *Markov decision process*.

7.3.1 Markov Chain

A Markov chain is a discrete-time stochastic process with the Markov property[2]. It can be used to model dynamical systems that are observable, uncontrollable, and whose set of states is known. The classical definition of Markov chain is a sequence of random variables $X_1, X_2, \ldots$ sharing the same alphabet S representing the set of states of the dynamical system. $Pr(X_3 = s_2 \,|\, X_2 = s_4)$ designates the probability to be in state s_2 at time step $t = 3$ such that the system was in state s_4 at time step $t = 2$. In this thesis, we prefer viewing the Markov chain as an automaton with a set of states and transitions between these states.

Definition 7.1. *A Markov chain is a tuple* (S, T) *where*

- S *is a set of states,*

[2]The Markov property is described in Section 3.2.2.

- $T : S \to \Pi(S)$ *is a transition function which maps each state* S *to a probability distribution over* S.

An example of a dynamical system that can be modelled by a Markov chain is a simple candy land board game with one player (Figure 7.1). The player starts from the *start* grid square and moves by rolling a single die. Hence, at each state, it has probability $\frac{1}{6}$ to move to one of the 6 next states. This is a very simple version of the game, but any version of Candy Land can be represented exactly as a Markov chain since from any square (state), the probability to move to any other square is fixed and is independent of any previous game history.

7.3.2 Markov Decision Process

An MDP can be viewed as a controllable Markov chain where each executed action yields a reward. Hence, any dynamical system which is completely observable, controllable, and whose states set is known can be modelled by an MDP. The automata definition of MDP as well as the main algorithms to solve MDPs have been detailed in Chapter 3.

7.4 Models for Partially Observable Dynamical Systems

Markov chains and MDPs can only be used to model dynamical systems where states are completely observable. However, several dynamical systems of interest do not satisfy this property. Indeed, the system can be in some state, but the agent has no way to figure out exactly that state. Generally, in such systems, there exist some observable events, called observations, which give hints about the actual state of the system at each step. Therefore they are called partially observable. By way of example, consider a robot which relies on its sensing hardware (camera, audio recorder, etc) to identify its current state. The sensing hardware, generally, are not completely reliable. Indeed, sometimes it cannot see a transparent glass door, sometimes a corridor looks like a corner, etc. Hence, the information acquired via the sensing hardware should be treated as observable events and not mapped deterministically to states because the same observation can be observed in different states. In this section, we describe two widely used models for such dynamical systems, namely, hidden Markov model (HMM) and partially observable Markov decision process (POMDP).

7.4.1 Hidden Markov Model

A HMM [41] is a Markov chain where states are not completely observable. That is, the underlying mechanics correspond to a Markov chain but the state of the system at a given moment is not exactly known. Instead, at each time step, a stochastic observation is generated based on the current state.

Definition 7.2. *A hidden Markov model is a tuple* (S, T, O, φ) *where*

- S *is a set of states,*
- $T : S \to \Pi(S)$ *is a state transition function which maps each state* S *to a probability distribution over* S*,*
- O *is a set of observations,*
- $\varphi : S \to \Pi(O)$ *is an observation function.* $\varphi(s, o)$ *represents the probability to observe* o *from state* s*.*

The HMM can be viewed as a doubly embedded stochastic process with an underlying stochastic process (the Markov chain internal mechanics) which is hidden and can only be observed through another set of stochastic processes that generate the sequence of observations. In real applications, the parameters of a HMM, namely, T and φ functions, are generally not known. Typically, the first step is to estimate the parameters of the HMM given a representative set of observation sequences, called the training set. Then, the estimated model can be used to analyze other sequences, in particular for pattern recognition. This suggests three problems that are associated with HMMs:

1. Given a set of observations sequences, figure out the HMM parameters that match the most these sequences.
2. Given a HMM, what is the probability to accept a particular observation sequence?
3. Given a HMM and an observation sequence, what is the most likely state sequence that could have generated the observation sequence?

HMMs have been particularly successful in speech recognition [41] and in DNA sequence analysis [14]. The two use cases are detailed in Appendix B.

7.4.2 Partially Observable Markov Decision Process

HMMs are not controllable and hence observations are generated at time steps without explicit interaction with the environment. From this point of view, partially observable Markov decision processes (POMDPs) [25] are more general since they are controllable[3]. POMDPs are mainly used for planning and hence come with a reward function.

Definition 7.3. *A POMDP is a tuple* (S, A, T, O, φ, R) *where*

- S *is a set of states,*
- A *is a set of actions,*
- $T : S \to \Pi(S)$ *is a state transition function which maps each state* S *to a probability distribution over* S*,*
- O *is a set of observations,*
- $\varphi : A \times S \to \Pi(O)$ *is an observation function.* $\varphi(a, s, o)$ *represents the probability to observe* o *when action* a *is executed and state* s *is reached. We assume that the observation depends on the action and the reached state. However, it is easy to formulate an equivalent model in which the observation depends on the action and the previous state.*
- $R : S \times A \to \mathbb{R}$ *is the reward function which maps each state-action pair to a real number representing the immediate reward.*

Solving a POMDP, that is, finding the optimal policy, is generally a very complex task. One interesting and very popular approach to deal with this complexity is to maintain a vector of state occupation probabilities [50]. This vector, known as the belief state, indicates at each time step the probability of being at any state. Then, the POMDP planning algorithm can be reformulated into an MDP having the belief states as state representation. Since a belief state is a vector of probabilities, it is continuous and has as many dimensions as POMDP states and hence the problem is apparently still complex (if not more complex). However, with belief states, Smallwood and Sondik [50] have shown that the optimal value function can be approximated arbitrarily well by a piecewise-linear and convex function. The piecewise linear convexity is illustrated by the following simple example. Consider a POMDP with two states s_1 and s_2. Hence, the belief state is a vector of two occupation probabilities: the probability of being in state s_1 and the probability of being in state s_2. From a geometrical point of view, any

[3] A HMM can be seen as a POMDP with only one action.

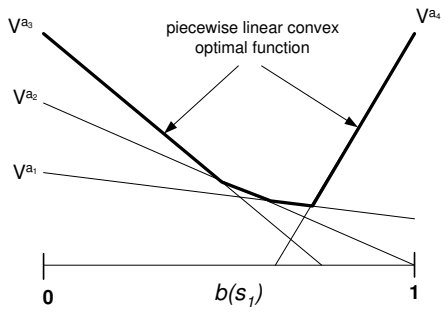


Figure 7.2: The optimal function of a POMDP with two states

belief state in this POMDP corresponds to a point in the segment $[0, 1]$ representing the probability of being at state s_1, noted $b(s_1)$. The probability of being at state s_2 is simply $1 - b(s_1)$ (Figure 7.2). In this setting, any action of the POMDP induces a value function that is linear in b. The optimal value function will intuitively correspond to the upper surface of those functions. An interesting aspect of this function is that it is piecewise linear and convex.

The convexity of the optimal value function can be explained intuitively as follows. A belief state in the middle of the segment $[0, 1]$ indicates a high level of uncertainty about the real state in the POMDP. In such a belief state, the agent cannot make a good choice of action and consequently it tends to get a limited long term reward. However, in the borders of the segment, there is less uncertainty about the current real state and the agent will select most of the time an appropriate action which guarantees more reward on the long run. For POMDPs with 3 states, the optimal function can be represented geometrically as a bowl shape that is composed of linear facets. For POMDPs with more states, the pattern is the same but it is difficult to draw geometrically. Most of the techniques to solve POMDPs are based on these ideas.

7.5 History-based Models

All models described so far assume the prior knowledge of the set of states. In several situations, this is possible because there is a natural state representation. For example, in physical systems, the set of states can be deduced directly from the attributes of the physical systems. However, in some problems, choosing the set of states is not so obvious. In such systems, the states are generally represented in terms of statistics over observations. In the following, we describe two stochastic models which can be used for such dynamical systems.

History-based models represent the states in terms of past observations or histories. They can be used to model dynamical systems which are partially observable, controllable, and without a prior knowledge of the set of states. The state at a given time step t is represented by the history h_t of past observations:

$$h_t = a_t, o_t, a_{t-1}, o_{t-1}, \ldots, a_{t-k}, o_{t-k}$$

where k can be fixed or variable from state to state. If k is fixed, the model is called n^{th}-order Markov model [4]. One of the drawbacks of this model is the choice of k. If k is greater than needed, the number of states will exponentially increase. If k is smaller than needed, several different world states will be confounded that yields to a bad representation of the dynamical system. Utile Suffix Memory (USM) [37] is a history-based model that represents each state by a different memory length (k). The idea is to vary the length of memory according to the relevance of the state with respect to the task goal. Hence, a state with a high expected value will be represented by a longer history. History-based models are easy to use in practice but they lack generality. The main reason to this is that several world states may have the same history and the same world state may have several histories.

7.6 Prediction-based Models (PSR)

Predictive State Representation (PSR) [34] is a new approach for representing dynamical systems which hold the promise of a more compact representation. Unlike POMDPs and MDPs, PSRs do not require a prior knowledge and enumeration of the set of states. Instead, it uses observations it experiences during interaction to figure out the states of the system. Hence, this representation is more "grounded" in data. On the other hand, unlike history-based models, PSR does not represent states as histories. Instead, it represents states as predictions of the possible outcome of tests. Hence, it uses future observations rather than past history to distinguish states.

The future in a controllable dynamical system is represented as a test. Let A be a set of actions and O be a set of observations. A test is an alternating sequence of actions and observations ($\{AO\}^*$) having the form:

$$t = a_1 o_1 a_2 o_2 \ldots a_n o_n.$$

Since it designates a future the prediction of test t, noted $p(t)$ is the probability to observe the sequence of observations $o_1 o_2 \ldots o_n$ given that the actions $a_1 a_2 \ldots a_n$ are to be selected. That is,

$$p(t) = P(o_1 o_2 \ldots o_n \,|\, a_1 a_2 \ldots a_n).$$

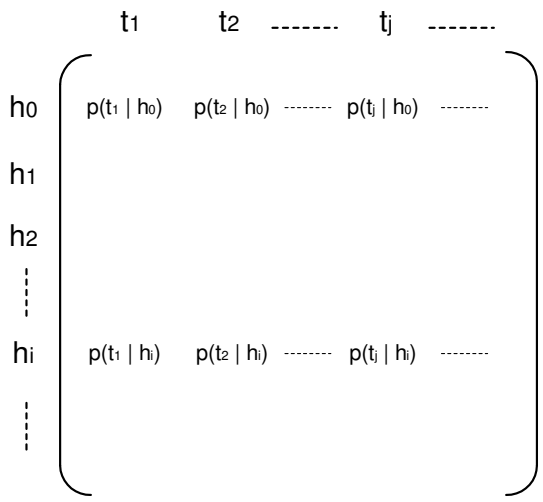


Figure 7.3: System-dynamics matrix of a PSR

If we have the prediction of any test in $\{AO\}^*$, then we know everything we need to know about the system.

One way to view the system is to consider a matrix whose columns are tests and whose rows are histories (Figure 7.3). A history has the same structure as a test, that is, an alternating sequence of actions and observations. The difference lies in the interpretation since a history is a test that has already happened. The entry of the matrix corresponding to history h and to test t is the prediction of test t given history h, noted $p(t \,|\, h)$. The columns and rows contain respectively all tests and histories organized in a lexicographic order. The resulting matrix with its infinitely many rows and columns is not a model of the system but the system itself since it represents everything we need to know about the system.

Having defined the system-dynamics matrix, the PSR approach is based on identifying a subset of tests that are sufficient to determine the prediction of any other test. This subset of tests is called *core tests*. Let $Q = q_1 q_2 \dots q_k$ be the set of core tests of a given system. Given a history h, the prediction of the core tests is a $1 \times k$ vector:

$$p(h) = (p(q_1|h), p(q_2|h), \dots, p(q_k|h)).$$

Each prediction vector for the core test corresponds to a state in the PSR model. Hence the state representation is the set of predictions for the core tests. The compactness of the PSR representation is illustrated by the fact that with a minimal subset of tests (core tests) it is possible to compute the prediction of any other test t. Indeed, for any

history h and test t, the prediction of test t given history h is

$$p(t|h) = f_t(p(h))$$

where $f_t : [0,1]^q \rightarrow [0,1]$ is a projection function. The size of the PSR representation depends directly on the number of the core tests. More tests are in the core tests, more accurate the PSR representation is to the real dynamical system.

PSR is a very recently introduced model and there are several open problems related to it, in particular, how to efficiently identify the set of core tests? how to use PSR for control, etc.

7.7 Quantifying the Difference between POMDPs

All stochastic models described so far can be used to model dynamical systems. A model is, by nature, an abstraction of the real system. Very often, it is more interesting to work with the model than the system itself mainly for two reasons. First, a model is a mathematical representation of the system or the phenomenon that makes further analysis more straightforward. Second, when the size of the real system is very large, one is better considering a smaller and tractable approximating model that discards "unimportant" details of the real system. Hence, the model may more or less deviate from the original behavior. To which level the model deviates from the original behavior is an important issue to assess the accuracy and fidelity of the model representation. In particular, one may wonder to which level an optimal policy of a model remains optimal in the real system. It is clear that a notion of distance or divergence between the dynamical system and its model may play an important role in such situations.

In real scenarios, it is very common that one needs to confront two dynamical systems to see whether they define the same behavior. For example, in presence of two robots, each one trying to find the exit door in a different room, it is interesting to figure out which robot has the simplest task. Depending on the situation, the comparison may need to be performed between two models of the same type (e.g. two POMDPs, two HMMs, two PSRs, etc), two models of different types (e.g. a POMDP and a PSR, a MC and HMM, etc), or even directly between the dynamical systems themselves.

The divergence notion between stochastic systems presented so far in this thesis can be applied in all these situations. Indeed, the approach we use is completely independent of the internal structure of the stochastic systems to be compared and it does not pose any restriction on them. It needs only the possibility to interact with them. This

reduces to a prediction capability. That is, given a selected action, it returns the resulting observation which can be an observation in a POMDP sense, a reward in an MDP sense, success or failure in an LMP sense, etc. In the following we show how the approach can be used to quantify the trace divergence between two POMDPs. However, the ideas can be easily extended to the other types of models and to the other TOE equivalences.

In Definition 7.3, a POMDP is defined as a tuple (S, A, T, O, φ, R). Note that the reward obtained on a state can be seen as an observation. Hence, the POMDP definition can be easily reformulated by considering rewards as part of observations without loosing any expressive power. This results in a simpler structure of the POMDP where the reward function is discarded, that is, a tuple (S, A, T, O, φ). This simpler structure is used in the rest of this chapter.

7.7.1 Testing Trace Equivalence in POMDPs

Let $P = (S, A, T, O, \varphi)$ be a POMDP and $t = a_1 a_2 \ldots a_n$ be a test generated from $\mathcal{T}_{trace}$ of Definition 2.7. Running a test t on P produces a sequence of observations of the form $e = o_1 o_2 \ldots o_{\text{n}}$ where $o_1, o_2, \ldots, o_{\text{n}}$ are elements of O. The set of all such sequences will be denoted O^*. For simplicity, we will make an abuse of notation and call them *observations*.

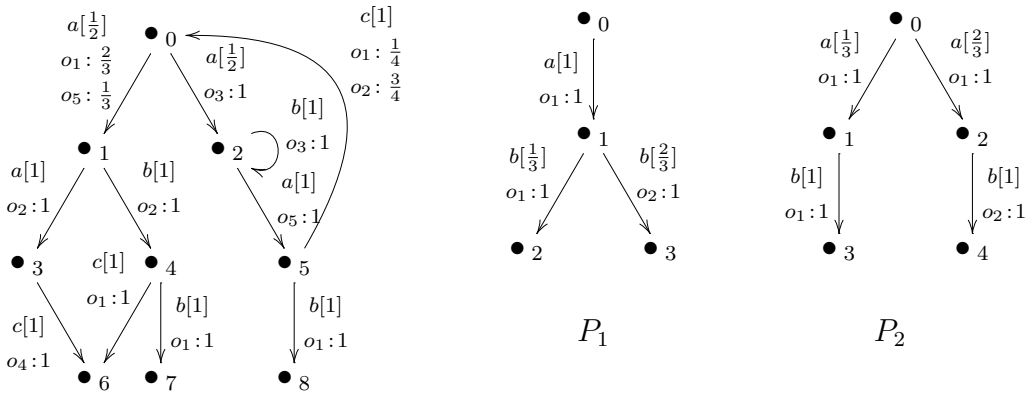
The probability of observing e after running test t from state s, $Q_t^s(e)$, is equal to:

$$\begin{aligned} Q_t^s(e) &= Pr(o_1, o_2, \ldots, o_{\text{n}} | a_1, a_2, \ldots, a_n) \\ &= Pr(o_{\text{n}} | a_1 o_1 \ldots o_{\text{n-1}} a_n) \times Pr(o_{\text{n-1}} | a_1 o_1 \ldots o_{\text{n-2}} a_{n-1}) \times \ldots \times Pr(o_2 | a_1 o_1 a_2) \times Pr(o_1 | a_1) \end{aligned}$$

where $Pr(o_{\text{n}} | a_1 o_1 \ldots o_{\text{n-1}} a_n)$ is the probability of observing o_{n} after running action a_n given that the first $n-1$ actions were $a_1, \ldots, a_{n-1}$ and the first $n-1$ observations were $o_1, \ldots, o_{\text{n-1}}$. Note that the second equality is a consequence of the Markov property. For example, in Figure 7.4 (a), $Pr(o_4 | a o_1 a o_2 c) = 1$.

Given a state s, each test t defines a probability distribution Q_t^s on observations of O^*; in particular, $\sum_{e \in O^*} Q_t^s(e) = 1$. In Figure 7.4(a), $Q_a^0(o_1) = \frac{1}{2}\frac{2}{3} + \frac{1}{2}0 = \frac{1}{3}$, $Q_a^0(o_5) = \frac{1}{6}$, and $Q_a^0(o_3) = \frac{1}{2}$. Also, for test $t = aac$, $Q_t^0(o_1 o_2 o_4) = \frac{1}{3}$, $Q_t^0(o_5 o_2 o_4) = \frac{1}{6}$, $Q_t^0(o_3 o_5 o_1) = \frac{1}{8}$, and $Q_t^0(o_3 o_5 o_2) = \frac{3}{8}$.

According to Theorem 2.8, two POMDPs are trace equivalent if for any test generated from $\mathcal{T}_{trace}$ they yield the same probability distribution on observations.



(a) A POMDP example **(b)** Non *K-moment* equivalent POMDPs

Figure 7.4: Partially Observable Markov Decision Processes

7.7.2 Testing *K-moment* Equivalence in POMDPs

As discussed in Chapter 5, for many applications, trace equivalence does not discriminate enough. This is also true for POMDPs. For instance, it cannot discriminate between the POMDPs P_1 and P_2 of Figure 7.4(b). Indeed, in both POMDPs, test a produces the observation o_1 with probability 1, test ab produces observation o_1o_1 with probability $\frac{1}{3}$ and observation o_1o_2 with probability $\frac{2}{3}$. Hence, trace equivalence cannot catch the fact that P_1, after executing action a, can produce o_1 and o_2 from the same state. However, *K-moment* does discriminate between them. Recall that a test t generated from the test grammar $\mathcal{T}_{Kmoment}$ of Definition 5.10 has the form $t = a_1^{k_1}a_2^{k_2}\dots a_n^{k_n}$. Running the test t on a POMDP may produce an observation e which is longer than the test t, that is, $e = o_{11}o_{12}\dots o_{1k_1}o_{21}\dots o_{2k_2}\dots\dots o_{nk_n}$. *K-moment* can distinguish between P_1 and P_2 of Figure 7.4 (b) with test $t = ab^2$: in P_1, $Q_t^0(o_1o_1o_2) = \frac{2}{9}$ whereas in P_2, $Q_t^0(o_1o_1o_2) = 0$.

For simplicity, we show how the problem of quantifying the trace-equivalence divergence between POMDPs is formulated. Based on the results of the previous chapter, the ideas can be generalized to *K-moment* equivalence as well as any TOE.

7.7.3 The Stochastic Game

Let P_1 and P_2 be two POMDPs with the same set of actions A and the same set of observations O. As for LMPs "Impl" and "Spec" of Chapter 4, the idea is to define an

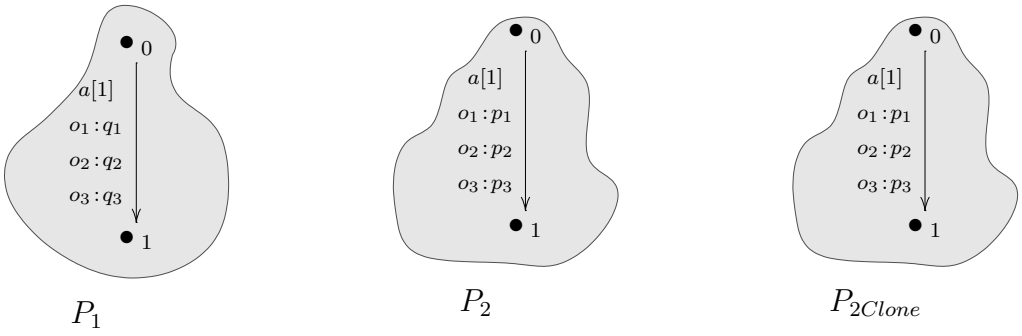


Figure 7.5: A generic example.

MDP out of P_1 and P_2 and to interpret the optimal value of this MDP as a divergence between them. As in Chapter 4, we prefer introducing the MDP formulation by a stochastic game which illustrates how the interaction with both POMDPs occur.

Recall that with LMPs, when an action is executed in a state, two outcomes are possible: success or failure. However, with POMDPs, an action may yield several observations. Hence, the symmetric and negative cases discussed in Section 4.2 can occur in more complex forms. Consider the generic example of Figure 7.5. Note that the clone of P_2 is called P_{2Clone}, or shortly P_{2c}. In this example, when the POMDP P_1 runs action a, the observations o_1, o_2, and o_3 can be generated with probabilities q_1, q_2, and q_3. For P_2 and P_{2c}, these probabilities ar p_1, p_2, and p_3. Let us compute the expected reward of policy $\pi = a$ in the same spirit as $\mathbf{Game}_0$ in Section 4.2, that is,

$$R = (Obs.P_1 \neq Obs.P_2) - (Obs.P_2 \neq Obs.P_{2c}).$$

For any observation $o \in O$, let $\overline{o}$ designates any observation except o. Then,

$$\begin{aligned}
R_{\pi=a}(Reward) &= Pr(\langle o_1\overline{o}_1\overline{o}_1\rangle)R(\langle o_1\overline{o}_1\overline{o}_1\rangle) + Pr(\langle \overline{o}_1 o_1 o_1\rangle)R(\langle \overline{o}_1 o_1 o_1\rangle) \\
&\quad + Pr(\langle \overline{o}_1\overline{o}_1 o_1\rangle)R(\langle \overline{o}_1\overline{o}_1 o_1\rangle) + Pr(\langle o_1 o_1\overline{o}_1\rangle)R(\langle o_1 o_1\overline{o}_1\rangle) \\
&\quad + Pr(\langle o_2\overline{o}_2\overline{o}_2\rangle)R(\langle o_2\overline{o}_2\overline{o}_2\rangle) + Pr(\langle \overline{o}_2 o_2 o_2\rangle)R(\langle \overline{o}_2 o_2 o_2\rangle) \\
&\quad + Pr(\langle \overline{o}_2\overline{o}_2 o_2\rangle)R(\langle \overline{o}_2\overline{o}_2 o_2\rangle) + Pr(\langle o_2 o_2\overline{o}_2\rangle)R(\langle o_2 o_2\overline{o}_2\rangle) \\
&\quad + Pr(\langle o_3\overline{o}_3\overline{o}_3\rangle)R(\langle o_3\overline{o}_3\overline{o}_3\rangle) + Pr(\langle \overline{o}_3 o_3 o_3\rangle)R(\langle \overline{o}_3 o_3 o_3\rangle) \\
&\quad + Pr(\langle \overline{o}_3\overline{o}_3 o_3\rangle)R(\langle \overline{o}_3\overline{o}_3 o_3\rangle) + Pr(\langle o_3 o_3\overline{o}_3\rangle)R(\langle o_3 o_3\overline{o}_3\rangle) \\
&= q_1(1-p_1)(1-p_1)(+1) + (1-q_1)p_1p_1(+1) \\
&\quad + (1-q_1)(1-q_1)p_1(-1) + q_1p_1(1-p_1)(-1) \\
&\quad + q_2(1-p_2)(1-p_2)(+1) + (1-q_2)p_2p_2(+1) \\
&\quad + (1-q_2)(1-q_2)p_2(-1) + q_2p_2(1-p_2)(-1) \\
&\quad + q_3(1-p_3)(1-p_3)(+1) + (1-q_3)p_3p_3(+1) \\
&\quad + (1-q_3)(1-q_3)p_3(-1) + q_3p_3(1-p_3)(-1) \\
&= (2p_1-1)(p_1-q_1) + (2p_2-1)(p_2-q_2) + (2p_3-1)(p_3-q_3). \quad (7.1)
\end{aligned}$$

Equation (7.1) can be equal to zero for different combinations of values. To mention only one: $q_1 = \frac{1}{2}$, $q_2 = q_3 = \frac{1}{4}$, and $p_1 = p_2 = p_3 = \frac{1}{3}$. Once again the solution for such undesirable cases is to use prediction. The stochastic game with prediction proceeds as follows:

Game$_{\text{tracePOMDP}}$: Put the three POMDPs (P_1, P_2, and P_{2c}) on their initial states; then

Step 1 : The player chooses an action and makes a prediction o of the obtained observation on P_2.

Step 2 : Action a is run on P_1, P_2. and P_{2c}.

Step 3 : A reward is computed as follows. Writing 1 for P_1, 2 for P_2, $2c$ for P_{2c}, and *Obs* for observation,

$$R \;:=\; \big(Obs.2 = o\big)\,\big((Obs.1 \neq Obs.2) - (Obs.2 \neq Obs.2c)\big) \tag{7.2}$$

where 0 and 1 are used as both truth values and numbers. This reward will be revealed to the player at the end of the game.

Step 4 : Repeat until the episode ends on one of the three processes.

7.7.4 MDP Formulation

The MDP corresponding to **Game**$_{\text{tracePOMDP}}$ is defined as follows.

Definition 7.4. Given two POMDPs P_1 and P_2 with the same sets of actions and observations, A and O, the induced MDP $\mathcal{M}$ is a tuple $(S, \mathsf{i}, Act, Pr^{\mathcal{M}}, R)$. The set of states S is the set of possible traces of the processes:

$$S := \{\tau : A^*\}.$$

The initial state is $\mathsf{i} = \epsilon$ (the empty sequence). The set of actions of $\mathcal{M}$ is:

$$Act := \{a^o \mid a \in A \text{ and } o \in O\}.$$

A state $s = \tau$ can make a transition to state $s' = \tau\, a$ through actions of the form a^o with probability 1 and then obtain the reward of Equation (7.2). That is,

$$Pr^{\mathcal{M}}_{s \to s'}(a^o) := 1 \quad \text{and} \quad R^{a^o}_s := Pr^2(a^o|s)\,(Pr^{2c}(a^o|s) - Pr^1(a^o|s)) \tag{7.3}$$

where $Pr^i(a^o|s)$ is the conditional probability of observing o after action a given a successfully executed trace $s = \tau$ in the POMDP P_i.

The trace equivalence divergence between POMDPs can be defined as follows:

Definition 7.5. Let P_1 and P_2 be two POMDPs and i the initial state of their induced MDP. We define their *trace equivalence divergence* as

$$\mathrm{div}_{\mathrm{trace}}(P_1 \| P_2) := V^\star(\mathsf{i}).$$

This definition takes its origin in the following theorem.

Theorem 7.6. *Let $\mathcal{M}$ be the MDP induced by POMDPs P_1 and P_2. If $\gamma < 1$ or $|\mathcal{M}| < \infty$ then the optimal value $V^\star(\mathsf{i})$ is greater than or equal to 0, and $V^\star(\mathsf{i}) = 0$ if, and only if, P_1 and P_2 are trace equivalent.*

The proof is very similar to the proof of Theorem 6.7. Indeed, as a meta-action α yields different possible outcomes in ψ_α, a simple action a in the case of POMDPs yields several possible observations $o_1, o_2, \ldots, o_n$.

7.7.5 Experimental Results

We made the following experiment on various benchmark problems from Tony Cassandra's POMDP web page [66] to check if, in practice, our algorithm can rapidly find differences between slightly different POMDPs. The experimentation consists in comparing each POMDP benchmark with a slightly modified version of the same POMDP. We chose the following modifications. We choose at random an action a, a state s of the model and two other states s' and s'' reachable from s through a. Then we swap the probabilities of the two next-state transitions from s to s' and s''. Table 7.1 shows the result of this experimentation. For each benchmark (Column 1), Column 2 indicates the number of states of the POMDP, Column 3 its number of actions, Column 4 the number of episodes used by the algorithm, and Column 5 the selected precision (ϵ) of the PAC lower bound of Equation (4.12) (Chapter 4). The confidence value δ of that bound has been fixed to 0.05 for all benchmarks. The lower bound of $\mathrm{div}_{\mathrm{trace}}$ is given in Column 6. Note that this lower bound is positive for all benchmarks except mini-hall2. This tends to confirm that, in practice, the algorithm is able to detect slight differences. The negative lower bound in the case of mini-hall2 (-0.0033) can be explained by the fact that the divergence was too small to be detected after 10^6 episodes. The large divergence value returned in the case of Shuttle Docking is due to the fact that the benchmark turns out to be an MDP and hence, each state generates only one observation with probability 1.

	♯ of states	♯ of actions	♯ of observations	♯ of episodes	Precision (ϵ)	Bottom bound for divergence
Random POMDP	4	4	20	$5 * 10^5$	0.005	0.0115
4x3 maze	11	4	18	10^6	0.005	0.0809
Cheese maze	11	4	7	$2 * 10^5$	0.005	0.0335
Tiger	2	3	6	$5 * 10^5$	0.005	0.0015
Shuttle docking	8	3	10	10^6	0.005	0.6949
Part painting	4	4	6	10^6	0.005	0.2289
mini-hall2	13	3	9	10^6	0.005	-0.0033
hallway	60	5	21	10^7	0.0005	0,000539

Table 7.1: The results of experimenting our divergence algorithm on slightly different POMDPs.

7.8 Conclusion

In the last three decades, several stochastic formalisms have been proposed to model dynamical systems. Depending on the characteristics of these systems (observability, controllability, etc.) one might prefer to use a model rather than another. Partially observable Markov decision process (POMDP) is considered as the most general model since it can be used in partially observable and controllable environments. In this last chapter, we showed how the main contribution of this thesis can be extended to quantify the divergence between POMDPs. But more importantly, this chapter opens the way to apply the proposed approach on other formalisms such as PSRs, HMMs, MDPs, etc.

Chapter 8

Conclusion

8.1 Summary of Contributions

The three main contributions of this thesis are:

1. a new approach to quantify the divergence between pairs of stochastic systems based on reinforcement learning [10, 31]
2. a new family of equivalence notions, called K-*moment* which lies between trace equivalence and bisimulation [11]
3. a refined testing framework, called TOE, to define equivalence notions [12].

The first and most important point of this thesis is that RL can be used to quantify efficiently the divergence between stochastic systems. The idea is to define an MDP out of the systems to be tested and then to interpret the optimal value of the MDP as the divergence between the systems. From the optimal policy one can extract the test that exhibits the most this divergence. The most appealing feature of the proposed algorithm is that it does not rely on the knowledge of the internal structure of the systems. Only a possibility of interacting with them is required. Because of this, the approach can be applied to different types of stochastic systems. In this thesis, we detailed the case of LMP and POMDP, however, one can straightforwardly extend the ideas to MDPs, PSRs, etc.

The second contribution is a new family of equivalence notions, K-*moment*, that constitute a good compromise between trace equivalence (too weak) and bisimulation

(too strong). This family has natural definitions using, respectively, coincidence of moments of random variables and the function F (the one whose fixed point corresponds to bisimulation), but more importantly, it has a simple testing characterization. The testing of *K-moment* does use the replication of states but not recursively. Recursive replication is required to test bisimulation and for this reason the latter is considered too strong.

The test-observation-equivalence (TOE) framework is the third contribution of this thesis. It is a refined testing framework to define equivalence notions with more flexibility. The idea is to make it possible to group together "similar" observations which result in more appropriate observation functions. On one hand, this allows to focus only on the important aspects of observations, on the other hand, it makes the evaluation of tests more efficient since the number of observations can be considerably optimized. Any equivalence notion defined as a TOE is recursive replication free (RRF). Hence, it fits in the RL approach to quantify the divergence between stochastic processes.

8.2 Future Work

The first work we plan to investigate in the future is related to the main limitation of the proposed approach, namely, the large number of actions of the induced MDP. Indeed, since the immediate reward following an action depends on the prediction of its outcome, the number of actions is multiplied by the number of possible outcomes of each action. For example, in the particular case of POMDP, the number of actions of the induced MDP is $|A| \times |O|$ where A and O are respectively the sets of actions and observations of the two POMDPs to be compared. There are methods to reduce the number. For example, up to bisimulation, we found that a POMDP with a very large or continuous set of observations is equivalent to a POMDP whose observation set has exactly two elements. However, further reduction is needed if one wants this method to perform efficiently on huge systems.

The notion of divergence we propose in this thesis is not symmetric nor satisfying the triangle inequality. A symmetric version of the divergence can be naturally defined as:

$$\frac{\mathrm{div}(P_1 \,||\, P_2) + \mathrm{div}(P_2 \,||\, P_1)}{2}$$

A research direction we are considering is to explore more the properties of the divergence notion. For instance, given two equivalent processes P_1 and P_2 ($\mathrm{div}(P_1 \,||\, P_2) = 0$)

and a third one T, it is interesting to prove that:

$$\mathrm{div}(P_1 \,||\, T) = \mathrm{div}(P_2 \,||\, T).$$

Many practical applications can benefit from the proposed approach. For example, the two use cases of Appendix B, namely, automatic speech recognition and DNA sequence analysis. In the first, each word is represented by a HMM. The parameters of the HMM are obtained by iterating over the training set. The iterations continue until the parameters become relatively stable. Hence, each iteration involves the estimation of the divergence between the new model and the previous one. Our approach can be used at that level. In the second use case, the HMM is used to represent a gene or a family of genes. Given a particular DNA sequence, one of the most important tasks of DNA sequence analysis is to identify to which gene the sequence corresponds. Since our approach is independent of the internal structure of stochastic processes, one can use it to estimate the divergence between a DNA sequence and respectively, the HMMs of all genes. The gene to which it corresponds will more likely be the one whose HMM is closer to the sequence.

One of the important long term motivations of this thesis is to contribute to a theory of approximation for stochastic systems, in particular for POMDPs. Most of RL algorithms assume that the POMDP state space is finite, and hence, they store the value of each state in a separate memory location. However, most practical applications have very large state spaces, for which such representation is not feasible. Indeed, the problem of finding a good state representation in stochastic systems with observations is a topic of increasing interest. PSR [34] is a promising approach but there are many other possibilities. For example, a new theoretical framework has been recently proposed in [20]. The idea we want to explore is to reshape the divergence algorithm we presented in this thesis to propose an approximation theory for POMDPs. The main objective is to be able to construct small approximations of huge POMDPs. Traditional algorithms can then be applied to solve the approximation POMDPs. But more importantly, solving an approximation POMDP will give a sufficiently good estimation of how to solve the original huge one.

Bibliography

[1] J. Tamarkin A. Shohat. *The Problem of Moments*. American Mathematical society, 1943.

[2] A. Barto and M. Duff. Monte Carlo matrix inversion and reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, volume 6, pages 687–694, 1994.

[3] R. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[4] D. Bertsekas and S. Shreve. *Stochastic Optimal Control*. Academic Press, 1978.

[5] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[6] J. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimates of parameters. In *Advances in Neural Information Processing Systems (NIPS)*, pages 211–217, 1989.

[7] T. Cover and J. Thomas. *Elements of Information Theory*, chapter 12. Wiley, 1991.

[8] J. Desharnais, A. Edalat, and P. Panangaden. A logical characterization of bisimulation for labeled Markov processes. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS)*, page 478, Washington, DC, USA, 1998. IEEE Computer Society.

[9] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

[10] 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.

[11] J. Desharnais, F. Laviolette, and S. Zhioua. Testing probabilistic equivalence through reinforcement learning. In *Lecture Notes in Computer Science*, volume 4337, pages 236–247, 2006.

[12] J. Desharnais, F. Laviolette, and S. Zhioua. Testing probabilistic equivalence through reinforcement learning. *Submitted to Information and Computation*, 2008.

[13] J. Desharnais. *Labelled Markov processes*. PhD thesis, School of Computer Science, McGill University, Montreal, November 1999.

[14] R. Durbin and S. Eddy. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[15] G. Forney. The Viterbi algorithm. *IEEE*, 61:268–278, 1973.

[16] 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, pages 443–458, 1990.

[17] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[18] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[19] J. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[20] C. Hundt, P. Panangaden, J. Pineau, and D. Precup. Representing systems with hidden state. In *AAAI Press*, 2006. Canadian Conference on Artificial Intelligence.

[21] M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *Logic in Computer Science*, pages 111–122, 1997.

[22] T. Jaakkola, M. Jordan, and S. Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, volume 6, pages 703–710, 1994.

[23] C. Jou and S. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In *Proceedings of the First International Conference on Concurrency Theory (CONCUR)*, number 458 in Lecture Notes in Computer Science. Springer-Verlag, 1990.

[24] B. Juang and L. Rabiner. A probabilistic distance measure for hidden return models. *AT&T Technical Journal*, 62(2):391–408, 1985.

[25] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[26] A. Klopf. Brain function and adaptive systems - A heterostatic theory. Technical report, Air Force Cambridge Research Laboratories, 1972.

[27] M. Kwiatkowska and G. Norman. Metric denotational semantics for PEPA. In *Proceedings of the 4th Process Algebra and Performance Modelling Workshop*, pages 120–138, 1996.

[28] M. Kwiatkowska and G. Norman. Probabilistic metric semantics for a simple language with recursion. In *Proceedings of the Mathematical Foundations of Computer Science (MFCS)*, number 1113 in Lecture Notes in Computer Science, pages 419–430. Springer-Verlag, 1996.

[29] M. Kwiatkowska and G. Norman. A testing equivalence for reactive probabilistic processes. *Electronic Notes in Theoretical Computer Science*, pages 114–132, 1998.

[30] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[31] F. Laviolette and S. Zhioua. Testing stochastic processes through reinforcement learning. In *NIPS Workshop on Testing of Deployable Learning and Decision Systems*, page 8, 2006.

[32] N. Leveson and C. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 27(7):18–41, July 1993.

[33] S. Levinson and L. Rabiner. An introduction to the application of the theory of probabilistic functions of a return process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, 1983.

[34] M. Littman, R. Sutton, and S. Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS) 14*, pages 1555–1561, 2001.

[35] G. Lowe. Representing nondeterministic and probabilistic behaviour in reactive processes. Technical Report PRG-TR-11-93, Oxford University, Computing Laboratory, 1993.

[36] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1-3):159–195, 1996.

[37] A. McCallum. *Reinforcement learning with selective perception and hidden state.* PhD thesis, University of Rochester, 1995.

[38] D. Michie and R. Chambers. BOXES: An experiment in adaptive control. *Machine Intelligence*, 2:137–152, 1968.

[39] R. Milner. *A calculus of communicating systems*. Springer-Verlag, 1980.

[40] D. Parker. *Implementation of Symbolic Model Checking for Probabilistic Systems*. PhD thesis, University of Birmingham, 2002.

[41] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[42] L. Rabiner. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[43] S. Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, 1983.

[44] G. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical report, Engineering Department, Cambridge University, 1994.

[45] G. Rummery. *Problem Solving with Reinforcement Learning*. PhD thesis, Cambridge University, 1995.

[46] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:211–229, 1959.

[47] C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[48] S. Singh and R. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158, 1996.

[49] B. Skinner. *The Behavior of Organisms*. Appleton Century, 1938.

[50] R. Smallwood and E. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[51] R. Sutton and A. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998.

[52] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[53] R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning (ICML)*, pages 216–224, 1990.

[54] R. Sutton. Planning by incremental dynamic programming. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 353–357, 1991.

[55] R. Sutton. Generalization in reinforcement learning: successful examples using sparse coding. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1038–1044, 1995.

[56] G. Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38:58–68, 1995.

[57] F. van Breugel and J. Worrell. Towards quantitative verification of probabilistic transition systems. In *Lecture Notes in Computer Science*, volume 2076, pages 421–432, 2001.

[58] F. van Breugel and J. Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theoretical Computer Science*, 2006.

[59] R. van Glabbeek, S. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[60] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, 1967.

[61] C. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

[62] D. White. *Dynamic Programming*. Holden Day, 1969.

[63] P. Whittle. *Optimization over Time*. Wiley, 1982.

[64] R. Williams and L. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Northeastern University, College of Computer Science, Boston, MA, 1993.

[65] CISMO Tool. http://www2.ift.ulaval.ca/~jodesharnais/cismo/.

[66] Tony's POMDP page. http://www.cs.brown.edu/research/ai/pomdp/.

Appendix A

Tuning Q-learning Parameters

The RL algorithm we use in our implementation of the proposed approach is Q-learning. The quality and the accuracy of the result returned by this algorithm depends on the values of some parameters, in particular, the learning rate α, the exploration rate ϵ if ϵ-greedy is used for action selection, and the temperature τ if the Softmax is used for action selection. Tuning these parameters consists in figuring out the combination of values that produces the best results, which can differ from one task to another. Intuitively, a good combination of values will guarantee a good tradeoff between exploration and exploitation. The objective of this section is to illustrate the tuning procedure of the Q-learning parameters for the kind of tasks described in this thesis, that is, solving the MDP induced by "Impl" and "Spec". For each one of the mentioned parameters, we try several fixed values and also several functions to vary these values in the same experiment. In order to obtain a useful combinations of values, we carry out the tuning on an example that requires a good tradeoff between exploration and exploitation.

Consider the example of Figure A.1. The objective is to compute the trace equivalence divergence between "Spec"and "Impl". The optimal policy of the induced MDP is $\pi^* = a^{\checkmark}a^{\checkmark}c^{\checkmark}c^{\times}$ and $V^*(\mathsf{i}) = 0.262$ (i is the initial state of the MDP). This example is interesting because the biggest difference between "Spec"and "Impl"occurs relatively deep in the LMPs (states 6 in both processes.) However, there exists a sub-optimal policy which is easier to detect but with lower value. This sub-optimal policy is $\pi = b^{\checkmark}a^{\checkmark}b^{\checkmark}c^{\checkmark}$ and $V^{\pi}(\mathsf{i}) = 0.153$. This example requires a good strategy of exploration in order to figure out the optimal policy without prematurely converging to the sub-optimal policy.

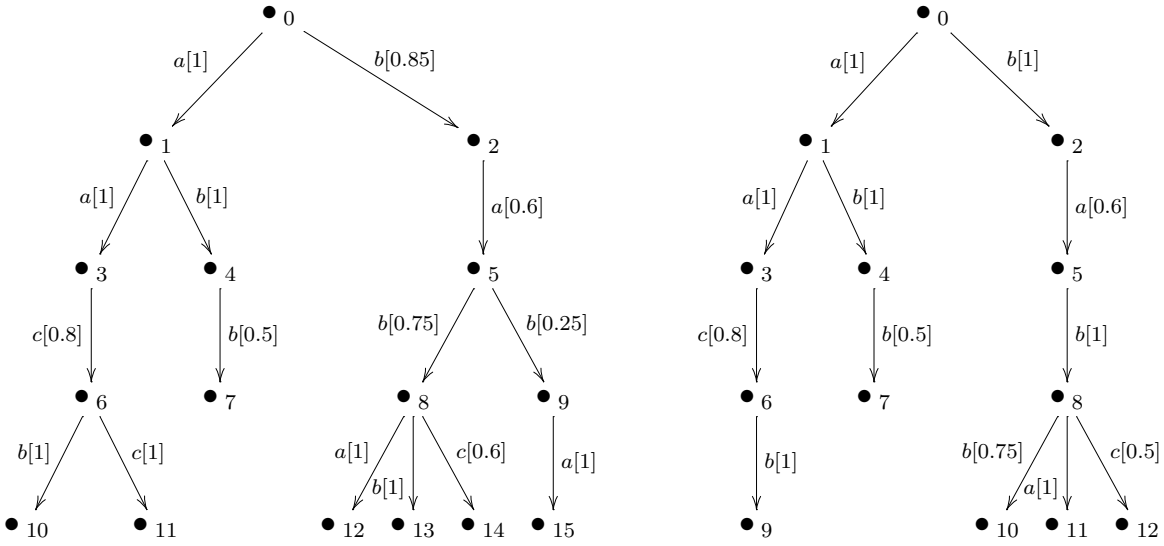


Figure A.1: Example for tuning

A.1 Learning Rate : α

Recall that the learning rate or the step-size parameter is used to update Q-values of state-action pairs. A well known result in stochastic approximation theory states that, in order to obtain convergence, two conditions are required for α:

$$\sum_{x=1}^{\infty} \alpha_x(s,a) = \infty \quad \text{and} \quad \sum_{x=1}^{\infty} \alpha_x^2(s,a) < \infty$$

where $\alpha_x(s,a)$ is the value of α when action a is selected from state s for the x^{th} time. Hence, the α parameter is typically computed in terms of x, the number of times the state-action pair has been visited. In order to figure out a good function to vary α, we try several functions, namely, a constant $\alpha = 0.1$, $\frac{1}{x}$, $\frac{3}{x}$, and $\frac{1}{1+\frac{x}{7}}$.

The experiment we perform is to run a task of Figure A.1 with 20 000 episodes and we track the optimal value V^*. For each of the 5 functions, we repeat the same experiment 7 times in order to figure out more clearly how the learning algorithm converges to the optimal value. Hence, each of the following figures shows 7 plots obtained after repeating the same task 7 times. The figures are centered around the optimal value $V^* = 0.262$ represented by a line. Note also that the X axis is labelled to 5000 episodes and not 20 000 because for plotting, we sampled the Q-values every 4 episodes.

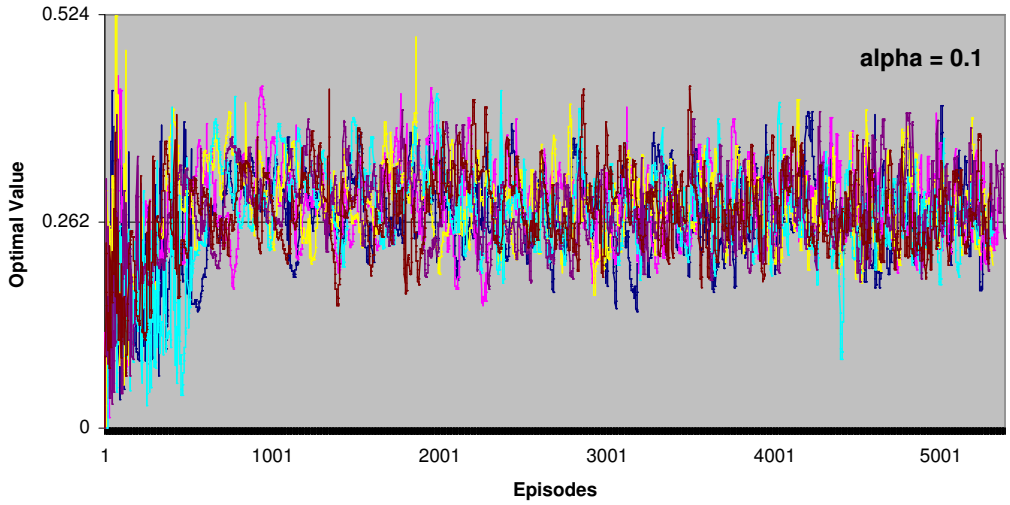


Figure A.2: Learning rate $\alpha = 0.1$. With a constant α, the learning algorithm does not converge.

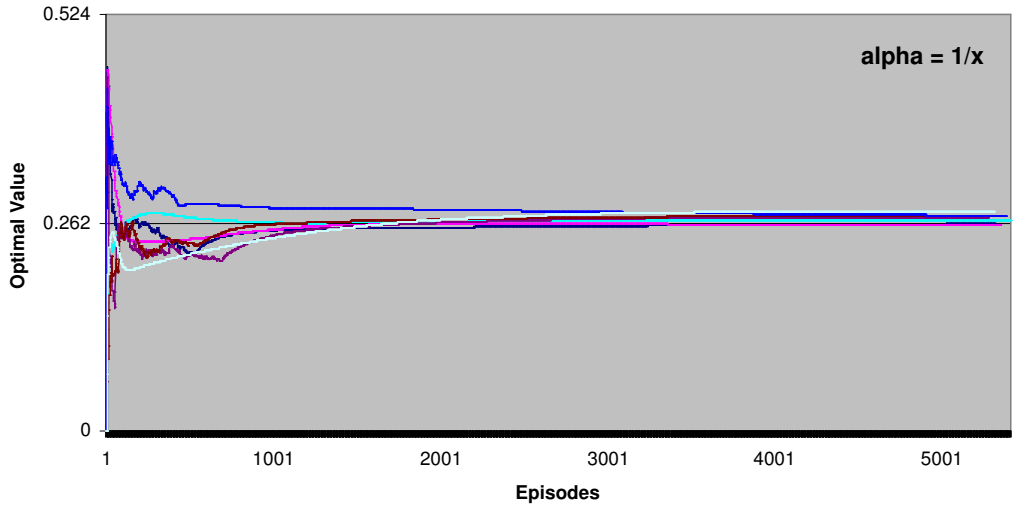


Figure A.3: Learning rate α decreasing according to function $1/x$. With function $\frac{1}{x}$, we obtain the best convergence results: the plots become stable after 4000 episodes.

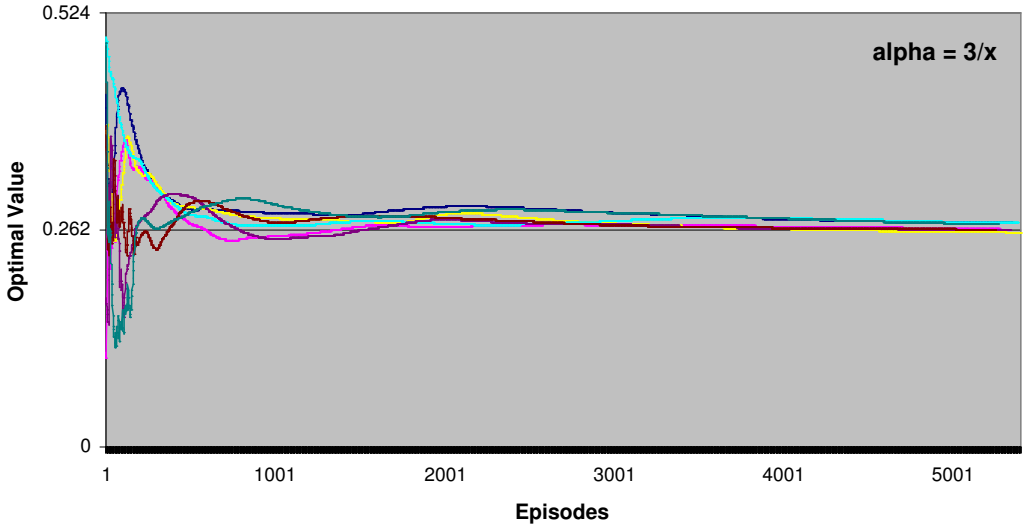


Figure A.4: Learning rate α decreasing according to function $3/x$. The obtained results are good but not as good as $\frac{1}{x}$.

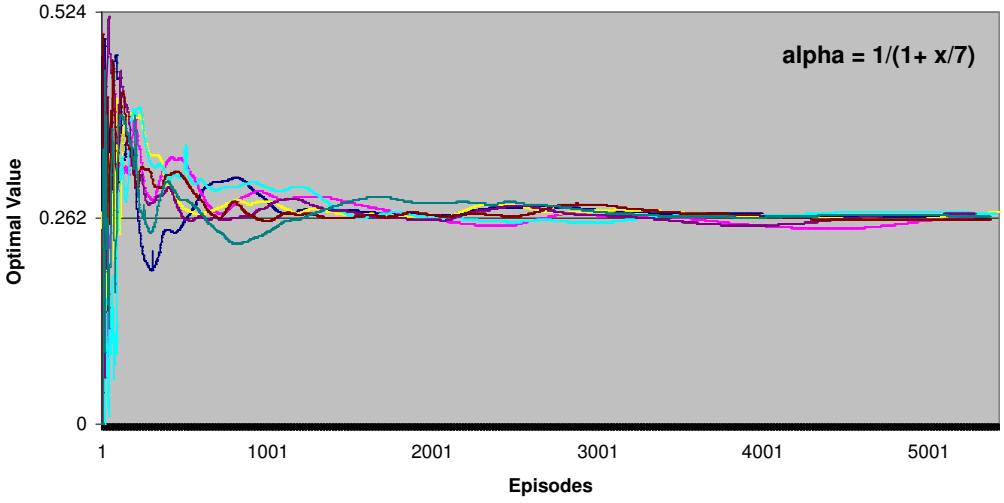


Figure A.5: Learning rate α decreasing according to the function $\dfrac{1}{1+\frac{x}{7}}$.

A.2 Epsilon : ϵ

When using the ϵ-greedy method to select actions, the question is what is the best value for ϵ? In this experiment, we tried the task of Figure A.1 with different ϵ values, namely, a constant $\epsilon = 0.1$, a variable ϵ decreasing from 0.3 to 0.1, and from 0.8 to 0.1. ϵ is decreased according to the function $\epsilon = \frac{k}{\text{currentEpisode}+l}$ where k and l are constants computed in terms of the number of episodes as well as the first and the last values of ϵ. For each ϵ value or function, we show the results in two figures. The first is obtained by repeating the task of Figure A.1 7 times with 20 000 episodes each time and tracking the optimal value whereas the second figure is obtained by repeating the same task 100 times but showing only the final optimal value. Both figures are centered around the optimal value $V^* = 0.262$, represented with a line.

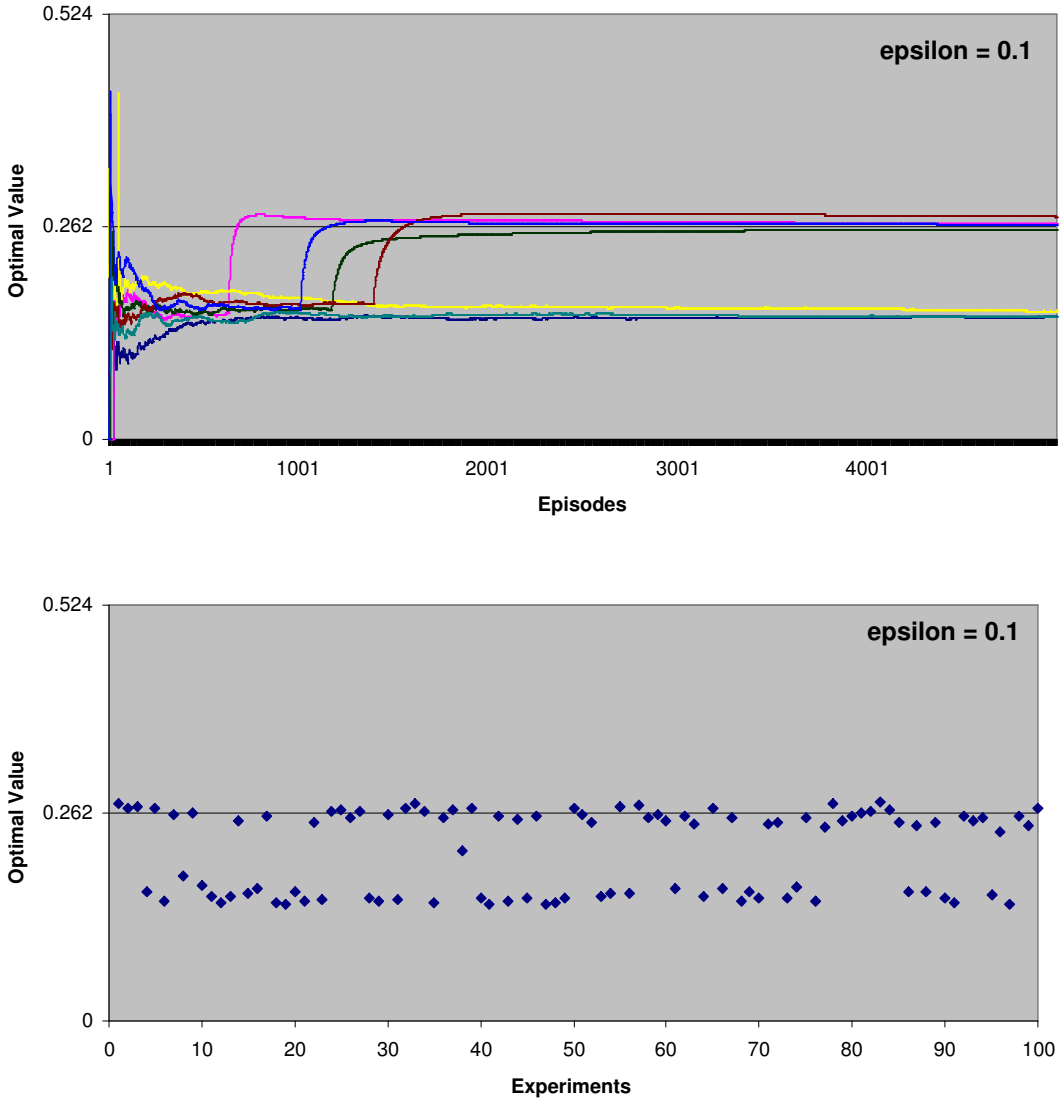


Figure A.6: The action selection method used is ϵ-greedy with a fixed $\epsilon = 0.1$. (Upper figure) In 3 out of 7 experiments, the learning algorithm converges to the sub-optimal policy π. This is due to insufficient exploration mainly in the first phases of the task. (Lower figure) The upper dots indicate that the learning algorithm converged to the optimal value V^{π^*} in 55 out of 100 experiments, while the lower dots indicate that the learning algorithm prematurely converges to the sub-optimal policy π in 45 out of 100 experiments.

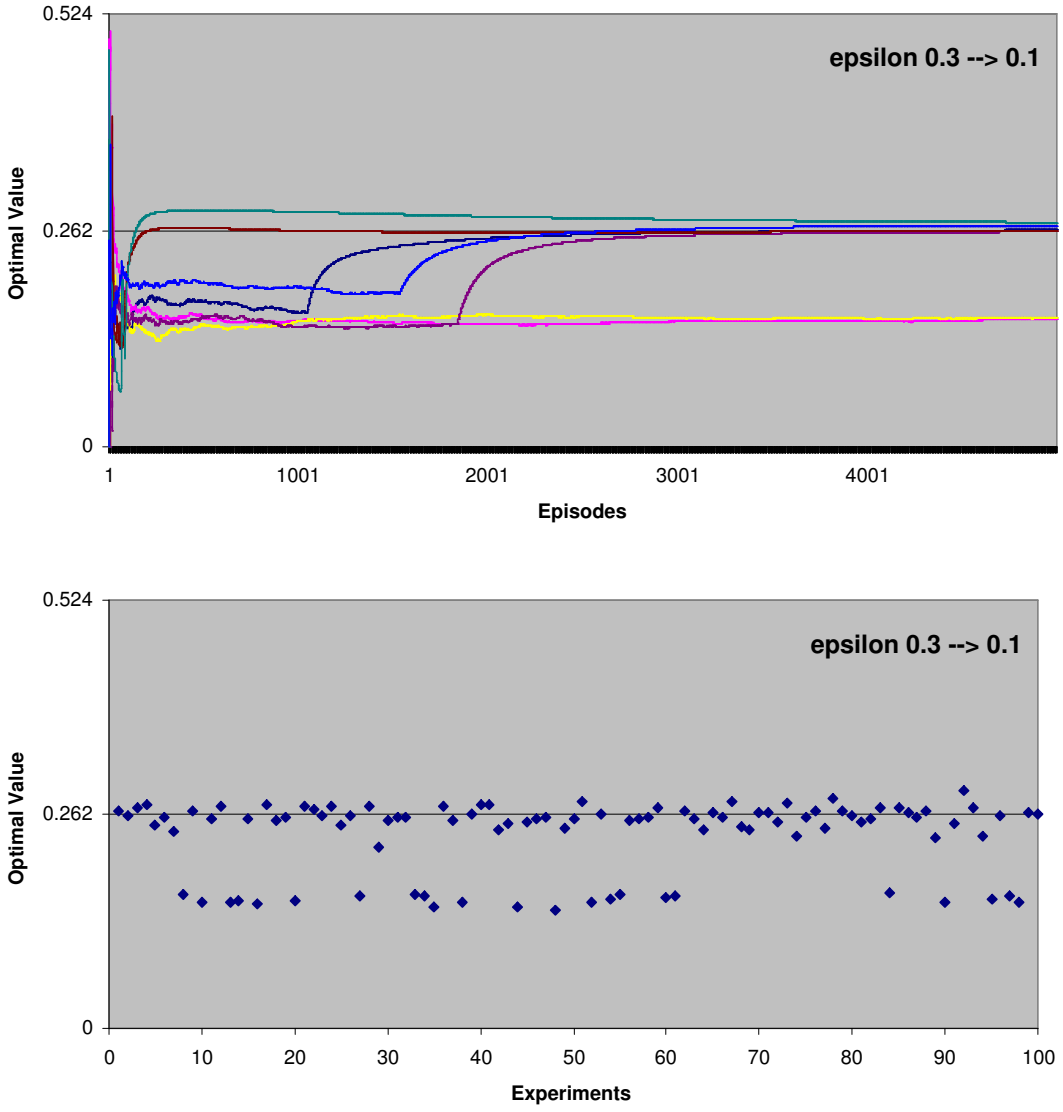


Figure A.7: The action selection method used is ϵ-greedy with an ϵ decreasing from 0.3 to 0.1. (Upper figure) An ϵ decreasing from 0.3 to 0.01 will result in more exploration in the first phase, which will make the learning algorithm discover the optimal policy π^* most of the time. In 2 out of 7 experiments, the algorithm prematurely converges to the sub-optimal policy π. (Lower figure) In 23 experiments the learning algorithm converges to the sub-optimal policy. Hence, still more exploration is required.

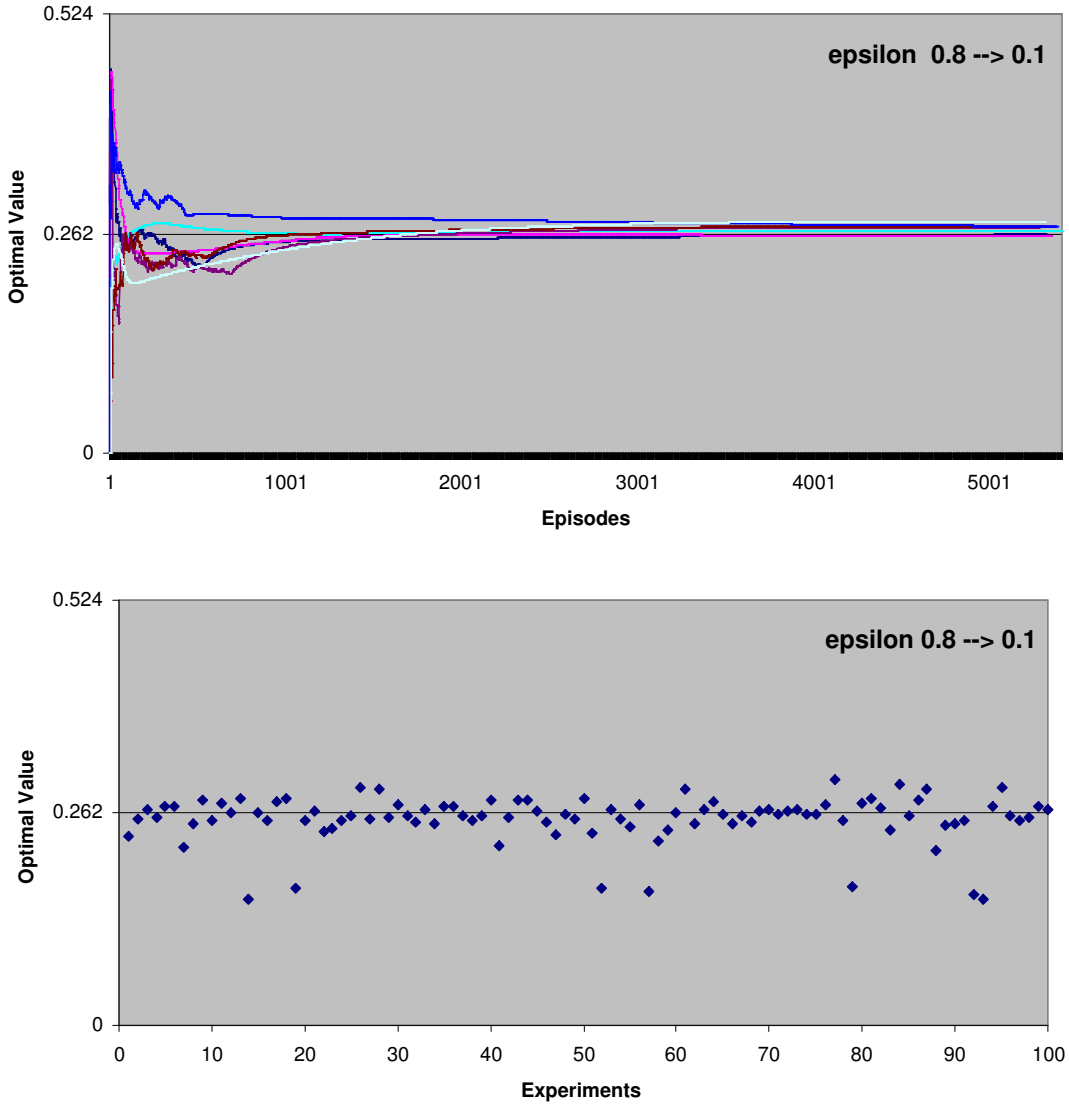


Figure A.8: The action selection method used is ϵ-greedy with an ϵ decreasing from 0.8 to 0.1. (Upper figure) With ϵ decreasing from 0.8 to 0.01, we obtain the best results for ϵ-greedy. In all of the 7 experiments, the algorithm converges to the optimal policy π^*. (Lower figure) In only 7 experiments out of 100 the learning algorithm converges to the sub-optimal policy. However, in the remaining 93 experiments, one can note the high variance of the optimal values.

A.3 Temperature : τ

If the Softmax action selection method is used, the parameter to tune is the temperature τ. Recall that, the bigger τ is, more equiprobable the actions will be, and the smaller is it, the more accentuated the probabilities of actions with different Q-values will be. We tried Softmax with 3 different τ values, namely, 1, 5, and 0.01 in addition to a variable τ decreasing from 5 to 0.01.

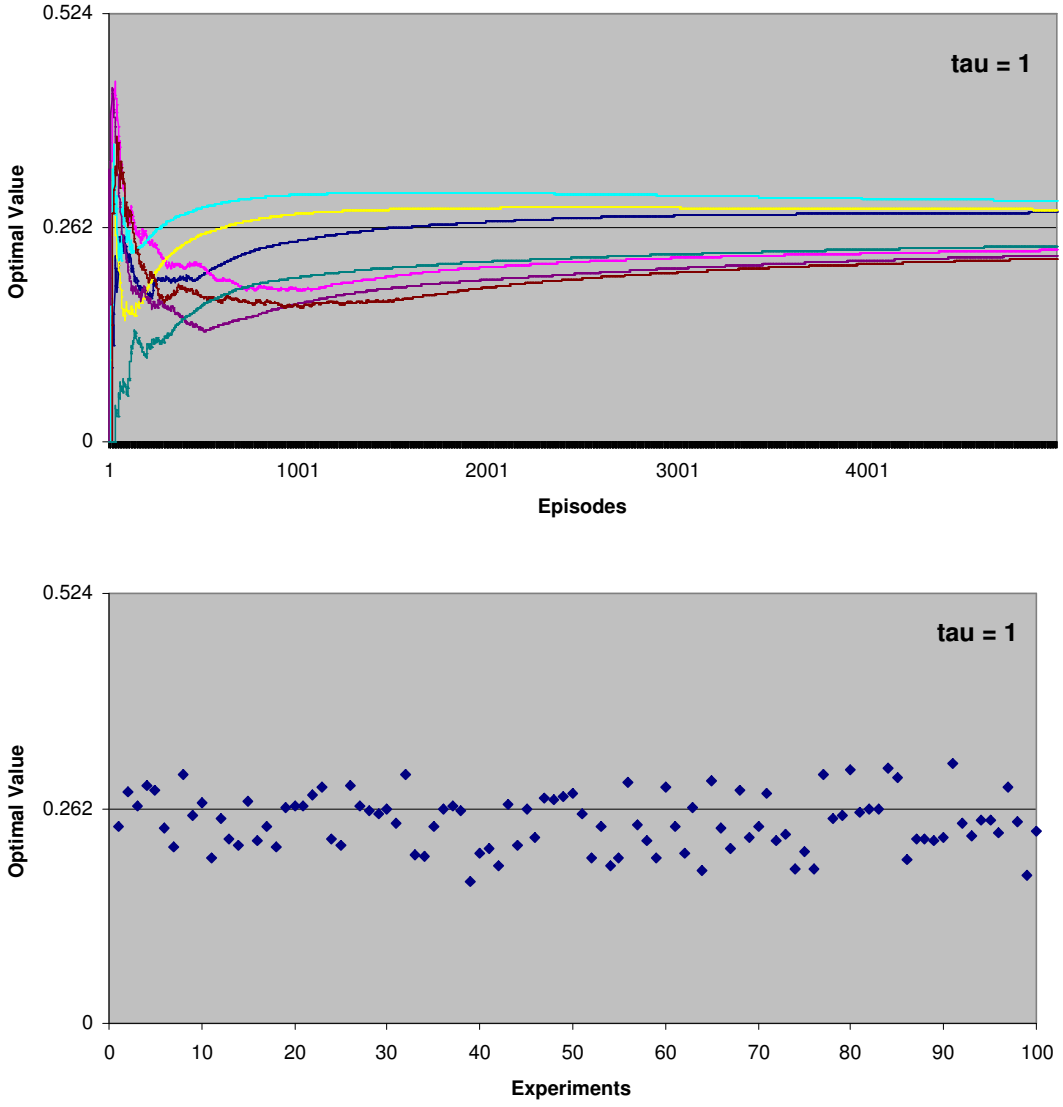


Figure A.9: The action selection method is Softmax with a fixed $\tau = 1$. (Upper figure) With $\tau = 1$, the learning algorithm finds the optimal policy π^* in all experiments since it explores very well. However, it does not exploit enough this optimal policy because it continues exploration with the same rate until the end. (Lower figure) Most of the time, the learning algorithm finds the optimal policy but it does not evaluate the policy enough, which results in sparse dots around the optimal value.

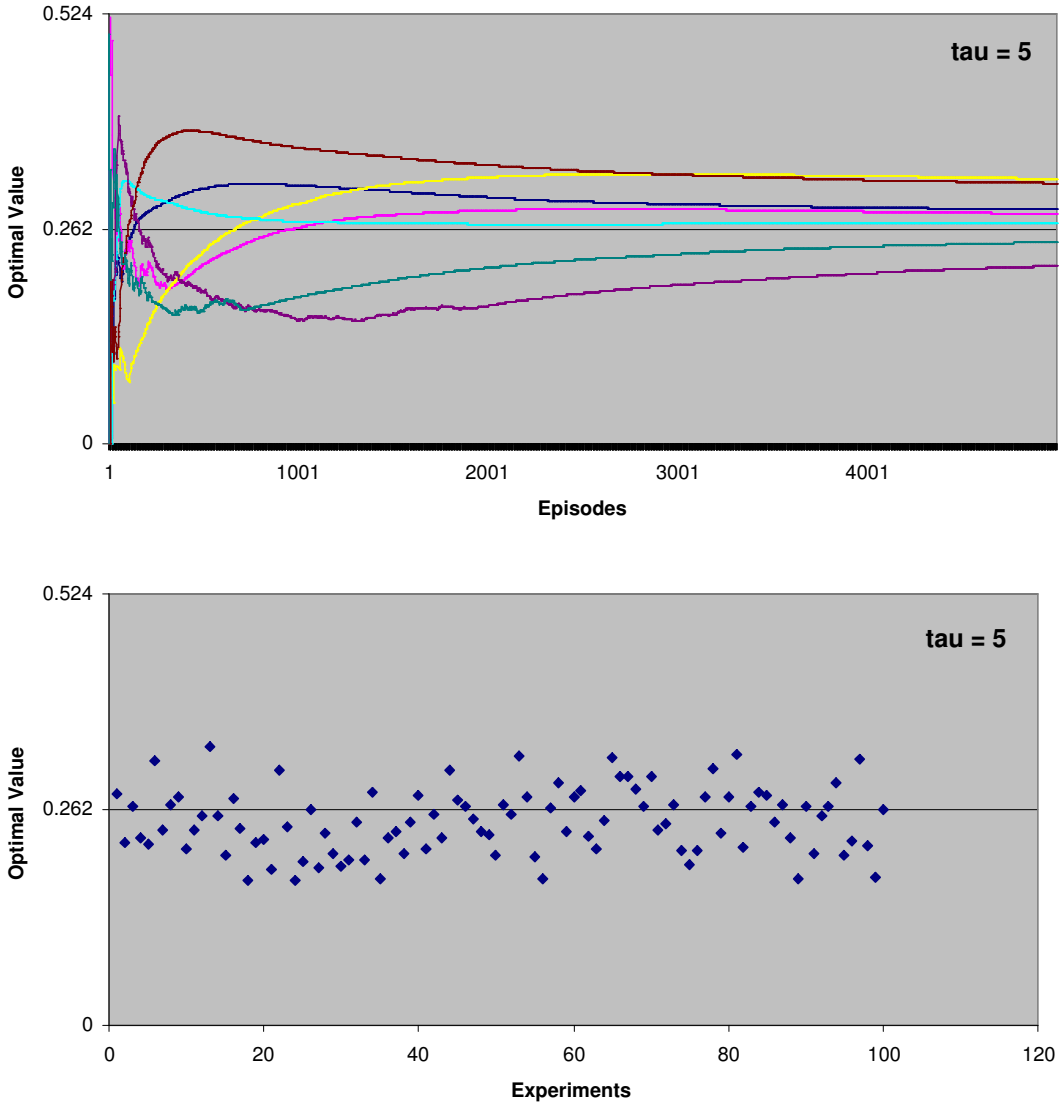


Figure A.10: The action selection method is Softmax with a fixed $\tau = 5$. (Upper figure) With $\tau = 5$, actions with different values will be equiprobable. This is good in the first phase where exploration is required. However, continuing the same strategy until the end will prevent having an accurate value for the optimal policy. Hence, with $\tau = 5$, the learning algorithm lacks greediness in the later phase. (Lower figure) The dots indicate higher variance in the optimal value estimates than with $\tau = 1$.

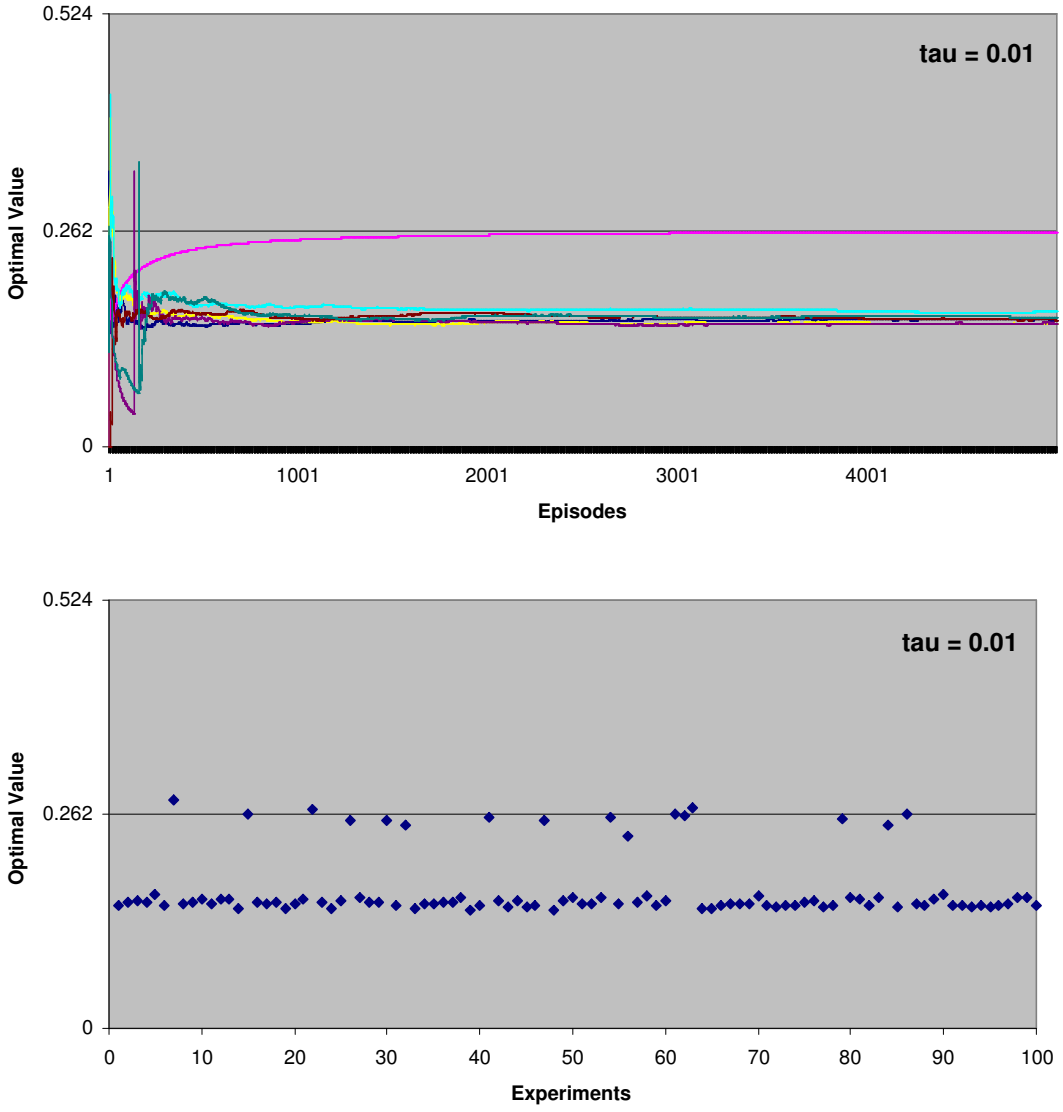


Figure A.11: The action selection method is Softmax with a fixed $\tau = 0.01$. (Upper figure) With $\tau = 0.01$, the learning algorithm does not do enough exploration. In 6 out of 7 experiments, it prematurely converges to the sub-optimal policy π. However, in the single case where it converges to the optimal policy π^*, it returns a very accurate estimation of its value. (Lower figure) In 84 out of 100 experiments, the algorithm prematurely converges to the sub-optimal policy.

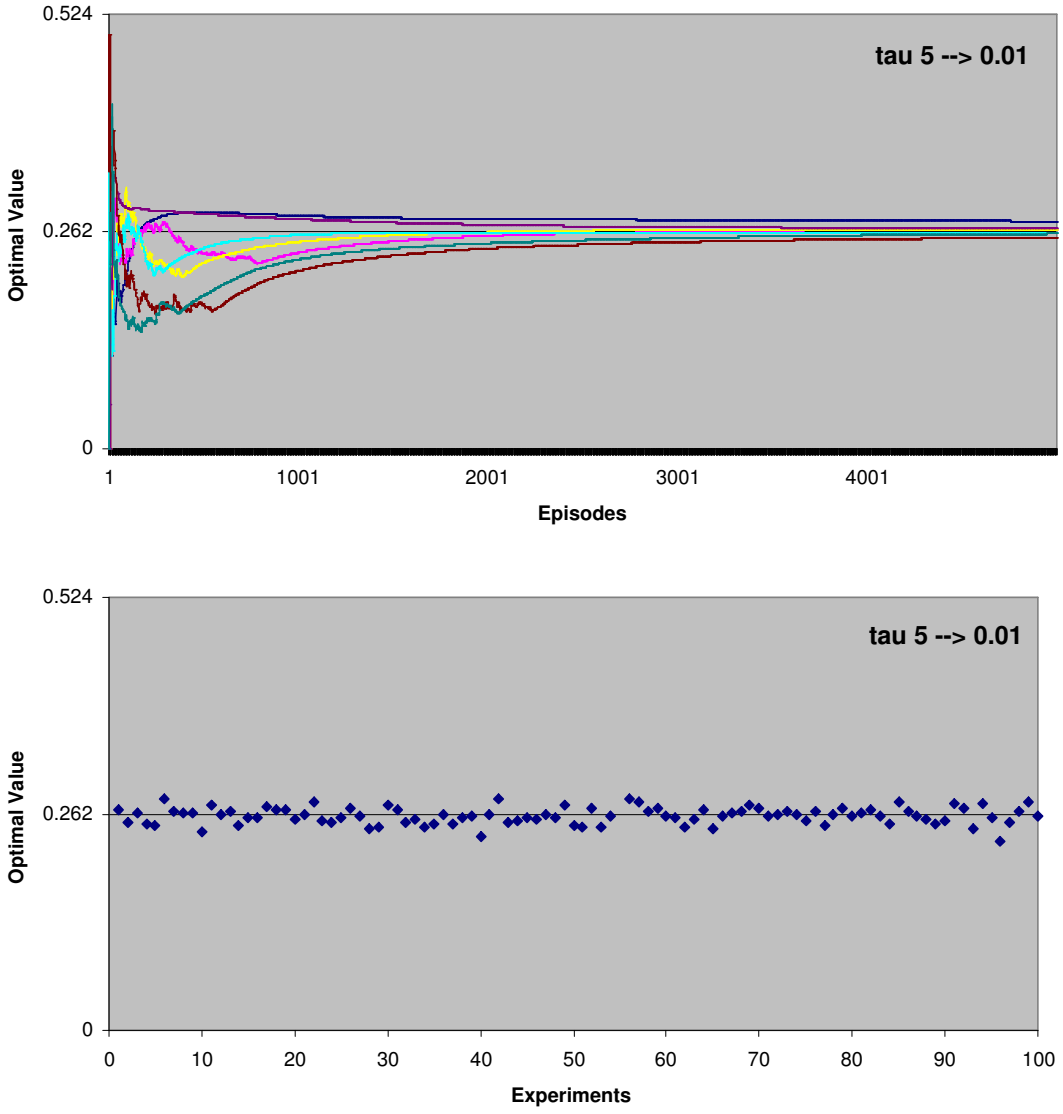


Figure A.12: The action selection method is Softmax with a temperature τ decreasing from 5 to 0.01 according to the function $\tau = \frac{k}{\text{currentEpisode}+l}$ where k and l are constants computed in terms of the number of episodes as well as the first and the last values of τ (5 and 0.01). (Upper figure) With τ decreasing from 5 to 0.01, we obtain the best tradeoff between exploration and exploitation. (Lower figure) In all 100 experiments, the learning algorithm converges to the optimal policy.

Appendix B

HMM Use Cases

B.1 Isolated Word Recognizer

One of the first and most successful utilization of HMM was in automatic speech recognition [41, 42]. In automatic speech recognition, one is interested in identifying the words articulated by a speaker. Obviously, the main difficulty in such systems lies in the diversity of human voices which are characterized by different intonations, spectral and temporal specificities. In the following, we show how HMM theory gets applied in the design of a specific component of an automatic speech recognition system which is the isolated word recognizer. As its name indicates, this component is in charge of recognizing words uttered by a speaker. It takes as input a speech signal and returns the word in the vocabulary corresponding to the sound signal. The key idea is to model each word of the vocabulary as a distinct HMM and to consider each speech signal as a sequence of observations. In this setting, the system will return as a result the word of the vocabulary whose HMM maximizes the probability to accept the sequence of observations given as input. Hence, the first step is to come up with a HMM for each word of the vocabulary (we suppose that the vocabulary contains v words). Then each speech signal must be processed according to Figure B.1. For this, the speech signal is transformed into a sequence of short sounds (observations). Since it is of no particular interest to the current illustration, we will not go through the details of this step. Then, this sequence of observations is evaluated in each of the v HMMs of the vocabulary by computing its probability to be accepted $Pr(O|HMM_i)$. The probability computation is generally performed using the Viterbi algorithm [15, 60]. The word whose HMM accepts the observation sequence with the maximum of probability is returned as a result.

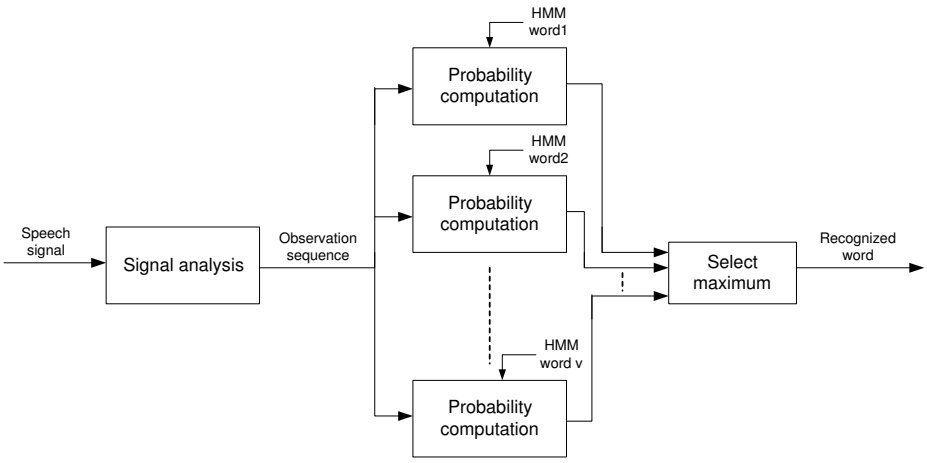


Figure B.1: Steps of the word recognizer.

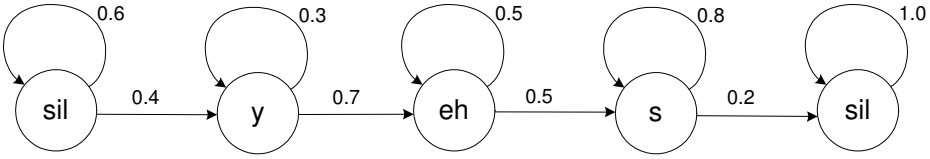


Figure B.2: HMM representation of the word "yes".

It is easy to see that the reliability of the isolated word recognizer depends particularly on the quality of the HMM of each word. A good word model would assign high probability to all sound sequences that are most likely pronunciations of the word it models, and low probability to any other sequence. In order to construct the HMM for a given word of the vocabulary, one needs to have the word uttered by a representative set of speakers (male, female, serious voice, etc). This set is called the *training set*. The model parameters of the HMM are chosen in such a way to optimize the acceptance likelihood for all sequences of the training set. Hence, the more representative the training set is, the more reliable the constructed HMM will be. Let us go deeper in the details and see the steps of the HMM construction.

The states of the HMM are generally chosen to be the phonemes of the word. A phoneme is a sound of a language as represented without reference to its position in a word. Hence, it is a conception of a sound in the most natural form possible. In other approaches, a state corresponds to an observation interval (about 15 milliseconds). Figure B.2 illustrates a HMM corresponding to the word "yes". Notice also that the HMM has a particular form. This type of HMMs is called *left-right* model where all transitions are from left to right. It is the more appropriate model for this task since it allows to associate time to model states.

The first step of the construction is the initialization of the model. Generally, the initial parameters of the model are chosen randomly. Let this initial model be called λ_0. Then, the following loop is entered:

```
Repeat
    j = j + 1
    For every sample of the training set
        Segment the observation sequence between the states of λ_j
        Estimate the parameters of the model
    End
    New model λ_{j+1}
Until (div(λ_j, λ_{j+1}) < ε)
```

Hence, for every sample of the training set, the observations are segmented between the states. Then, the model parameters are updated according to this segmentation. The HMM parameters get updated based on the training set until convergence, that is, until the divergence between two successive models become insignificant (less than a certain ϵ). After going through all samples of the training set, a new model is obtained (λ_{j+1}). This model is compared with the old one. If the divergence is more than a given threshold ϵ, the old model is replaced by the new one and the process is repeated. The loop stops when the divergence between two successive models become negligible.

B.2 DNA Sequence Analysis

Another success story of HMMs can be found in bioinformatics. Indeed, HMMs have been very helpful in several DNA and protein analysis related problems [14].

The entire biological information necessary for a living organism to generate new cells is contained in its DNA (Deoxyribonucleic Acid). DNA can be viewed as a huge sequence of information encoded using 4 nucleobases: Adenin (A), Cytosine (C), Guanine (G), and Thymine (T). For example, the human DNA is a sequence of around 3 billions nucleobases located in 23 chromosomes. Biological information is organized inside the DNA in the form of genes. A gene is a segment of DNA which represents a particular characteristic of the living organism. The position, the limits, and the role of each gene are the subject of extensive research in bioinformatics. Several intrinsic aspects of the DNA make these kinds of tasks (and some others) difficult to undertake. First, not all DNA sequences are useful and code genes: only 30% of DNA represents genes. All

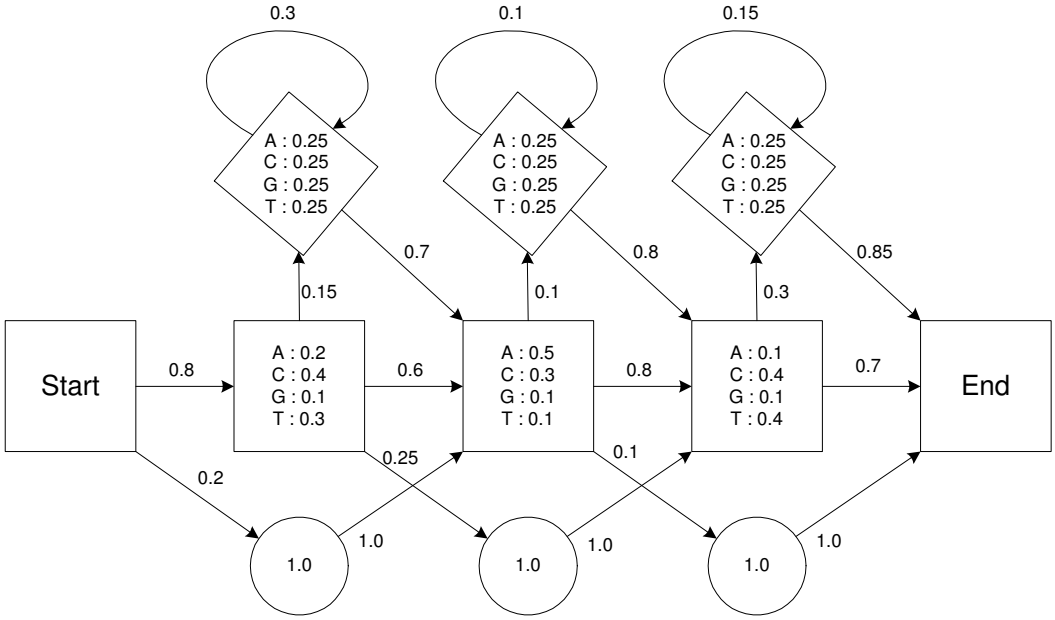


Figure B.3: HMM of a DNA sequence with insertions and deletions.

the rest of the bases are called intergenic blocks and are of no "apparently" usefulness. Second, within the segment of a particular gene, there are three kinds of blocks: Introns, Exons, and regulatory sequences. Introns do not represent any apparently useful information, Exons code the gene information, and the regulatory sequence specifies how this particular gene is expressed in the hosting cell (whether a gene is active in a cell depends on the type of the cell). Third, and more importantly, there are several modifications (insertion, deletion, and substitution of Nucleobases) that happen within DNA from one generation to the next. Currently, there is no explanation for these modifications except the evolution theory; therefore these phenomena are considered as perfectly random.

In this context, the HMM model turns out to be very helpful in the tasks of detecting similarities between sequences and identifying the general pattern of a given gene. The following example illustrates how HMMs can be used to recognize genes (or sequences of Nucleobases) belonging to the same family in the presence of randomly deleted and inserted information. First, suppose that the sequence of Nucleobases to be analyzed has been precisely delimited. That is, the Intron and regulatory sequences have been discarded. The sequence is viewed as a sequence of observations. Hence, we have an observation set of size 4: A, C, G, and T. The HMM corresponding to the sequence will have a form similar to Figure B.3. The first kind of states are match states (the boxes in the middle). Each of the states can generate an observation with a corresponding probability. The second type of states is the delete states (circles in the bottom) which

allow to skip a match state, hence representing the deletion of Nucleobase. This type of state does not generate an observation. The third type of states is the insertion states (diamonds) which represents the insertion of Nucleobases. The self loop in the insertion states models the possibility of multiple insertions between two match states. Each state has 1, 2, or 3 outgoing transitions to other states, so each state defines a probability distribution on the next states. Since match and insertion states can generate observations, each one of them defines a probability distribution on the 4 possible observations A, C, G, T.

Once the HMM of a given gene or a family of genes is constructed, it can be used to identify sequences of that family. Indeed, for each unknown sequence, one can simply evaluate it in the model (HMM) and see how it scores. The score is the probability of the sequence given the model. The question that remains to be answered is how to estimate the parameters (observation and transition probabilities) of the HMM. Like on the speech recognition case, the HMM parameters are estimated based on a training set of sequences. The estimation starts with a randomly selected initial model, which is then reestimated based on the training set until convergence (that is, until there are no further changes to the model).