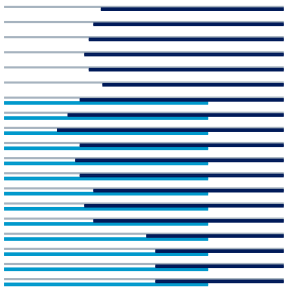
A Taxonomy for Abstract Environments

Shane Legg
Marcus Hutter



A Taxonomy for Abstract Environments No. IDSIA-20-04
November 9, 2004

IDSIA / USI-SUPSI
Dalle Molle Institute for Artificial Intelligence
Galleria 2, 6928 Manno, Switzerland

A Taxonomy for Abstract Environments*

Shane Legg†
Marcus Hutter‡

November 9, 2004

Abstract

A wide range of problems can be abstractly expressed in terms of two systems interacting over time with the second system providing feedback to the first on its success in achieving some goal. The agent–environment model in reinforment learning and the controller–plant model from control theory are both examples of this framework. Problems which can be expressed in terms of this framework include statistical sampling processes, prediction problems, classification problems, decision problems, games and function optimisation problems. While these problem classes are very common and strong relationships between them are apparent, they are not usually considered *in toto*. Here we formally define many of these common problem classes as environments using the chronological system formalism. From this we establish the elementary relationships between the problem classes giving a taxonomy of abstract environments.

1 Agent-Environment Model

1.1 Introduction

The agent-environment model is very simple and yet surprisingly general. It consists of two entities called the *agent* and the *environment* which communicate with each other. The agent receives input information from the environment, which we will refer to as *perceptions*, and sends output information back to the environment, which we call *actions*. The environment on the other hand receives actions from the agent as input and generates perceptions as output. The model says nothing about how the agent or the environment work; it only describes their role within the framework and thus many different environments and agents are possible.

The model does have one additional and important complexity: each perception can be split into an *observation* component and a *reward* component. Observations are just regular information, however rewards have a special significance because the goal of the agent is to try and gain as much reward as possible from the environment. The only way that the agent can influence the environment is through the action signals. Thus a good agent is one that

*This work was supported by SNF grant 2100-67712.02.
†shane@idsia.ch
‡marcus@idsia.ch

intelligently selects its actions so as to cause the environment to generate as much reward as possible. Presumably such an agent will make good use of any useful information contained in past rewards, actions and observations. For example, the agent might find that certain actions tend to produce rewards while others do not. In more complex environments the relationship between the agent's actions, what it observes and the rewards it receives might be very complex and difficult to discover. The basic structure of this agent-environment interaction model is illustrated in Figure 1.

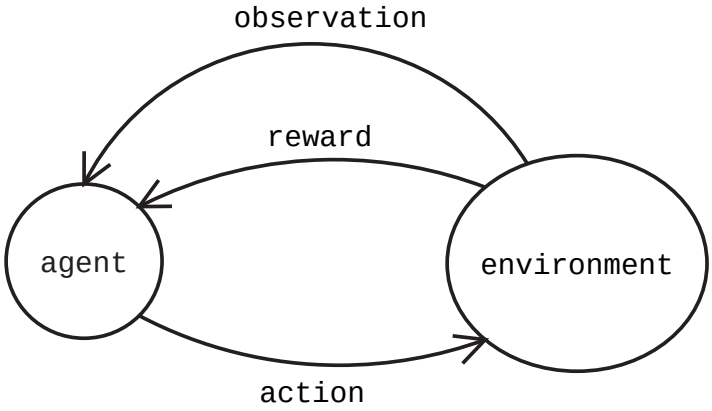


Figure 1: The agent and the environment interact by sending action, observation and reward signals to each other.

Before formalising this model a simple example will be helpful. We will be referring back to this example and the formalised version that follows numerous times though the text.

1.1 Example. We will call this the "Two coins game". In each cycle of the game two 50¢ coins are tossed. Before the coins settle the player must guess at the number of heads that will result: either 0, 1, or 2. If the guess is correct the player gets to keep both coins and then two new coins are produced and the game is repeated. If the guess is incorrect the player does not receive any coins and the game is also repeated.

In terms of the agent-environment model, the player is the agent and the system which produces all the coins, tosses them and distributes the reward when appropriate is the environment. The agent's actions are its guesses at the number of heads in each iteration of the game: 0, 1 or 2. The observation is the state of the coins when they settle, and the reward is either \$0 or \$1.

It is easy to see that for unbiased coins the most likely outcome is 1 head and thus the optimal strategy for the agent is to always guess 1. However if the coins are significantly biased it might be optimal to guess either 0 or 2 heads depending on the bias. If this were the case then after a number of iterations of the game an intelligent agent would realise that the coins were probably biased and change its strategy accordingly.

While this is a very simple example, with a little inventiveness it's not hard to formulate just about any sort of game or problem situation in terms of this framework: chess playing,

speech recognition, stock market trading, writing a hit song or even trying to pass a Turing test can all be expressed within this framework without too much difficulty (see [10] for many examples in the context of reinforcement learning). Of course none of this says anything about how an agent should work in order to achieve these things. Indeed from the perspective of this framework AI may be thought of as the search for agents which perform well within certain classes of environments. To do this we need to understand the nature of different types of environments.

In this document we approach this problem from a very abstract perspective by mathematically defining a range of environmental classes and establishing the basic formal relationships between them. This section presents the agent–environment model and formalises it in terms of chronological systems similar to [6] and [5]. The second section formally defines a number of common environmental classes and basic properties. The third section proves the basic relationships between the classes. The appendix contains a description of our mathematical formalism and some additional analysis.

1.2 Formal Agent-Environment Model

We will now formalise the Agent-Environment model presented above.

1.2 Definition. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two recursive prefix free languages and let $\pi_1, \pi_2, \ldots$ be total functions such that $\forall i \in \mathbb{N}$, $\forall z \in (\mathcal{L}_1 \times \mathcal{L}_2)^{i-1}$ we have $\pi_i : (\mathcal{L}_1 \times \mathcal{L}_2)^{i-1} \times \mathcal{L}_1 \to [0,1]$ and $\sum_{a \in \mathcal{L}_1} \pi_i(z\underline{a}) = 1$. Thus each π_i is a conditional probabilty measure. A **Type 1 Chronological System** is a total function $\pi : (\mathcal{L}_1 \times \mathcal{L}_2)^* \times \mathcal{L}_1 \to [0,1]$ defined $\forall i \in \mathbb{N}$, $\forall z \in (\mathcal{L}_1 \times \mathcal{L}_2)^{i-1}, \forall a \in \mathcal{L}_1 : \pi(z\underline{a}) := \pi_i(z\underline{a})$.

Informally, a type 1 chronological system describes the probability of every second symbol in a sequence of symbols from two alternating languages starting from the first symbol. To describe the remaining symbols of the sequence we need its mirror image:

1.3 Definition. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two recursive prefix free languages and let $\mu_1, \mu_2, \ldots$ be total functions such that $\forall i \in \mathbb{N}$, $\forall z \in (\mathcal{L}_1 \times \mathcal{L}_2)^{i-1} \times \mathcal{L}_1$ we have $\mu_i : (\mathcal{L}_1 \times \mathcal{L}_2)^i \to [0,1]$ and $\sum_{b \in \mathcal{L}_2} \mu_i(z\underline{b}) = 1$. Thus each μ_i is a conditional probabilty measure. A **Type 2 Chronological System** is a total function $\mu : (\mathcal{L}_1 \times \mathcal{L}_2)^+ \to [0,1]$ defined $\forall i \in \mathbb{N}$, $\forall z \in (\mathcal{L}_1 \times \mathcal{L}_2)^{i-1} \times \mathcal{L}_1, \forall b \in \mathcal{L}_2 : \mu(z\underline{b}) := \mu_i(z\underline{b})$.

Thus a type 2 chronological system is the same as a type 1 chronological system except that it describes the probabilities of every second symbol starting from the second symbol. If we have a type 1 and a type 2 chronological system for a sequence then we have the distribution of the next symbol at any point during the generation of the sequence. Clearly these two types of chronological systems can play the role of two interacting systems as described informally in the previous section.

1.4 Definition. A **Chronological Agent** is a tuple $(\mathcal{X}, \mathcal{Y}, \pi)$ where $\mathcal{X}$ is a recursive prefix free language called the **perception space**, $\mathcal{Y}$ is a recursive prefix free language called the **action space** and π is a type 1 chronological system $\pi : (\mathcal{Y} \times \mathcal{X})^* \times \mathcal{Y} \to [0,1]$ called

the **policy**. If the action and perception spaces are finite then we call the system a **Finite Chronological Agent**.

As the policy π can be any chronological system it can describe a deterministic system simply by setting the probably of one of the output symbols to be 1 in each cycle. Often the policies that we will be considering will be deterministic and in these cases we will often talk about the policy outputting some particular symbol where strictly speaking the policy indicates this output through a distribution over the action space. While describing the agent as a chronological system has the advantage of keeping the framework as general as possible, it also gives the model an elegant symmetry:

1.5 Definition. A **Chronological Environment** is a tuple $(\mathcal{Y}, \mathcal{X}, \mu)$ where $\mathcal{Y}$ is a recursive prefix free language called the **action space**, $\mathcal{X}$ is a recursive prefix free language called the **perception space**, and $\mu : (\mathcal{Y} \times \mathcal{X})^* \to [0, 1]$ is a type 2 chronological system. We further define $\mathcal{X} := \mathcal{R} \times \mathcal{O}$ where $\mathcal{R}$ is the **reward space** and $\mathcal{O}$ is the **observation space**. If the action and perception spaces are finite we call the system a **Finite Chronological Environment**.

Whenever in the text we write $\mathcal{Y}$, $\mathcal{X}$, $\mathcal{R}$ or $\mathcal{O}$ we will assume that they fit the above definitions for a chronological environment. We will also use lower case variables such as y_k, x_k, r_k and o_k to denote elements of these sets. In particular whenever we have $x_k \in \mathcal{X}$ we take this to be defined $x_k := r_k o_k$ and simply write r_k or o_k to mean the corresponding components of x_k, that is, the components of the perception in the k^{th} cycle.

When we need an enumeration on the action space $\mathcal{Y}$ we will denote this as $y^{(1)}, y^{(2)}, y^{(3)}, \ldots$. We will also use similar notation for enumerations on the perception and observation spaces where required.

Strictly speaking r_k is an element of the reward space $\mathcal{R}$ which is a recursive prefix free language. Thus r_k is a string. However we are really only interested in the value that r_k represents when mapped to $\mathbb{R}$ by some simple recursive injective function $R : \mathcal{R} \to \mathbb{R}$. As R performs only a technical role, rather than always writing $R(r_k)$ we will interpret r_k to be its associated real value were convenient.

The following is a useful technical condition. Essentially it just says that all the perceptions in the perception space are actually possible at least to start with.

1.6 Definition. A chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ is **accessible** if $\forall o \in \mathcal{O}, \exists k \in \mathbb{N}, \exists y x_{<k} y_k \in (\mathcal{Y} \times \mathcal{X})^{k-1} \times \mathcal{Y}$ such that $\mu(y x_{<k} y_k \underline{o}) > 0$.

This is not a serious condition to impose as we can always reduce an non-accessible environment to an equivalent accessible environment simply by dropping the unused observations from the observation space $\mathcal{O}$. However it does clean up some of the results as we don't have to account for observations that can never actually occur. Thus we will always assume that the environments that we are working with are accessible in the above sense. We have defined this condition over o rather than x because normally only a subset of $\mathcal{O} \times \mathcal{R}$ is possible with a given system.

Finally we now have all the required components to define the AI framework that we will be working with:

1.7 Definition. The **Chronological Agent-Environment Model** consists of an agent $(\mathcal{X}, \mathcal{Y}, \pi)$ and an environment $(\mathcal{Y}, \mathcal{X}, \mu)$ entangled so that the output of one forms the input to the other and vice versa. Together they generate a sequence of actions and perceptions in the following way: Firstly the agent generates an action y_1 according to its policy $\pi(\underline{y}_1)$. In response the environment generates a perception x_1 according to its conditional distribution $\mu(y_1\underline{x}_1)$. The agent then generates another action, this time according to $\pi(y_1x_1\underline{y}_2)$ followed by another perception from the environment according to $\mu(y_1x_1y_2\underline{x}_2)$. This process continues indefinitely with each new symbol being conditioned by the current shared interaction history of actions and perceptions.

In our model we have chosen to have the agent start the interaction sequence by producing an action which is then followed by a perception generated by the environment. Alternatively we could have the environment starting followed by the agent. Which of these alternatives seems the most natural depends on the situation being modelling and perhaps personal preference. At any rate the difference is rather technical and the resulting analysis remains largely the same.

We are now able to formalise the two coins game in Example 1.1.

1.8 Example. Let $\mathcal{O} := \{0, 1, 2\}$ be the observation space representing the number of heads after tossing the two coins. Likewise let $\mathcal{Y} := \{0, 1, 2\}$ be the action space representing the agent's guess at the number of heads that will occur. The reward space is simply $\mathcal{R} := \{\$0, \$1\}$.

We can represent the environment by the chronological system $\mu : (\mathcal{Y} \times \mathcal{X})^* \to [0, 1]$ defined $\forall k \in \mathbb{N}, \forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$,

$$\mu(yx_{<k}y_k\underline{x}_k) = \mu(y_k\underline{x}_k) := \begin{cases} \frac{1}{4} & \text{if } y_k = o_k \in \{0, 2\} \wedge r = \$1, \\ \frac{3}{4} & \text{if } y_k \neq o_k \in \{0, 2\} \wedge r = \$0, \\ \frac{1}{2} & \text{if } y_k = o_k = 1 \wedge r = \$1, \\ \frac{1}{2} & \text{if } y_k \neq o_k = 1 \wedge r = \$0, \\ 0 & \text{otherwise.} \end{cases}$$

where as usual $x_k := o_kr_k$.

An optimal policy for this environment can similarly be represented by the chronological measure $\pi : (\mathcal{Y} \times \mathcal{X})^* \times \mathcal{Y} \to [0, 1]$ defined $\forall k \in \mathbb{N}, \forall yx_{<k}y_k \in (\mathcal{Y} \times \mathcal{X})^{k-1} \times \mathcal{Y}$,

$$\pi(yx_{<k}\underline{y}_k) = \pi(\underline{y}_k) := \begin{cases} 1 & \text{for } y_k = 1, \\ 0 & \text{otherwise.} \end{cases}$$

That is, always guess that 1 head will be the result of the two coins being tossed.

We see in this example how even a very simple game like guessing the outcome of two coins being tossed becomes a little messy when fully formalised in terms of chronological systems. Usually one would not describe either an environment or agent in this way for this reason, however it will be useful for our purposes as many fundamental classes of environments can be succinctly described in this formalism as we will see in the next section.

Given an agent and an environment we can take the environment's chronological system μ and the agent's policy π and fuse them together to give us a measure over the space of agent-environment interactions. Call this probability measure $\tau : \mathcal{S} \to [0, 1]$ where,

$$\mathcal{S} := (\mathcal{Y} \times \mathcal{X})^* \cup \mathcal{Y} \times (\mathcal{X} \times \mathcal{Y})^*.$$

It is not hard to see that $\mathcal{S}$ is a sigma algebra, indeed it contains all possible strings from the space of alternating words from the languages $\mathcal{Y}$ and $\mathcal{X}$. From the multiplication rule for probabilities we know that,

$$\tau(\underline{y}_1\underline{x}_1\underline{y}_2\underline{x}_2 \ldots) = \tau(\underline{y}_1) \cdot \tau(y_1\underline{x}_1) \cdot \tau(y_1x_1\underline{y}_2) \cdot \tau(y_1x_1y_2\underline{x}_2) \cdots .$$

However from the conditional measures π and μ defined by the agent and the environment we see that $\tau(\underline{y}_1) := \pi(\underline{y}_1)$, $\tau(y_1\underline{x}_1) := \mu(y_1\underline{x}_1)$, $\tau(y_1x_1\underline{y}_2) := \pi(y_1x_1\underline{y}_2)$ and so on. Thus we can define,

$$\tau(\underline{y}_1\underline{x}_1\underline{y}_2\underline{x}_2 \ldots) := \ \pi(\underline{y}_1) \cdot \mu(y_1\underline{x}_1) \cdot \pi(y_1x_1\underline{y}_2) \cdot \mu(y_1x_1y_2\underline{x}_2) \cdots .$$

This relation is especially useful when dealing with ergodicity and other environmental properties which depend on there existing a policy under which the agent–environment interaction behaves in a certain way.

2 Environmental Classes

2.1 Elementary Environmental Classes

Many kinds of environments are possible within the agent-environment reinforcement learning framework. In this section we will define and categorise the most fundamental classes of environments from an abstract mathematical perspective. In particular we will be interested in how various dependencies on the prior history of agent-environment interaction give rise to a multitude of different environmental classes.

2.1.1 Passive Environments

The first group of environmental classes that we will consider are the passive environments. Loosely speaking such environments are not affected by the agent's actions. This of course severely limits their power and as such we will usually be more interested in the active environments to be described later. Nevertheless there are some examples where passive environments are important, in particular sequence prediction is a significant passive environment.

2.1 Definition. An **Independent and Identically Distributed Process** or **i.i.d. Process** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(yx_{<k}y_k\underline{x}_k) = \mu(\underline{x}_k).$$

Other than perhaps a constant environment, i.i.d. processes are about the simplest environments possible. The history of previous perceptions and actions has no effect at all on future perceptions. Because the agent can't in any way influence its future rewards this environment is, from the perspective of artificial intelligence, rather uninteresting.

It is worth noting that the above definition, like many others that are to follow, involves some notational shorthand as described in Appendix A.3. Essentially we are defining a new function (on the right hand side) over a reduced parameter space by dropping any parameters that have no effect on the original function's value. In this case it indicates that the function μ is completely independent of its history $yx_{<k}y_k$. It also indicates that the distribution over $\underline{x}_k$ is completely independent of the length of the history.

2.2 Example. Many very simple situations can be described as i.i.d. processes. Imagine a game where a 6 sided die is thrown repetitively. We, as the agent, get a reward of 1 whenever a 6 is thrown and 0 otherwise. There are no actions that we can take and the die is simply thrown without any control from us. Formally: the action space $\mathcal{Y} := \{\epsilon\}$, the observation space $\mathcal{O} := \{1, 2, 3, 4, 5, 6\}$ and the reward space $\mathcal{R} := \mathbb{B}$. The environment's chronological system is defined $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$,

$$\mu(yx_{<k}y_k\underline{x}_k) := \begin{cases} \frac{1}{6} & \text{for } x_k = r_k o_k \in \{01, 02, 03, 04, 05, 16\}, \\ 0 & \text{otherwise.} \end{cases}$$

If we allow the next perception x_k to depend on the previous observation o_{k-1} we get a more complex environment:

2.3 Definition. A **Markov Process** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(yx_{<k}y_k\underline{x}_k) = \mu(o_{k-1}\underline{x}_k).$$

In both i.i.d. processes and Markov processes the agent is totally passive in the sense that the agent's actions $y_1, y_2, \ldots, y_k$ have no effect on future perceptions including rewards. For this reason they often do not have reward functions associated with them. (See [4] for an comprehensive analysis of Markov processes.)

2.4 Example. Imagine a game where we have a ring shaped playing board that has been divided into 20 different cells. We have a small pebble that starts in cell one that moves around the board as follows: On each turn a standard 6 sided die is thrown to decide how many positions our pebble will be moved clockwise around the board. In cells 5 and 15 we receive a reward of 1 and we receive 0 reward otherwise. We can model this system as a Markov process as follows: define the action space $\mathcal{Y} := \{\epsilon\}$, the observation space $\mathcal{O} := \{0, 1, 2, \ldots, 19\}$ and the reward space $\mathcal{R} := \mathbb{B}$. The environment's chronological system is a Markov process defined as $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$,

$$\mu(yx_{<k}y_k\underline{x}_k) := \begin{cases} \frac{1}{6} & \text{for } o_k \in \{(o_{k-1}+1) \bmod 20, \ldots, (o_{k-1}+6) \bmod 20\} \\ & \text{and } r_k = \delta_{k\,5} + \delta_{k\,15} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly in this game we have a $\frac{1}{6}$ chance of obtaining reward if our pebble is currently in one of the six cells before either cell 5 or cell 15.

We can generalise the previous totally passive environmental classes as follows:

2.5 Definition. A **Totally Passive Environment** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(yx_{<k}y_k\underline{x}_k) = \mu(x_{<k}\underline{x}_k).$$

We can see that the agent's actions have no effect at all, but otherwise the system is completely unconstrained. This class of environments is clearly a superset of all the environments we have defined thus far. Again we have again abused our notation slightly in this definition. This time we have dropped not only the old action history $y_{<k}$ but also the last action y_k as these do not affect the value of the chronological function.

We could attempt to define a more limited version of this class where the next state depends on the last n states rather than the full history. However in this case the system can be reduced to the first order case of a standard Markov process by a simple transformation where for of each of the histories of length n we create a unique state. Also this definition only makes sense when the perception space is finite. If we allow $|\mathcal{X}| = \infty$ we can always reduce any arbitrary process back to being a Markov process through a transformation where we map every possible finite history to a unique state.

While totally passive environments are useful in modelling some systems, in terms of the AI problem they are uninteresting because the agent plays no role at all as it cannot affect its environment. We can relax these constraints just a little by allowing the environment to send reward signals to the agent that depend on the agent's actions. This gives us our first class of environments that is genuinely interesting from the perspective of artificial intelligence.

2.6 Definition. A **Passive Environment** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\forall k \in \mathbb{N}$ and $\forall yx_{<k}y_k o_k \in (\mathcal{Y} \times \mathcal{X})^{k-1} \times \mathcal{Y} \times \mathcal{O}$ we have

$$\mu(yx_{<k}y_k\underline{o}_k) = \mu(x_{<k}\underline{o}_k).$$

Here we have restricted the observation to depend on only the past perceptions; the reward however is now no longer constrained. Thus a passive environment is allowed to react to the agent's actions but only in the reward signal. The non-reward part of the signal, the observation, is not affected at all. Thus the environment is in some senses passive but it is still able to teach the agent through reward signals that depend on the agent's actions. The problem of sequence prediction, which will be defined later, is one important example of this.

2.1.2 Bandits

The simplest fully active environment possible is where the next perception x_k depends on only the last action y_k:

2.7 Definition. A **Bandit** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(yx_{<k}y_k\underline{x}_k) = \mu(y_k\underline{x}_k).$$

In one respect bandits are weaker than Markov processes because they do not have the ability to remember what happened in the past. They do however ever have the ability to react to the last action generated by the agent and so they are in this respect more powerful than Markov processes.

2.8 Example. The name *bandit* comes from the bandit machines found in casinos around the world. Imagine a machine that had n different levers or arms that we can pull. Each arm has a different but fixed probability of generating a reward. For our example imagine that the reward is simply either a 0 or a 1, of course with a real bandit machine in a casino the reward is monetary. Clearly we would like to know which arm to pull so that we are most likely to get a reward, but we are not told this information. One approach might be to spend some time pulling different arms and watching to see which arms seems to be most likely to give us reward.

We can formally define such a bandit as follows: Let $\mathcal{O} := \{\epsilon\}$, $\mathcal{R} := \mathbb{B}$ and $\mathcal{Y} := \{1, 2, 3, \ldots, n\}$ represent the n different arms that the agent can pull. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be the respective probabilities of obtaining a reward of 1 after pulling each arm. Let

$$\mu(y_k\underline{r}_k) := \begin{cases} \alpha_{y_k} & \text{for } r_k = 1, \\ 1 - \alpha_{y_k} & \text{otherwise.} \end{cases}$$

We can see that the reward r_k is solely determined by the last action y_k. Thus in a bandit environment an agent only has to learn which action produces the highest expected return. Conceptually at least the problem is quite simple in this situation, however for complex distributions over large action spaces even a bandit problem can be very difficult to solve in practise.

2.1.3 Markov Decision Processes

If we allow the next perception to depend on both the last perception and the last action then we get a much more powerful system called a Markov decision process or MDP for short. MDPs generalise i.i.d. Processes, Bandits and Markov processes.

2.9 Definition. A (stationary) **Markov Decision Process** or **MDP**[1] is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(yx_{<k}y_k\underline{x}_k) = \mu(o_{k-1}y_k\underline{x}_k).$$

In the context of MDPs the observations are usually called *states* as they fully reflect the current state of the environment. In more general contexts however this is not the case, or at least the word is potentially misleading, and so we will continue to call them observations.

The form of our MDP is what is known as a *stationary* MDP. This is because the probability of a perception given the action and the last perception is fixed, or stationary, over time. That is, the distribution is independent of k. In some more general contexts μ is allowed to vary over time. This makes the system considerably more difficult to work with. These non-stationary MDPs can be modelled as POMDPs which we will define later.

For many results in the literature, in particular in reinforcement learning, the action and perception spaces are finite. This makes them much easier to work with. We will call these *finite Markov decision processes*.

2.10 Example. Consider again the simple Markov process in Example 2.3. This can be simply extended to be an MDP by allowing the agent to decide whether to move clockwise or counter-clockwise around the board. In this case $\mathcal{O} := \{0, 1, \ldots, 19\}$ and $\mathcal{R} := \mathbb{B}$ as before, but now $\mathcal{Y} := \{\circlearrowright, \circlearrowleft\}$ as the agent selects which direction to move in.

The environment's chronological system μ can now be defined as $\forall k \in \mathbb{N}$ and $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$,

$$\mu(yx_{<k}y_k\underline{x}_k) := \begin{cases} \frac{1}{6} & \text{for } o_k \in \{(o_{k-1}+1) \bmod 20, \ldots, (o_{k-1}+6) \bmod 20\} \\ & \wedge r_k = \delta_{k\,5} + \delta_{k\,15} \wedge y_k = \circlearrowright \\ \frac{1}{6} & \text{for } o_k \in \{(o_{k-1}-1) \bmod 20, \ldots, (o_{k-1}-6) \bmod 20\} \\ & \wedge r_k = \delta_{k\,5} + \delta_{k\,15} \wedge y_k = \circlearrowleft \\ 0 & \text{otherwise.} \end{cases}$$

The optimal policy in this situation is now more complex because in order to maximise future rewards the agent must take into account not only which direction is most likely to

[1] Our definition of an MDP is a little different to the standard formal definition as given by Bellman [1] but it is equivalent. For more on this see Appendix B.

generate reward in the next cycle, but also how its decision may affect its ability to receive rewards in cycles further into the future. This is a major issue in the theory of making optimal decisions.

A natural way to generalise the class of MDPs is to allow the next perception to depend on the last n perceptions and actions:

2.11 Definition. A (stationary) $\boldsymbol{n^{th}}$ **order Markov Decision Process** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\forall k \in \mathbb{N}$ and $\forall \underline{yx}_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(\underline{yx}_{<k} y_k \underline{x}_k) = \mu(x_{k-m} \underline{yx}_{k-m+1:k-1} y_k \underline{x}_k).$$

where $m := \min\{n, k\}$.

Immediately from the definition we can see that a standard MDP is an n^{th} order MDP where $n = 1$. The added complication of the variable m is to allow for the situation where the current history length is less than n.

2.1.4 Partially Observable Markov Decision Processes

In more realistic situations the agent isn't always able to observe the full state of the environment that is has to deal with. Indeed often an agent can only observe a very small part of the environment's state. Taking this into account gives us an important generalisation of the MDP class of environments.

2.12 Definition. A **Partially Observable Markov Decision Process** or **POMDP** is environment $(\mathcal{Y}, \mathcal{X}, \mu)$ defined as follows. Let $(\tilde{\mathcal{Y}}, \tilde{\mathcal{X}}, \tilde{\mu})$ be a stationary MDP environment called the **core MDP**. Let $\tilde{\mathcal{Y}} = \mathcal{Y}$ and $\phi : \tilde{\mathcal{X}} \times \mathcal{X} \to [0, 1]$ be a conditional distribution of the form $\phi(\tilde{x}\underline{x})$ which expresses the probability of perceiving an x when the core MDP output an $\tilde{x}$. Define $\forall \underline{yx}_{1:n} \in (\mathcal{Y} \times \mathcal{X})^n$,

$$\mu(\underline{yx}_{1:n}) := \sum_{\tilde{x}_0, \ldots, \tilde{x}_n \in \tilde{\mathcal{X}}} \tilde{\mu}(y_1 \tilde{\underline{x}}_1) \cdot \phi(\tilde{x}_1 \underline{x}_1) \cdots \tilde{\mu}(y_n \tilde{\underline{x}}_n) \cdot \phi(\tilde{x}_n \underline{x}_n).$$

Both theoretically and practically this class of environments can be very difficult to work with.

2.2 Special Environmental Properties

Often a chronological environment can have certain algebraic or other structural properties. These can make working with the environment much easier.

2.2.1 Unobservable Environments

2.13 Definition. A chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ is said to be **unobservable** if $\mathcal{O} = \{\epsilon\}$.

An unobservable environment still may send reward signals to the agent and thus may still be quite a rich environment if the reward signals contain a lot of information.

2.2.2 Ergodic Environments

There are many definitions of *ergodicity* (see for example [8]) and the word has different meanings in different contexts. The following simple definition is relevant to our work and comes from [6]:

2.14 Definition. A chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ is **observation ergodic** if there exists an agent $(\mathcal{X}, \mathcal{Y}, \pi)$ such that under policy π every possible observation $o \in \mathcal{O}$ occurs infinitely often with probability 1.

The above definition only really makes sense for MDPs where the last perception represents the complete state of the environment. For more general environments we need a stronger form of ergodicity. The most comprehensive way to compare the current state of two systems with different histories is to compare their distributions over possible futures. If they are the same then in a very strong sense the two systems are equivalent from the perspective of the agent. The fact that the two environments may have different histories is not important. We can use this idea to define a stronger form of ergodicity as follows.

2.15 Definition. A chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ is said to be **strongly ergodic** if $\forall yx_{<i}$ and $\forall \tilde{y}\tilde{x}_{<j}$ there exists a policy π such that if the agent follows this policy from time j onward then with probability arbitrarily close to 1 some finite sequence $\tilde{y}\tilde{x}_{j:j+k}$ will be generated after which $\forall \overline{yx}_{<n}$

$$\mu(yx_{<i}\underline{\overline{yx}}_{<n}) = \mu(\tilde{y}\tilde{x}_{<j}\tilde{y}\tilde{x}_{j:j+k}\underline{\overline{yx}}_{<n}).$$

In simple terms this means that no matter what history an environment has, the agent can always follow some policy that changes the environment in such a way that the environment becomes as if it has some other history of the agent's choosing. This is of course a very powerful and useful condition. Intuitively it means that an agent can always recover from any mistake it makes. It can be shown that strong ergodicity implies ergodicity and in the case of MDPs and in weaker environments the two are equivalent.

2.2.3 Factorisable Environments

Another related concept is that of factorisability.

2.16 Definition. A chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ is **Factorisable** if we can break the cycles up into groups of independent episodes $e = 1, 2, 3, \ldots$ where each episode e consists of the cycles $k = n_r + 1, \ldots, n_{r+1}$ for some $0 = n_0 < n_1 < \cdots < n_s = n$ such that,

$$\mu(\underline{yx}_{1:n}) = \prod_{e=0}^{s-1} \mu_e(\underline{yx}_{n_r+1:n_{r+1}}).$$

2.3 Environmental Problem Classes

Many types of AI problems can be expressed as particular kinds of chronological environments. Here we will formalise some of these in this framework.

2.3.1 Sequence Prediction

2.17 Definition. A **Sequence Prediction Problem** is a passive environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that $\mathcal{Y} = \mathcal{O}$ and $\forall k \in \mathbb{N}$, $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$:

$$\mu(yx_{<k}y_k\underline{x}_k) > 0 \Rightarrow r_k = 1 - \ell(y_k, o_k)$$

where $\ell : \mathcal{Y} \times \mathcal{O} \to [0,1] \cap \mathbb{Q}$ is an arbitrary loss function.

Sequence prediction environments are limited in the sense that future observations (the non-reward part) do not depend on the agent's actions. However they are also quite powerful in the sense that there is no limit on how long the relevant history might be. One simple example of a loss function is $\ell(y, o) := 1 - \delta_{yo}$, that is, the loss is 1 if the predicted symbol is different to the actual symbol and 0 otherwise. The reason for using $\mathbb{Q}$ is that we need to ensure that $\text{ran}(\ell)$ is countable so that $\mathcal{R}$ remains countable and r_k can be encoded in a finite string.

2.18 Example. Let $\mathcal{Y} = \mathcal{O} := \{0, 1, \ldots, 9\}$ and $\ell(y, o) := \frac{1}{9}|y - o|$. Let o_k be the k^{th} digit of the number π, that is, $o_1o_2o_3o_4 \ldots := 31415 \ldots$. Thus in order to minimise loss so as to maximise reward the agent must generate the successive digits of the number π. A correct digit gets a reward of 1 while an incorrect digit gets a lesser reward proportional to the difference between the correct digit and the guess.

2.3.2 Function Minimisation

There are a lot of possible interpretations of the function minimisation problem (see for example page 613 of [6]). This is only one possibility that should work well with the concept of self-optimisingness that we will define and also covers some of the other possible interpretations of the problem.

2.19 Definition. A **Function Minimisation Problem** is an unobservable Bandit environment $(\mathcal{Y}, \mathcal{X}, \mu)$ such that for some $f : \mathcal{Y} \to [0,1] \cap \mathbb{Q}$ we have $\forall yx \in \mathcal{Y} \times \mathcal{X}$

$$\mu(y\underline{x}) > 0 \Rightarrow r = 1 - f(y).$$

The agent faces a problem where its actions are interpreted as the input to some function and in each cycle the result of the function is turned into a reward value representing the optimality of the value of the function at that point. Clearly the agent needs to find an action which corresponds to a point in the function's domain where the function has a low value. Finding such values will allow the agent to attain a high average reward in the long run. However in order to find such points the agent may have to test numerious non-optimal points as it searches for good values. During this exploration time the agent will most likely examine points that are actually worse than other points that it already knows about. While this represents a short term opportunity cost in the sense that the agent isn't fully exploiting the good points that it already knows about, there is a probability that this exploration will lead to new and even better points which will, in the long run, outweigh the cost of exploration.

The exploration versus exploitation trade off is a reoccurring theme for agent's facing difficult problems in active environments (see for example [10] or [2] for more on this). In passive environments like sequence prediction the agent cannot actively explore the environment and so this problem does not occur.

2.20 Example. Let $\mathcal{Y} := [0,1] \cap \mathbb{Q}$, $\mathcal{R} := [0,1] \cap \mathbb{Q}$ and $f(y) := (y - \frac{1}{4})^2$. Thus in order to maximise reward the agent must find the minimum of the quadratic function f at $y = \frac{1}{4}$ and generating this value as its action.

2.3.3 Repeated Strategic Games

As the name suggests, in a repeated strategic game environment the agent faces some game repetitively. Many games such as chess, various card games, tic-tac-toe and others can be modelled in this way if at the end of each match a new match is started. These environments have a weak form of ergodicity as any mistakes that the agent makes only effect the environment during the current match and not in the following matches. In the following definition the length of one match, or *episode* as we call it, is fixed. If we want to allow games to finish before the episode finishes we can pad the remaining cycles and perhaps also reward the system for padded cycles following a victory.

2.21 Definition. A **Repeated Strategic Game** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ where $\exists l \in \mathbb{N}$ such that $\forall k \in \mathbb{N}, \forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(yx_{1:k}) = \mu(yx_{lm:k}) \prod_{i=1}^{m-1} \mu(yx_{l(i-1):li-1})$$

or equivalently,

$$\mu(yx_{<k}y_k x_k) = \mu(yx_{lm:k-1}y_k x_k)$$

where $m := \lfloor k/l \rfloor$ is the episode number in cycle k.

Clearly i.i.d. processes and bandits are repeated strategic games.

One thing to keep in mind is that in our framework, compared to standard game theory, the environment is both the opponent and also provides the reward signal. This isn't a problem because we can always wrap an opponent and goal up together to make a chronological environment.

.

There are many examples of repeated strategic games. The two coins game described in Examples 1.1 and 1.8 is a simple strategic game with an episode length of 1. Indeed any game of fixed length can be simply repeated to define a strategic game.

2.3.4 Classification

With a classification problem there is a domain space $\mathcal{W}$ and a set of classes $\mathcal{Z}$ and some function $f : \mathcal{W} \to \mathcal{Z}$. The agent must try to learn this mapping based on examples and then when given cases where the class is missing it has to guess at the correct class in order to obtain rewards.

2.22 Definition. A **Classification Problem** is a chronological environment $(\mathcal{Y}, \mathcal{X}, \mu)$ set up as follows. Let $\mathcal{W}$ and $\mathcal{Z}$ be two countable sets called the **attribute space** and the **class space** respectively. $\mathcal{Z}$ includes a special symbol "?" used to indicate that the agent needs to guess the class. Let $\mathcal{O} := \mathcal{W} \times \mathcal{Z}$ and let $f : \mathcal{W} \to \mathcal{Z} \setminus \{\text{"?"}\}$ where $\forall k \in \mathbb{N}$, $\forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ where $x_{k-1} = r_{k-1}w_{k-1}z_{k-1} \in \mathcal{R} \times \mathcal{W} \times \mathcal{Z}$, and $x_k = r_k w_k z_k \in \mathcal{R} \times \mathcal{W} \times \mathcal{Z}$, $\mu(yx_{<k}y_k\underline{x}_k) > 0$ implies

$$\mu(yx_{<k}y_k\underline{w}_k) = \mu(\underline{w}_k)$$

and

$$z_k \in \{f(w_k) \cup \text{"?"}\}$$

and

$$r_k = \begin{cases} 1 & \text{if } z_{k-1} = \text{"?"} \wedge y_k = f(w_{k-1}) \\ 0 & \text{otherwise.} \end{cases}$$

The first condition says that the distribution of the points in the attribute space is independent of the system's history. In other words, this part of x_k is i.i.d.. The second condition says that in each cycle the system must either provide a training instance or ask for the agent to classify based on the attribute vector. The third condition says that reward is given when the system asks for a classification and the agent gets it correct.

It is not difficult to see that this system is both a passive environment and an MDP environment.

3 Relations between the Environmental Classes

Many of the environmental classes defined in the previous section are related to each other. The relation between the totally passive, passive and prediction environments is simple and has already been noted. In this section we investigate these relations between the active environmental classes. These relations will be useful later on because anything that we prove about a class of environments will of course also be true for those classes which are subset of the class.

Figure 2 summarises the main results in advance. The most general and powerful class of environments (chrolonogical environments) at the top and the most limited and specific classes at the bottom. As is clear from the diagram, Ergodic MDPs play a central role in this analysis. In particular Ergodic MDPs have the useful property that they admit self-optimising policies.

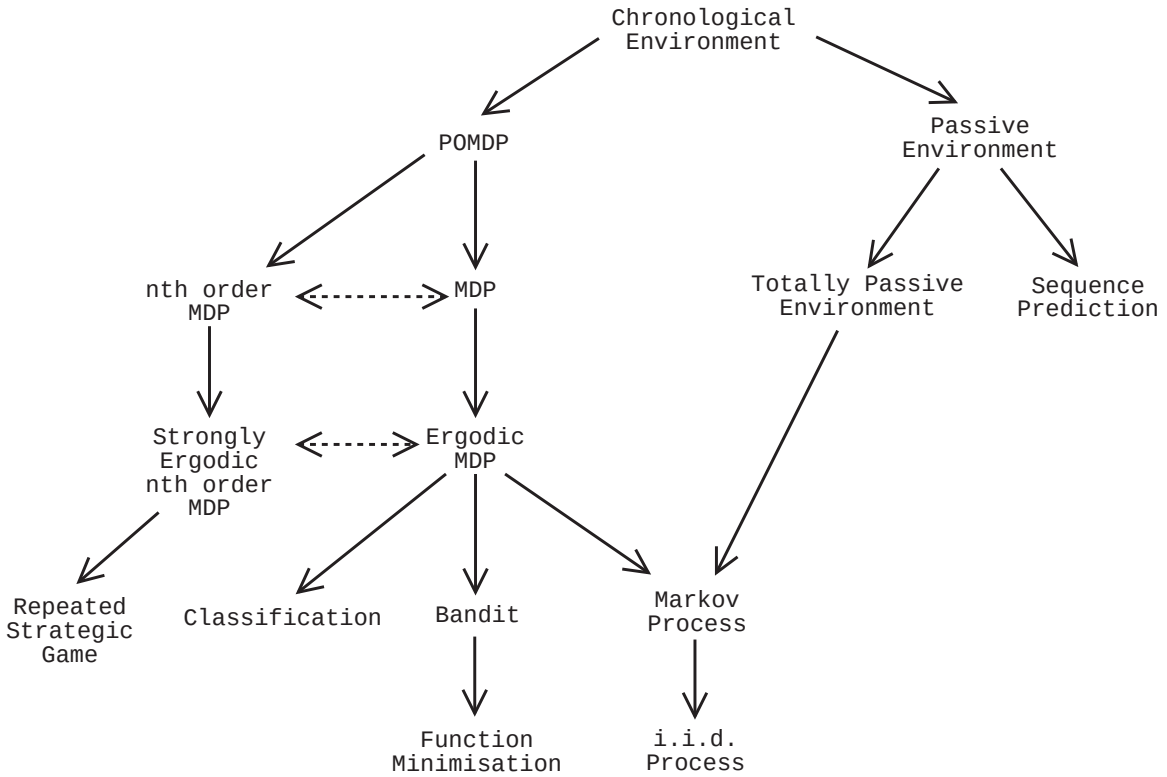


Figure 2: Relations between environment types from the most general at the top to the most specific at the bottom. The dotted horizontal lines indicate that two environmental classes are reducible to each other.

3.1 MDPs and n^{th} order MDPs

It is immediate that a first order MDP is an n^{th} order MDP where $n = 1$. It also appears that n^{th} order MDPs are more general than standard MDPs, however this is only true in a weak

sense as the follow result shows: any n^{th} order MDP can be converted into a standard MDP by extending the action and perception spaces and appropriately extending the measure. The proof follows the same pattern as the reduction of n^{th} order Markov processes to the standard Markov process except that now we have to deal with the complication of having actions in the history as well.

3.1 Lemma. *n^{th} order MDPs can be reduced to MDPs.*

Proof. Consider an n^{th} order MDP $(\mathcal{Y}, \mathcal{X}, \mu)$ and define the set,

$$\mathcal{Z} := \bigcup_{i=1}^{n} \mathcal{X} \times (\mathcal{Y} \times \mathcal{X})^{i-1}.$$

Obviously there is a bijective correspondence between elements of $\mathcal{Z}$ and histories with length of no more than n. Though it complicates things we need to have histories of lengths less than n to accommodate the first n cycles of the system. From the definition of $\mathcal{Z}$ and the fact that $\mathcal{X}$ and $\mathcal{Y}$ are both recursive, $\mathcal{Z}$ must be recursively enumerable as required by the definition of a chronological environment.

We need to define a first order MDP over the space $\mathcal{Z}$ which is equivalent to our n^{th} order MDP in order to prove the result.

Define a chronological system $\tilde{\mu}$ as follows: $\forall i \in \{1, \ldots, n\}$ and $\forall z := x_{k-i} y x_{k-i+1:k-1} \in \mathcal{X} \times (\mathcal{Y} \times \mathcal{X})^{i-1}$ and $\forall j \in \{1, \ldots, n\}$ and $\forall z' := x'_{k-j+1} y' x'_{k-j+2:k} \in \mathcal{X} \times (\mathcal{Y} \times \mathcal{X})^{j-1}$ define,

$$\tilde{\mu}(z y_k \underline{z}') := \begin{cases} \mu(x_{k-i} y x_{k-i+1:k-1} y'_k \underline{x}'_k) & \text{if } x_{k-i+1} y x_{k-i+2:k-1} = \\ & \qquad x'_{k-j+1} y' x'_{k-j+2:k-1}, \\ 0 & \text{otherwise.} \end{cases}$$

From this definition $\tilde{\mu}$ is defined for all $z, z' \in \mathcal{Z}$ and $y_k \in \mathcal{Y}$. In particular, for any transition z to z' which is impossible, because the two histories under the original n^{th} order MDP would be inconsistent, are given a transition probability of zero under $\tilde{\mu}$.

It remains to be shown that $\tilde{\mu}$ has the required properties of a chronological system. Firstly $\tilde{\mu}$ is trivially positive from the definition. Also

$$\sum_{z' \in \mathcal{Z}} \tilde{\mu}(z y_k \underline{z}') = \sum_{x'_k \in \mathcal{X}} \mu(x_{k-i} y x_{k-i+1:k-1} y_k \underline{x}'_k) = 1.$$

The first equality follows from the definition of $\tilde{\mu}$ and the fact that we can ignore any zeroed elements in the sum. Also for these non-zero terms in the sum it is guaranteed that $y'_k = y_k$. The second equality is because μ is a chronological system. Thus $(\mathcal{Y}, \mathcal{Z}, \tilde{\mu})$ is a first order MDP.

Finally we can drop any unused states from $\mathcal{Z}$ so that this MDP also meets our assumption that our chronological systems are accessible in the sense that the action and perception spaces do not contain elements that can never possibly occur. □

It may appear that $\tilde{\mu}$ in the above proof contains some information about the probability of the actions $y'_{k-n+2:k}$ as z' contains these elements of $\mathcal{Y}$. However $\tilde{\mu}$, like μ, is not in fact

a proper measure; it is only a chronological system and thus it is only ever conditionally defined. In the case of $\tilde{\mu}(zy_k\underline{z}')$, all the actions $y'_{k-n+2:k}$ in z' are already conditioned by $z = x_{k-n}yx_{k-n+1:k-1}$ and y_k. Thus the function $\tilde{\mu}$ never contains any information about the probabilities of actions as all of the actions in z' have already had their values set by the conditional parameters.

While an n^{th} order MDP is reducible to a first order MDP, often it's more natural to model a specific problem using an n^{th} order MDP rather than by considering its equivalent MDP representation.

3.2 Ergodic MDPs

It turns out that many classes of environments are in fact subsets of the more general class of ergodic MDPs. This will be convenient later on as ergodic MDPs have many important properties. In this section we will look at which environmental classes can be reduced to ergodic MDPs and under what conditions.

The first two cases considered are trivial and are quickly dealt with as the policy can in no way affect the system. Thus ergodicity follows directly from the definitions.

3.2 Theorem. *i.i.d. processes are ergodic MDPs.*

Proof. Consider an i.i.d. process $(\mathcal{Y}, \mathcal{X}, \mu)$. From the definition of an i.i.d. process we immediately see that they are a special case of the MDP definition and thus are trivially MDPs.

From the accessible property (definition 1.6) we see that $\forall x \in \mathcal{X}, \exists k \in \mathbb{N}, \exists yx_{<k}y_k \in (\mathcal{Y}\times\mathcal{X})^{k-1}\times\mathcal{Y} : \mu(yx_{<k}y_k\underline{x}) > 0$. Applying the definition of an i.i.d. process this immediately reduces to just $\forall x \in \mathcal{X} : \mu(\underline{x}) > 0$ giving ergodicity independent of the policy. □

Similarly for Markov processes,

3.3 Theorem. *Markov processes are ergodic MDPs.*

Proof. Consider a Markov process $(\mathcal{Y}, \mathcal{X}, \mu)$. From the definition of a Markov process we immediately see that they are a special case of the MDP definition and thus are trivially MDPs.

From the accessible property we see that $\forall x \in \mathcal{X}, \exists k \in \mathbb{N}, \exists yx_{<k}y_k \in (\mathcal{Y} \times \mathcal{X})^{k-1} \times \mathcal{Y} : \mu(yx_{<k}y_k\underline{x}) > 0$. Applying the definition of a Markov process this immediately reduces to just $\forall x \in \mathcal{X}, \forall x_k \in \mathcal{X} : \mu(x_k\underline{x}) > 0$. This implies that $\forall x_k : \mu(\underline{x}) > 0$ giving us ergodicity independent of the policy. □

In the case of Bandits things are a little more interesting as we now need to find a policy that demonstrates ergodicity.

3.4 Theorem. *Bandits are ergodic MDPs.*

Proof. Consider a Bandit $(\mathcal{Y}, \mathcal{X}, \mu)$. From the definition of a Markov process we immediately see that they are a special case of the MDP definition and thus are trivially MDPs.

From the accessible property we see that $\forall x \in \mathcal{X}, \exists k \in \mathbb{N}, \exists yx_{<k}y_k \in (\mathcal{Y} \times \mathcal{X})^{k-1} \times \mathcal{Y} : \mu(yx_{<k}y_k\underline{x}) > 0$. Applying the definition of a Bandit this immediately reduces to,

$$\forall x \in \mathcal{X}, \exists y_k \in \mathcal{X} : \mu(y_k\underline{x}) > 0. \tag{1}$$

Next we need to show that there exists a policy under which ergodicity holds. Firstly consider the case where $|\mathcal{Y}| < \infty$. Define a policy $\forall k \in \mathbb{N}, \forall yx_{1:k}y \in (\mathcal{Y}\times\mathcal{X})^k\times\mathcal{Y} : \pi(yx_{1:k}\underline{y}) := \frac{1}{|\mathcal{Y}|}$. As the history is arbitrary, $\forall y \in \mathcal{Y} : \pi(\underline{y}) := \frac{1}{|\mathcal{Y}|}$. As this is strictly non-zero we can simply drop y_k from equation 1 to get $\forall x \in \mathcal{X} : \mu(\underline{x}) > 0$ giving ergodicity under π

Now consider the case where $|\mathcal{Y}| = \infty$. Let $y^{(1)}, y^{(2)}, \ldots$ be the standard recursive enumeration of $\mathcal{Y}$. Now define $\forall k \in \mathbb{N}, \forall yx_{1:k}y^{(i)} \in (\mathcal{Y} \times \mathcal{X})^k \times \mathcal{Y} : \pi\left(yx_{1:k}\underline{y}^{(i)}\right) := 2^{-i}$. Once again we can drop the dependence on the arbitrary history giving $\forall y^{(i)} \in \mathcal{Y} : \pi\left(\underline{y}^{(i)}\right) := 2^{-i}$. As this is strictly non-zero for each action we can again combine with equation 1 to obtain the desired result. □

Note that in both cases in the above proof the policy π is recursive as the enumeration on $\mathcal{Y}$ is recursive as well as the defined conditional distribution. Also, neither of the policies require any knowledge of μ. A non-constructive proof would have been simpler as we could then have just dropped the dependency on y_k in equation 1 immediately.

However, as noted previously, our standard definition of ergodicity really only makes sense for MDPs. This is because with an MDP the last perception fully represents the state of the system. For n^{th} order MDPs we require the more powerful notion of strong ergodicity.

3.5 Theorem. *Strongly ergodic n^{th} order MDPs are equivalent to ergodic MDPs.*

Proof. Let $(\mathcal{Y}, \mathcal{X}, T, \mu)$ be an n^{th} order MDP which is strongly ergodic. Following the construction in lemma 3.1 let $(\mathcal{Y}, \mathcal{Z}, \tilde{R}, \tilde{\mu})$ be its equivalent first order MDP. We need to show that $(\mathcal{Y}, \mathcal{Z}, \tilde{R}, \tilde{\mu})$ is ergodic, that is, there exists a policy π under which all $z \in \mathcal{Z}$ are visited infinitely often with probability 1.

From strong ergodicity we know that for any history $\tilde{y}\tilde{x}_{<j}$ there exists a policy $\tilde{\pi}$ such if the agent follows this policy then with probability 1 some finite sequence $\exists \tilde{y}\tilde{x}_{j:j+k}$ will be generated after which for all $yx_{<n}$ we have $\mu(y\underline{x}_{<n}) = \mu(\tilde{y}\tilde{x}_{<j}\tilde{y}\tilde{x}_{j:j+k}y\underline{x}_{<n})$.

Let $z^{(1)}, z^{(2)}, \ldots$ be some enumeration of $\mathcal{Z}$. As the MDP is non-reducible we have $\forall i \in \mathbb{N}$, there exists a sequence $k \in \mathbb{N}$, $yx_{<k}y_k \in (\mathcal{Y} \times \mathcal{Z})^{k-1} \times \mathcal{Y}$ such that $\tilde{\mu}(yx_{<k}y_k\underline{z}^{(i)}) > 0$.

We can simply combine the above two properties in a simple algorithm that doves tails through the space $\mathcal{Z}$:

1. let $n := 1$
2. let $i := 1$
3. use the strong ergodicity property to reset the system back to its original state with arbitrarily probability 1 by following the appropriate policy
4. use the accessible property to follow a finite path that leads to perception $z^{(i)}$ with some non-zero probability

5. let $i := i + 1$
6. if $i < n$ goto step 2
7. let $n := n + 1$
8. go to step 1

On each trial we use strong ergodicity to allow us to reset the system back to its initial state (with probably 1) and then attempt to reach the state $z^{(i)}$. We can reach $z^{(i)}$ with a non-zero probability due to the accessible property. The algorithm dove tails through the space $\mathcal{Z}$ attempting to reach each $z^{(i)}$ infinitely often. As the probability of success is non-zero on each trial and each $z^{(i)}$ is attempted infinitely often we will reach each perception $z^{(i)}$ infinitely often with probably 1. That is, ergodicity holds. □

Now we move on to considering some of the special problem classes. In some cases the previous results make this trivial.

3.6 Theorem. *Function minimisation problems are ergodic MDPs.*

Proof. From the definition of a function minimisation problem we see that they are Bandits. Thus the result follows from theorem 3.4. □

3.7 Theorem. *Classification problems are ergodic MDPs.*

Proof. From definition 2.22 of a classification problem we see that $x_k := r_k w_k z_k$. From the first condition in the definition we see that w_k is independent, from the second condition we see that z_k depends on only w_k, and in the third condition we see that r_k depends on only z_{k-1}, y_k and w_{k-1}. Thus x_k depends on only y_k and x_{k-1}, which makes μ an MDP. From the definition for a classification problem we see that we can factorise μ as follows: $yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$,

$$\begin{aligned} \mu(yx_k y_k \underline{x}_k) &= \mu(w_{k-1} z_{k-1} y_k \underline{r}_k \underline{w}_k \underline{z}_k) \\ &= \mu(w_{k-1} z_{k-1} y_k \underline{r}_k) \cdot \tilde{\mu}(\underline{w}_k \underline{z}_k). \end{aligned}$$

To show ergodicity we need to show that there exists a policy under which every state $x := rwz \in \mathcal{X}$ is visited infinitely often with probability 1. From the definition we see that $\forall w_k \in \mathcal{W}$ and corresponding $z = f(w_k)$ we have $\tilde{\mu}(\underline{w}_k \underline{z}_k) > 0$. As this is also independent, it only remains to be shown that there exists a policy under which every $r \in \mathcal{R}$ is attained infinitely often with probability 1.

This is clearly true under a policy which chooses y_k at random as each possible $w_{k-1} z_{k-1}$ combination occurs with probability 1 and r_k depends on only $w_{k-1} z_{k-1}$ and y_k. □

3.8 Theorem. *Repeated Strategic Games are reducible to ergodic MDPs*

Proof. Let $(\mathcal{Y}, \mathcal{X}, \mu)$ be a repeated strategic game with episode length $l \in \mathbb{N}$. From the definition of a repeated strategic game we see that $\forall k \in \mathbb{N}, \forall yx_{1:k} \in (\mathcal{Y} \times \mathcal{X})^k$ we have

$$\mu(yx_{<k}y_k\underline{x}_k) = \mu(yx_{lm:k-1}y_k\underline{x}_k)$$

where $m := \lfloor k/l \rfloor$ is the episode number in cycle k. Thus we can view μ as a (stationary) l^{th} order MDP which has the special property that it resets back to the start state with no history every l cycles. By lemma 3.1 let $(\mathcal{Y}, \mathcal{Z}, \tilde{R}, \tilde{\mu})$ be an equivalent first order MDP. Unlike in lemma 3.1 we don't just use the states which contain less than l original states just at the start when the history length is less than l, here we use them again and again where the number of cycles so far in the current episode is less than l.

By the accessible property, $\forall x' \in \mathcal{X}$, $\exists k \in \mathbb{N}$, $\exists yx_{<k} \in (\mathcal{Y} \times \mathcal{X})^k$ such that $\mu(yx_{<k}y_k\underline{x}'_k) > 0$. Thus, as μ repeats infinitely often we can always reach any state x' infinitely often with probability 1 by following the required history above repetitively. That is, the environment is ergodic. Moreover, using the same technique we can also reach any possible sequence of states of length less that l infinitely often: we just take the sequence of actions under which the desired history sequence was possible initially and repetitively try this sequence every time the environment starts a new game. This implies that the equivalent first order MDP $(\mathcal{Y}, \mathcal{Z}, \tilde{R}, \tilde{\mu})$ is ergodic. That is, μ is equivalent to an ergodic first order MDP. □

A Notation and Definitions

A.1 Basic Mathematical Notation

A *set* is a collection of items called *elements* or *members*. For example, the letters of the English alphabet form a set where the individual letters, a, b, c and so on are the members of the set. We can denote this set by $\{a, b, c, \ldots, z\}$, though is should be noted that the order in which the elements are listed is not important. For example, $\{1, 2, 3\}$ and $\{3, 1, 2\}$ both describe the same set. In general the elements of a set can be anything, including other sets. The number of elements that a set can have is also unrestricted and can be finite or infinite. We call the set that has no elements the *empty set*, and denote this with the symbol $\varnothing$. We will denote arbitrary sets with symbols such as $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$. For sets that have a permanently fixed definition and collection of elements we will use symbols such as $\mathbb{B}$, $\mathbb{N}$ and $\mathbb{Z}$. By $x \in \mathcal{A}$ we mean that x is an element of the set $\mathcal{A}$ while $x \notin \mathcal{A}$ means that x is not an element of $\mathcal{A}$.

Let $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{Z}$ and $\mathbb{R}$ represent the sets of natural, rational, integer and real numbers respectively. We take $\mathbb{N}$ to exclude 0 as this appears to be the more common convention. When we want to include 0 we will use the notation $\mathbb{N}_0$. The symbol $=$ means that the expression on the left of the symbol is equal to the expression the right, while $\neq$ means that they are not equal. The symbol $:=$ will be used to indicate that the expression on the left is defined to be the expression on the right. Rather than listing the elements of a set it's usually more convenient to formally describe its members. For example, $\{x \; in \mathbb{N} : x > 10\}$ is the set of natural numbers greater than 10. In this was we define $\mathbb{R}^+ := \{x \in \mathbb{R} : x > 0\}$, $\mathbb{R}_0^+ := \{x \in \mathbb{R} : x \geq 0\}$ and similarly define $\mathbb{Q}^+$ and $\mathbb{Q}_0^+$.

By $\forall x \in \mathcal{X}$ we mean "for all x in the set $\mathcal{X}$" and by $\exists x \in \mathcal{X}$ we mean "there exists an x in the set $\mathcal{X}$". Thus the expression $\forall x \in \mathbb{R}, \exists i \in \mathbb{Z} : i > x$ would be interpreted as saying "for all real numbers x there exists an integer i such that i is greater than x". By $\exists!$ we mean "there exists a unique", by $\exists^\infty$ we mean "there exists infinitely many" and by $\nexists$ we mean "there does not exist". The symbols $\wedge$, $\vee$ and $\oplus$ mean "logical and", "logical or" and "logical exclusive or" respectively. By $\Rightarrow$ we mean "logically implies", by $\Leftrightarrow$ we mean "logically equivalent" and by $\neg$ we mean "logical not". When expressing logical equivalence in text, rather than using the symbol $\Leftrightarrow$ we will use the mathematical word "iff" which is short for "if and only if". By $:\Leftrightarrow$ we mean that the expression on the left is logically equivalent to the expression on the right by definition.

The symbols $\subset$ and $\subseteq$ are used to indicate the *proper subset* and *subset* relations respectively. More explicitly, $\mathcal{A} \subseteq \mathcal{B} :\Leftrightarrow \forall x \in \mathcal{A} : x \in \mathcal{B}$. And, $\mathcal{A} \subset \mathcal{B} :\Leftrightarrow \mathcal{A} \subseteq \mathcal{B} \wedge \mathcal{A} \neq \mathcal{B}$. For two sets $\mathcal{A}$ and $\mathcal{B}$ the *union* is defined to be $\mathcal{A} \cup \mathcal{B} := \{x : x \in \mathcal{A} \vee x \in \mathcal{B}\}$, the *intersection* is defined to be $\mathcal{A} \cap \mathcal{B} := \{x : x \in \mathcal{A} \wedge x \in \mathcal{B}\}$. The *relative complement* is defined to be $\mathcal{A} \setminus \mathcal{B} := \{x \in \mathcal{A} : x \notin \mathcal{B}\}$. Let $\wp(\mathcal{X}) := \{\mathcal{A} : \mathcal{A} \subseteq \mathcal{X}\}$ be the *power set* of the set $\mathcal{X}$ and let $|\mathcal{X}|$ be its cardinality. Thus if $\mathcal{X} := \{0, 9\}$ then $|\mathcal{X}| = 2$, $\wp(\mathcal{X}) = \{\varnothing, \{0\}, \{9\}, \{0, 9\}\}$ and $|\wp(\mathcal{X})| = 4$.

An *ordered n-tuple*, denoted $(x_1, x_2, \ldots, x_n)$ is a collection of $n \in \mathbb{N}$ items grouped to form a whole in such a way that the ordering is significant. When $n = 2$ we call the tuple an *ordered pair*. For example, $(a, 7, a, 1, 5)$ is an order 5-tuple while $(1, 2)$ and $(2, 1)$ are both ordered pairs where $(1, 2) \neq (2, 1)$. The *Cartesian product* of two sets $\mathcal{A}$ and $\mathcal{B}$ is denoted $\mathcal{A} \times \mathcal{B}$ and

defined to be $\{(a,b) : a \in \mathcal{A} \wedge b \in \mathcal{B}\}$. Thus for example if $\mathcal{A} := \{0,1\}$ and $\mathcal{B} := \{a,b,c\}$ then $\mathcal{A} \times \mathcal{B} = \{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$. Similarly we define for $n \in \mathbb{N}$ and sets $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ the Cartesian product $\mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n := \{(a_1, a_2, \ldots, a_n) : a_1 \in \mathcal{A}_1 \wedge a_2 \in \mathcal{A}_2 \wedge \cdots \wedge a_n \in \mathcal{A}_n\}$. When we take the Cartesian product of a set with itself $n \in \mathbb{N}$ times we denote this with an exponential,

$$\mathcal{A}^n := \underbrace{\mathcal{A} \times \mathcal{A} \times \cdots \times \mathcal{A}}_{n \text{ times}}.$$

A *relation* is any subset of a Cartesian product. When the Cartesian product is of just two sets, we call the relation a *binary relation*. For example the set $R := \{(x,y) \in \mathbb{R}^2 : y = x^2\}$ is a binary relation on $\mathbb{R}^2$. For a binary relation R we often denote the ordered pair $(a,b) \in R$ by placing the symbol between the two elements, for example aRb. Many familiar concepts can been seen as relations. For example we can view the usual concept of *less than* defined on $\mathbb{R}$ to be a binary relation denoted by the symbol $<$. When we write $2 < 3$ this is taken to mean that the ordered pair $(2,3)$ is in the subset of $\mathbb{R}^2$ that defines the less than relation.

A *(total) function* f from $\mathcal{X}$ to $\mathcal{Y}$, written $f : \mathcal{X} \to \mathcal{Y}$, is a relation that associates with every element of a set $\mathcal{X}$ one and only one element from a set $\mathcal{Y}$. Two different elements of $\mathcal{X}$ may be associated with the same element in $\mathcal{Y}$. A *partial function* f from $\mathcal{X}$ to $\mathcal{Y}$, written $f : \mathcal{X} \xrightarrow{o} \mathcal{Y}$, is a total function defined on a subset $\mathcal{Z} \subseteq \mathcal{X}$. We call $\mathcal{Z}$ is called the *domain* of f and denoted this $\mathrm{dom}(f)$. Thus for a total function $f : \mathcal{X} \to \mathcal{Y}$ we have $\mathcal{X} = \mathrm{dom}(f)$. For $x \in \mathrm{dom}(f)$ we denote the unique element of $\mathcal{Y}$ that corresponds to x by $f(x)$. For $x \notin \mathrm{dom}(f)$ our convention here will be to put $f(x) = \infty$. We define the *range* of a function f to be $\mathrm{ran}(f) := \{f(x) : x \in \mathrm{dom}(f)\}$. Because a function is a relation we can talk about its relation set. In the context of a function the function's relation set is called its *graph*. We will denote this by $\mathrm{graph}(f) := \{(x, f(x)) : x \in \mathrm{dom}(f)\}$. For two functions $f : \mathcal{X} \to \mathcal{Y}$ and $g : \mathcal{Z} \to \mathcal{X}$ we define the *composition* of f and g to be $f \circ g(z) := f(g(z))$.

A function $f : \mathcal{X} \to \mathcal{Y}$ is said to be an *injection* if $\forall x_1, x_2 \in \mathcal{X} : f(x_1) \neq f(x_2) \Rightarrow x_1 \neq x_2$. Thus, the function maps distinct elements of $\mathcal{X}$ to distinct elements of $\mathcal{Y}$. The function f is said to be a *surjection* if $\mathrm{ran}(f) = \mathcal{Y}$. A function is a *bijection* if it is both a injection and a surjection, formally, $\forall x \in \mathcal{X}, \exists! y \in \mathcal{Y} : f(x) = y$ and $\forall y \in \mathcal{Y}, \exists! x \in \mathcal{X} : f(x) = y$. Informally, a bijection pairs up the elements of the two sets $\mathcal{X}$ and $\mathcal{Y}$ and thus indicates that they are in some sense of equal size. We call two sets that are equal in this way *equipolent*, that is, sets $\mathcal{X}$ and $\mathcal{Y}$ are equipolent if there exists a bijection $f : \mathcal{X} \to \mathcal{Y}$. We call a set $\mathcal{A}$ that has an infinite number of elements *denumerable* if it is equipolent to the set $\mathbb{N}$. For example the set of integers $\mathbb{Z}$ is denumerable as we can define a bijection $f : \mathbb{N} \to \mathbb{Z}$ as follows:

$$f(i) := \begin{cases} (1-i)/2 & \text{for } i \text{ odd}, \\ i/2 & \text{for } i \text{ even}. \end{cases}$$

Thus, for $i = 1,2,3,4,5,6,\ldots$ we have $f(i) = 0,1,-1,2,-2,3,\ldots$. Clearly all the elements of $\mathbb{N}$ and $\mathbb{Z}$ are paired by this function. A set is said to be *countable* if it is either finite or denumerable. Not all sets are countable, in particular it can be proven that the infinite set $\mathbb{R}$ is not denumerable.

We will denote intervals on the real line with the usual bracket notation. Thus we define the *open interval* from $a \in \mathbb{R}$ to $b \in \mathbb{R}$ by $(a,b) := \{x \in \mathbb{R} : a < x \wedge x < b\}$ and we

define the *closed interval* as $[a, b] := \{x \in \mathbb{R} : a \leq x \wedge x \leq b\}$. While the notation for open intervals is unfortunately the same as for ordered pairs, the correct interpretation should be clear from the context. The notation can also be used to create half open intervals such as $(a, b] := \{x \in \mathbb{R} : a < x \wedge x \leq b\}$. We will extend this notation slightly and define $[a, b]_{\mathbb{Q}} := [a, b] \cap \mathbb{Q}$ and similarly for $(a, b)_{\mathbb{Q}}$ and so on.

By $x \gg y$ we mean that x is *much greater* than y. While this is a little ambiguous the significance of the statement should be clear from the context. It is used in situations were we want to emphasise the fact that x and y are not in some sense close as could be the case if we had only stated that $x > y$. Similarly we define $x \ll y$. By $x \propto y$ we mean that $\exists c \in \mathbb{R}$ such that $x = c \cdot y$ and we say that x is *proportional* to y.

We call $\lfloor \cdot \rfloor : \mathbb{R} \to \mathbb{Z}$ the *floor function* and define $\lfloor x \rfloor$ to be to be largest integer less than or equal to x. Similarly we call $\lceil \cdot \rceil : \mathbb{R} \to \mathbb{Z}$ the *ceiling function* and define $\lceil x \rceil$ to be the smallest integer greater than or equal to x. Thus, $\lfloor 0.26 \rfloor = 0$ and $\lfloor -3.7 \rfloor = -4$, while $\lceil 0.26 \rceil = 1$ and $\lceil -3.7 \rceil = -3$. For $i \in \mathbb{N}_0$ and $b \in \mathbb{N}$ we define i *modulus* b, denoted $i \bmod b$, to be the remainder when we divide i by b. For example, $13 \bmod 10 = 3$ and $161 \bmod 50 = 11$.

A useful function when integer values need to be compared is the *Kronecker delta function* $\delta : \mathbb{Z}^2 \to \{0, 1\}$ which is defined as follows,

$$\delta_{ij} := \begin{cases} 1 & \text{for } i = j, \\ 0 & \text{for } i \neq j. \end{cases}$$

A *total order* is a binary relation "$\leq$" on a set $\mathcal{A}$ such that $\forall a, b, c \in \mathcal{A}$ the following properties hold:

1. $a \leq a$ (reflexivity)
2. $a \leq b \wedge b \leq a \Rightarrow a = b$ (antisymmetry)
3. $a \leq b \wedge b \leq c \Rightarrow a \leq c$ (transitivity)
4. $a \leq b \vee b \leq a$ (trichotomy)

If the trichotomy property does not hold then the relation is called a *partial order* on the set. For convenience we define the relation $a \geq b :\Leftrightarrow a \leq b$. It is not hard to see that the usual less than or equal relation $\leq$ on $\mathbb{R}$ is a total ordering on the set $\mathbb{R}$.

A *strict order* is a binary relation "$<$" on a set $\mathcal{A}$ such that $\forall a, b, c \in \mathcal{A}$ the following properties hold:

1. $\neg(a < a)$ (irreflexivity)
2. $a < b \wedge b < a \Rightarrow a = b$ (antisymmetry)
3. $a < b \wedge b < c \Rightarrow a < c$ (transitivity)

For convenience we define the relation $a > b :\Leftrightarrow b < a$. Every strict order $<$ induces a partial order $\leq$ by defining $a \leq b :\Leftrightarrow a < b \vee a = b$. Similarly every partial order $\leq$ induces a strict order $\leq$ by the definition $a < b :\Leftrightarrow a \leq b \wedge a \neq b$.

For $n \in \mathbb{N}_0$ we define n *factorial*, denoted $n!$, to be

$$n! := \begin{cases} \prod_{i=1}^{n} i & \text{for } n > 0, \\ 1 & \text{for } n = 0. \end{cases}$$

Thus $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$. $n!$ is the number of unique orderings possible on a set of n distinct elements. This is used in the *binomial coefficient* function defined,

$$\binom{n}{r} := \frac{n!}{(n-r)!r!}$$

where $n, r \in \mathbb{N}_0$. This represents the number of unique unordered subsets with r elements that can be created from a set of n elements.

For a set $\mathcal{A}$ which is totally ordered by a relation $\leq$ we define the *minimum* of the set, denoted $\min \mathcal{A}$, to be the unique element $x \in \mathcal{A}$ such that $\forall y \in \mathcal{A} : x \leq y$, if such an element exists. For example, $\min\{3, 7, 2, 9, 5\} = 2$. Similarly we define the *maximum* of a set $\mathcal{A}$ and denoted this by $\max \mathcal{A}$. For finite sets the minimum and maximum always exist, however for infinite sets they may not. For example the interval $(1, 3) \subset \mathbb{R}$ has no minimum element, nor does the set $\{2^{-n} : n \in \mathbb{N}\}$. In many situations we are interested in the minimum etc. of the range of a function $f(x)$. In this case we will use the shorter notation $\min f(x) := \min \operatorname{ran}(f)$, if in fact such a value exists.

By $\arg\min_x f(x)$ we mean the set of values of x which minimises $f(x)$. Thus, $\arg\min_x(|x - 3.5| + 1) = \{3.5\}$ because when $x \in \{3.5\}$ the function $|x - 3.5| + 1$ is at its minimum value of 1. Often we will require any arbitrary member of the minimising solution set, in which case we will abuse the notation a little and consider $\arg\min_x f(x)$ to return just one arbitrary element from the set of minimising values.

The *logarithm* base $b \in \mathbb{R}^+$ of $x \in \mathbb{R}$ is denoted by $\log_b x$ and is defined to be the value $y \in \mathbb{R}$ such that $b^y = x$. So for example, $\log_3 9 = 2$ because $3^2 = 9$. We also define $\log x := \log_{10} x$, $\lg x := \log_2 x$ and $\ln x := \log_e x$ where $e := 2.7182818\ldots$.

A.2 Strings, Sequences and Codes

An *alphabet* is a finite set with cardinality at least two. For example the sets $\{a, b, c, d, \ldots, z\}$, $\{0, 1, 2, 3, \ldots, 9\}$ and $\{R, 7, hello\}$ are all alphabets. We call the elements of an alphabet *symbols*. We define the *binary alphabet* to be $\mathbb{B} := \{0, 1\}$. For any alphabet $\mathcal{A}$ we define $\mathcal{A}^n$ for $n \in \mathbb{N}$ to be the n-fold Cartesian product of the set. Thus $x \in \mathcal{A}^n$ is an ordered n-tuple of the form $x = (x_1, x_2, \ldots, x_n)$ where $\forall i \in \{1, \ldots, n\} : x_i \in \mathcal{A}$. Further define,

$$\mathcal{A}^+ := \bigcup_{n=1}^{\infty} \mathcal{A}^n$$

and $\mathcal{A}^* := \mathcal{A}^+ \cup \{\varnothing\}$. We will call $x \in \mathcal{A}^*$ a *string* and in particular we define $\epsilon := \varnothing$ and call this the *empty string*, thus $\mathcal{A}^* = \mathcal{A}^+ \cup \{\epsilon\}$. We call any set $\mathcal{L} \subseteq \mathcal{A}^*$ a *language*. Thus $\mathcal{L} := \{00, 01, 011, 11111\} \subseteq \mathbb{B}^*$ is a language over the binary alphabet.

We denote the binary operation of *concatenation* over strings by juxtaposition. That is, if $n, m \in \mathbb{N}$, $x = (x_1, \ldots, x_n) \in \mathcal{A}^n$ and $y = (y_1, \ldots, y_m) \in \mathcal{A}^m$ then $xy := (x_1, \ldots, x_n, y_1, \ldots, y_m) \in \mathcal{A}^{n+m}$. Additionally we define $\epsilon x = x\epsilon = x$. Clearly $\forall x, y, z \in \mathcal{A}^*$ we have $xy \in \mathcal{A}^*$ and $x(yz) = (xy)z$. Thus; $\mathcal{A}^*$ is closed under concatenation, concatenation is an associative binary relation and ϵ is the identity element under concatenation. As

concatenation is associative we can drop the parentheses and simply write xyz for the concatenation of three arbitrary strings. Further we will adopt a more compact notation and simply write $x = x_1x_2x_3\cdots x_n$ for an arbitrary string $x \in \mathcal{A}^*$ rather than the full n-tuple notation $x = (x_1, x_2, x_3, \ldots, x_n)$.

For convenience we will also extend the notation to allow products on strings and symbols using exponential notation. For example under the alphabet $\mathbb{B}$ the expression 0^n represents the string consisting of the symbol 0 repeated n times and $(001)^2 := 001001$. In places it will be useful to specify strings with an arbitrary unspecified symbol. We will use # for this purpose. Thus under the alphabet $\mathbb{B}$ the expression 0101#11 stands for any element of the set $0101x11$ where $x \in \mathbb{B}$. By $*$ we mean an arbitrary string rather than just one symbol. Thus under the alphabet $\mathbb{B}$ the expression 0101$*$11 stands for any element of the set $0101y11$ where $y \in \mathbb{B}^*$.

Let $l : \mathcal{A}^* \to \mathbb{N}_0$ be called the *length function* and define this to be $l(x) := n$ if $x \in \mathcal{A}^n$ and 0 if $x = \epsilon$. So for example, if our alphabet is $\mathbb{B}$ and we have the string $x := 01100010 \in \mathbb{B}^*$ then $l(x) = 8$ and $l(0^n1^m) = n + m$. Clearly,

$$\forall x, y \in \mathcal{A}^* \quad l(xy) = l(x) + l(y).$$

For an alphabet $\mathcal{A} := \{a_1, a_2, \ldots, a_n\}$ a strict ordering $a_1 < a_2 < \cdots < a_n$ induces a quasi-lexicographical ordering on $\mathcal{A}^*$:

$$\epsilon < a_1 < a_2 < \cdots < a_n < a_1a_1 < a_1a_2 < \cdots < a_1a_n < a_2a_1 < a_2a_2 < \cdots$$

$$\cdots < a_2a_n < a_3a_1 < a_3a_2 < \cdots < a_1a_1a_1 < a_1a_1a_2 < \cdots < a_na_na_n < \cdots .$$

Let $\mathrm{str} : \mathbb{N}_0 \to \mathcal{A}^*$ be a bijective function such that $\mathrm{str}(n)$ is the n^{th} string in the above quasi-lexicographical order on $\mathcal{A}^*$. The function $\mathrm{str}^{-1} : \mathcal{A}^* \to \mathbb{N}_0$ is its inverse. These functions will be used where we need to relate strings to natural numbers and vice versa.

For two strings $x, y \in \mathcal{A}^*$ we say that x is a *prefix* of y and denote this by $x \preceq y$ if $\exists z \in \mathcal{A}^* : xz = y$. The relation $\preceq$ is a partial order on the set of strings $\mathcal{A}^*$. If additionally $x \neq y$ then we say that x is a *proper prefix* of y and denote this by $x \prec y$. We say that a language $\mathcal{L} \subset \mathcal{A}^*$ is *prefix free* if $\forall x, y \in \mathcal{L} : x \preceq y \Rightarrow x = y$, that is, no element of $\mathcal{L}$ is a prefix of any other. For example, the language $\mathcal{C} := \{01, 001, 000011\}$ on the alphabet $\mathbb{B}$ is a prefix free language. Prefix free languages are important because any concatenation of arbitrary strings from such a language can always be "deconcatenated" to extract the original strings. For example consider the string 00001100100101001. If we know that the original language was $\mathcal{C}$ defined above then we can easily read through this string and separate out the original individual strings prior to concatenation; 000011, 001, 001, 01 and 001.

Any alphabet $\mathcal{A}$ is always trivially a prefix free language. This is because $\mathcal{A} \subseteq \mathcal{A}^*$ and $\mathcal{A}$ is always prefix free as each member of $\mathcal{A}$ is unique and has length 1 and thus no member can be a prefix of any other. Thus in basic examples where we need a prefix free language that is able to describe a finite list of possibilities simply using an alphabet with one symbol for each possibility is sufficient. For example the alphabet $\mathcal{A} := \{left, right, forward, backward, stop\}$ is a prefix free language.

For an alphabet $\mathcal{A}$ and a countable set $\mathcal{B}$ a *code* is an injective function $\phi : \mathcal{B} \to \mathcal{A}^*$. We call the elements of $\mathrm{ran}(\phi)$ *code strings*. If the set of code strings is a prefix free language then

we say that ϕ is an *instantaneous code*. Instantaneous codes are useful when we have a list of elements from a set and we need to represent these elements as a single string over some alphabet $\mathcal{A}$. For example consider the following instantaneous code $\phi : \mathbb{N}_0 \to \mathbb{B}^*$ defined $\phi(n) := 0^n 1$. Thus $\phi(3) = 0001$ is the code string for the number 3. If we take the code strings for the numbers 3, 1, 2 and 1 and concatenate them we get the string 00010100101. If somebody gave us this string and the instantaneous code function ϕ we could then work out the original sequence of numbers 3, 1, 2, 1.

For a string $x_1 x_2 x_3 \cdots x_k \in \mathcal{X}^k$ where $k \in \mathbb{N}$ we define the substring $x_{m:n} := x_m x_{m+1} x_{m+2} \cdots x_n$ for $m, n \in \{1, \ldots, k\}$ and $m \leq n$. For $m > n$ we define $x_{m:n} := \epsilon$. Similarly, $x_{<n} := x_1 x_2 \ldots x_{n-1}$. For string with alternating symbols, such as $x_1 y_1 x_2 y_2 \ldots x_k y_k \in (\mathcal{X} \times \mathcal{Y})^k$ where $k \in \mathbb{N}$ we similarly define $xy_{m:n} := x_m y_m x_{m+1} y_{m+1} \ldots x_n y_n$ and $xy_{<n} := x_1 y_1 \ldots x_{n-1} y_{n-1}$. Note that in this notation the x and y are pushed together as xy rather than xy so as to indicate that they are paired together.

A.3 Measure Theory

For a full explanation of how to define measures over sequences of strings see [3] or [9]. Our main deviation is in the way we represent conditional measures. The problem is that the usual way of representing a conditional measure over a string in time does not preserve the chronological order of the string. For example, the probably of x_n given $x_{<n}$ is usually represented as $\mu(x_n|x_{<n})$, however this places the x_n before the $x_{<n}$. We use a slightly different notation that keeps the chronological order intact by identifying the probability variables with an underscore. Thus, $\mu(x_{<n}\underline{x}_n) := \mu(x_n|x_{<n})$ and $\mu(xy_{<n}\underline{xy}_{n:n+3}) := \mu(xy_{n:n+3}|xy_{<n})$.

In places we will abuse the notation a little and write something like $\mu(yx_{<k}y_k\underline{x}_k) = \mu(y_k\underline{x}_k)$ to mean that $\forall yx_{<k} \in (\mathcal{Y} \times \mathcal{X})^*$ and $\forall y'x'_{<k} \in (\mathcal{Y} \times \mathcal{X})^*$ we have $\mu(yx_{<k}y_k\underline{x}_k) = \mu(y'x'_{<k}y_k\underline{x}'_k)$. Also, we write $\mu(y_{k-1}x_{k-1}y_x\underline{x}_k) = \mu(x_{k-1}\underline{x}_k)$ to mean that $\forall y_{k-1}, y_k \in \mathcal{Y}$ and $\forall y'_{k-1}, y'_k \in \mathcal{Y}$ we have $\mu(y_{k-1}x_{k-1}y_x\underline{x}_k) = \mu(y'_{k-1}x_{k-1}y'_k\underline{x}_k)$. Essentially all this says is that if some of the arguments have no effect at all on the value of a function then we can just leave them out and in effect define a new equivalent function over the reduced parameter space.

B Markov Decision Processes

In this section we will look more closely at Markov Decision Processes. The standard definition of a Markov Decision Process (MDP) is given in Bellman [1] or Kaelbling [7] as follows:

B.1 Definition. A **Markov Decision Process** is a tuple $(\mathcal{S}, \mathcal{A}, \Phi, T, R)$. The sets $\mathcal{S}$ and $\mathcal{A}$ are called the **state space** and **action space** respectively. The set $\Phi \subseteq \mathcal{S} \times \mathcal{A}$ is the set of admissible state-action pairs. The function $T : \Phi \times \mathcal{S} \to [0, 1]$ is called the **transition probability function** and the function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is called the **reward function**.

While $\mathcal{S}$ and $\mathcal{A}$ can be arbitrary sets, often in the theory they are finite. Here we will assume that they are finite unless otherwise indicated. The definition is sometimes modified by taking an expected value over the possible following states according to the transition probability function, $R(s, a) = \sum_{s' \in \mathcal{S}} T(s, a, s') \cdot R(s, a, s')$. This makes some of the equations that follow a little simpler. It's also possible to modify the reward function so that its domain is just the state space. We will call these *reduced Markov Decision Processes*:

B.2 Definition. A **reduced Markov Decision Process** (MDP) is a tuple $(\tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{\Phi}, \tilde{T}, \tilde{R})$. The sets $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{A}}$ are called the **state space** and **action space** respectively. The set $\tilde{\Phi} \subseteq \tilde{\mathcal{S}} \times \tilde{\mathcal{A}}$ is the set of admissible state-action pairs. The function $\tilde{T} : \tilde{\Phi} \times \tilde{\mathcal{S}} \to [0, 1]$ is called the **transition probability function** and the function $\tilde{R} : \tilde{\mathcal{S}} \to \mathbb{R}$ is called the **reward function**.

While the ability to define rewards over all possible combinations of transitions between states in the standard MDP definition might appear to be more powerful than the reduced definition, these two formalisms are in fact equivalent.

B.3 Theorem. *The standard definition of an MDP and the reduced definition of an MDP describe the same set of systems.*

Proof. We must show that any reduced MDP can be expressed as an equivalent standard MDP and vice versa. The first direction is trivial as a reduced MDP is just a special case of a standard MDP where $R(s, a, s') \equiv R(s')$. To show that any standard MDP can be expressed as a reduced MDP we include rewards and actions into the state.

Consider a standard MDP $M = (\mathcal{S}, \mathcal{A}, \Phi, T, R)$. From this we can define a reduced MDP $\tilde{M} = (\tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{\Phi}, \tilde{T}, \tilde{R})$ via the following definitions. Leave the action space unchanged, $\tilde{\mathcal{A}} := \mathcal{A}$, and define the state space in the reduced MDP to be $\tilde{\mathcal{S}} := \{\langle s, a', s' \rangle \in \Phi \times \mathcal{S} : T(s, a', s') > 0\} \cup \langle \varnothing, \varnothing \rangle \times \mathcal{S}$. Now define the new set of admissible state-action pairs as $\tilde{\Phi} := \{\langle \tilde{s}', \tilde{a}' \rangle \in \tilde{\mathcal{S}} \times \tilde{\mathcal{A}} : \tilde{s}' = \langle s, a, s' \rangle \wedge \langle s', \tilde{a}' \rangle \in \Phi\}$. Similarly we can define $\forall \langle \tilde{s}', \tilde{a}' \rangle \in \tilde{\Phi}$ and $\forall \tilde{s}'' \in \tilde{\mathcal{S}}$,

$$\tilde{T}(\tilde{s}', \tilde{a}', \tilde{s}'') \equiv \tilde{T}(\langle s, a, s' \rangle, \tilde{a}', \langle s', \tilde{a}', s'' \rangle) := T(s', \tilde{a}', s'').$$

Outside of this domain $\tilde{T}$ is undefined per our definition. Finally we can convert the reward function by defining it over our new state space $\forall \tilde{s}' \in \tilde{\mathcal{S}} : \tilde{R}(\tilde{s}') \equiv \tilde{R}(\langle s, a, s' \rangle) := R(s, a, s')$.

That $\tilde{M}$ conforms to the definition of a reduced MDP follows directly from the definitions above. To show that $\tilde{M}$ is equivalent to M requires some more work. Consider a state $s' \in \mathcal{S}$.

If s' is the initial state of M then its equivalent representation in $\tilde{M}$ will be $\langle \varnothing, \varnothing, s' \rangle \in \tilde{\mathcal{S}}$. If s' is not an initial state of M then $\exists s \in \mathcal{S}, \exists a \in \mathcal{A} : \langle s, a, s' \rangle \in \tilde{\mathcal{S}}$. Thus all states in $\mathcal{S}$ are represented in $\tilde{\mathcal{S}}$. The difference is that one state in $\mathcal{S}$ corresponds to a number of different states in $\tilde{\mathcal{S}}$. The fact that all of the actions are represented is trivial as the sets are the same.

Next we consider an arbitrary system transition in M and show that the corresponding transition in $\tilde{M}$ is equivalent. Consider when M is in state s' and takes action a' which causes it to transition to state s'' with probability $T(s', a', s'')$ and receive reward $R(s', a', s'')$. If s' is not an initial state of M then in $\tilde{M}$ the transition corresponds to being in state $\tilde{s}' = \langle s, a, s' \rangle$ for some $s \in \mathcal{S}$ and $a \in \mathcal{A}$ and taking action $\tilde{a}' = a'$. We know that action $\tilde{a}'$ is possible in state $\tilde{s}'$, that is, $\langle \tilde{s}', \tilde{a}' \rangle \in \tilde{\Phi}$, because from the definition of $\tilde{\Phi}$ we see that $\tilde{s}' = \langle s, a, s' \rangle$ where $\langle s', \tilde{a}' \rangle \in \Phi$. Furthermore, this action causes $\tilde{M}$ to transition to state $\tilde{s}''$ with probability $\tilde{T}(\tilde{s}', \tilde{a}', \tilde{s}'') = \tilde{T}(\langle s, a, s' \rangle, \tilde{a}', \langle s', \tilde{a}' s'' \rangle) = T(s', \tilde{a}', s'') = T(s', a', s'')$. That is, the transition in $\tilde{M}$ has the same probability as the original transition in M. Finally, the reward in $\tilde{M}$ is $\tilde{R}(\tilde{s}'') = \tilde{R}(\langle s', a', s'' \rangle) = R(s', a', s'')$ which is the same as for M. The case where s is an initial state of M is similar and causes no problems. □

It is perhaps instructive to contrast an example standard MDP and convert it to a reduced MDP using the above procedure in order to verify that the method works. However for just a three state standard MDP which contains all the different possibilities the conversion take several pages and so will not be presented here.

In our work it will be useful to be able to express the concept of MDPs in the more general mathematical framework of measures. This has the benefit that we can then use a common system of notation for both MDPs and other more general systems.

C Partially Observable Markov Decision Processes

C.1 Definition. A **Partially Observable Markov Decision Process** or **POMDP** is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, \mathcal{O})$. The sets $\mathcal{S}$ and $\mathcal{A}$ are called the **state space** and **action space** respectively. The function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is called the **transition probability function** and the function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is called the **reward function**. Ω is a set of possible observations that the agent can experience of its world. $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to \Omega$ is called the **observation function**.

References

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, New Jersey, 1957.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[3] C. S. Calude. *Information and Randomness*. Springer, Berlin, 2nd edition, 2002.

[4] J. L. Doob. *Stochastic Processes*. John Wiley & Sons, New York, 1953.

[5] M. Hutter. Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decisions. *Proceedings of the 12th Eurpean Conference on Machine Learning (ECML-2001)*, pages 226–238, December 2001.

[6] M. Hutter. Optimal sequential decisions based on algorithmic probability, 2003. 288 pages, draft, http://www.idsia.ch/∼marcus/ai/habil.htm.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[8] Ulrich Krengel. *Ergodic Theorems*. Walter de Gruyter, 1985.

[9] M. Li and P. M. B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2nd edition, 1997.

[10] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. Cambridge, MA, MIT Press, 1998.