# Chapter 2
# Universal Artificial Intelligence

## Practical Agents and Fundamental Challenges

**Tom Everitt and Marcus Hutter**

## 2.1 Introduction

Artificial intelligence (AI) bears the promise of making us all healthier, wealthier, and happier by reducing the need for human labour and by vastly increasing our scientific and technological progress.

Since the inception of the AI research field in the mid-twentieth century, a range of practical and theoretical approaches have been investigated. This chapter will discuss *universal artificial intelligence* (UAI) as a unifying framework and foundational theory for many (most?) of these approaches. The development of a foundational theory has been pivotal for many other research fields. Well-known examples include the development of Zermelo-Fraenkel set theory (ZFC) for mathematics, Turing-machines for computer science, evolution for biology, and decision and game theory for economics and the social sciences. Successful foundational theories give a precise, coherent understanding of the field, and offer a common language for communicating research. As most research studies focus on one narrow question, it is essential that the value of each isolated result can be appreciated in light of a broader framework or goal formulation. UAI offers several benefits to AI research beyond the general advantages of foundational theories just mentioned. Substantial attention has recently been called to the *safety* of autonomous AI systems [10]. A highly intelligent autonomous system may cause substantial unintended harm if constructed carelessly. The trustworthiness of autonomous agents may be much improved if their design is grounded in a formal theory (such as UAI) that allows formal verification of their behavioural properties. Unsafe designs can be ruled out at an early stage, and adequate attention can be given to crucial design choices.

T. Everitt (✉) · M. Hutter
Australian National University, Canberra, Australia
e-mail: Tom.Everitt@anu.edu.au

M. Hutter
e-mail: marcus.hutter@anu.edu.au



H. A. Abbass et al. (eds.), *Foundations of Trusted Autonomy*, Studies in Systems, Decision and Control 117, https://doi.org/10.1007/978-3-319-64816-3_2

UAI also provides a high-level blueprint for the design of practical autonomous agents, along with an appreciation of fundamental challenges (e.g. the induction problem and the exploration–exploitation dilemma). Much can be gained by addressing such challenges at an appropriately general, abstract level, rather than separately for each practical agent or setup. Finally, UAI is the basis of a general, non-anthropomorphic definition of intelligence. While interesting in itself to many fields outside of AI, the definition of intelligence can be useful to gauge progress of AI research.[1]

The outline of this chapter is as follows: First we provide general background on the scientific study of intelligence in general, and AI in particular (Sect. 2.2). Next we give an accessible description of the UAI theory (Sect. 2.3). Subsequent sections are devoted to applications of the theory: Approximations and practical agents (Sect. 2.4), high-level formulations and approaches to fundamental challenges (Sect. 2.5), and the safety and trustworthiness of autonomous agents (Sect. 2.6).

## 2.2 Background and History of AI

Intelligence is a fascinating topic, and has been studied from many different perspectives. Cognitive psychology and behaviourism are psychological theories about how humans think and act. Neuroscience, linguistics, and the philosophy of mind try to uncover how the human mind and brain works. Machine learning, logic, and computer science can be seen as attempts to make machines that *think*.

Scientific perspectives on intelligence can be categorised based on whether they concern themselves with thinking or acting (cognitive science vs. behaviourism), and whether they seek objective answers such as in logic or probability theory, or try to describe humans as in psychology, linguistics, and neuroscience. The distinction is illustrated in Table 2.1. The primary focus of AI is on *acting* rather than *thinking*, and on *doing the right thing* rather than *emulating humans*. Ultimately, we wish to build systems that solve problems and act appropriately; whether the systems are inspired by humans or follow philosophical principles is only a secondary concern.

**Induction and deduction**. Within the field of AI, a distinction can be made between systems focusing on *reasoning* and systems focusing on *learning*. *Deductive* reasoning systems typically rely on logic or other symbolic systems, and use search algorithms to combine inference steps. Examples of primarily deductive systems include medical expert systems that infer diseases from symptoms, and chess-playing agents deducing good moves. Since the deductive approach dominated AI in its early days, it is sometimes referred to as *good old-fashioned AI*.

[1] See [42, 43] for discussions about the intelligence definition.

**Table 2.1** Scientific perspectives on intelligence

| | Thinking | Acting |
|---|---|---|
| Humanly | Cognitive science | Turing test, behaviourism |
| Rationally | Laws of thought | **Doing the right thing** |

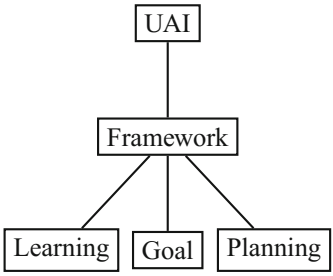
A more modern approach to AI shifts the focus from reasoning to learning. This *inductive approach* has become increasingly popular, both due to progress in machine learning and neural networks, and due to the failure of deductive systems to manage unknown and noisy environments. While it is possible for a human designer to construct a deductive agent for well-defined problems like chess, this task becomes unfeasible in tasks involving real-world sensors and actuators. For example, the reaction of any physical motor will never be *exactly* the same twice. Similarly, inferring objects from visual data could potentially be solved by a 'hard-coded' deductive system under 'perfect circumstances' where a finite number of geometric shapes generate perfectly predictable images. But in the real world, objects do not come from a finite number of geometric shapes, and camera images from visual sensors always contain a significant amount of noise. Induction-oriented systems that *learn* from data seem better fitted to handle such difficulties.

It is natural to imagine that some synthesis of inductive and deductive modules will yield superior systems. In practice, this may well turn out to be the case. From a theoretical perspective, however, the inductive approach is more-or-less self-sufficient. Deduction emerges automatically from a "simple" planning algorithm once the induction component has been defined, as will be made clear in the following section. In contrast, no general theory of AI has been constructed starting from a deductive system. See [67] (Sect. 1.1) for a more formal comparison.

## 2.3 Universal Artificial Intelligence

Universal Artificial Intelligence (UAI) is a completely general, formal, foundational theory of AI. Its primary goal is to give a precise mathematical answer to *what is the right thing to do in unknown environments*. UAI has been explored in great technical depth [28, 33], and has inspired a number of successful practical applications described in Sect. 2.4.

The UAI theory is composed of the following four components:




- **Framework**. Defines agents and environments, and their interaction.
- **Learning**. The learning part of UAI is based on Solomonoff induction. The general learning ability this affords is the most distinctive feature of UAI.
- **Goal**. In the simplest formulation, the goal of the agent will be to maximise reward.
- **Planning.** (Near) perfect planning is achieved with a simple expectimax search.

The following sections discuss these components in greater depth.

### *2.3.1 Framework*

The framework of UAI specifies how an *agent* interacts with an *environment*. The agent can take *actions* $a \in \mathscr{A}$. For example, if the agent is a robot, then the actions may be different kinds of limb movements. The environment reacts to the actions of the agent by returning a *percept* $e \in \mathscr{E}$. In the robot scenario, the environment is the real world generating a percept $e$ in the form of a camera image from the robot's visual sensors. We assume that the set $\mathscr{A}$ of actions and the set $\mathscr{E}$ of percepts are both finite.

The framework covers a very wide range of agents and environments. For example, in addition to a robot interacting with the real world, it also encompasses: A *chess-playing agent* taking actions $a$ in the form of chess moves, and receiving percepts $e$ in the form either of board positions or the opponent's latest move. The environment here is the chess board and the opponent. *Stock-trading agents* take actions $a$ in the form of buying and selling stocks, and receive percepts $e$ in the form of trading data from a *stock-market* environment. Essentially any application of AI can be modelled in this general framework.

A more formal example is given by the following toy problem, called *cheese maze* (Fig. 2.1). Here, the agent can choose from four actions $\mathscr{A} = \{\text{up, down, left, right}\}$ and receives one of two possible percepts $\mathscr{E} = \{\text{cheese, no cheese}\}$. The illustration shows a maze with cheese in the bottom right corner. The cheese maze is a commonly used toy problem in reinforcement learning (RL) [82].

- **Interaction histories**. The interaction between agent and environment proceeds in cycles. The agent starts taking an action $a_1$, to which the environment responds with a percept $e_1$. The agent then selects a new action $a_2$, which results in a

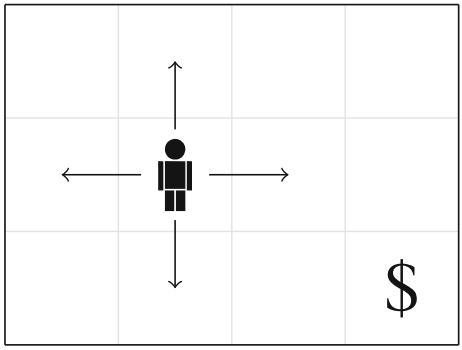



**Fig. 2.1** Cheese maze environment

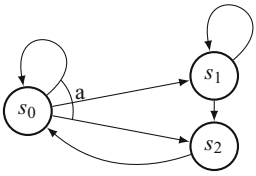
new percept $e_2$, and so on. The *interaction history* up until time $t$ is denoted $æ_{<t} = a_1 e_1 a_2 e_2 \ldots a_{t-1} e_{t-1}$. The set of all interaction histories is $(\mathcal{A} \times \mathcal{E})^*$.

- **Agent and environment**. We can give formal definitions of agents and environments as follows.

**Definition 1** (*Agent*) An *agent* is a *policy* $\pi : (\mathcal{A} \times \mathcal{E})^* \to \mathcal{A}$ that selects a new action $a_t = \pi(æ_{<t})$ given any history $æ_{<t}$.

**Definition 2** (*Environment*) An *environment* is a stochastic function $\mu : (\mathcal{A} \times \mathcal{E})^* \times \mathcal{A} \rightsquigarrow \mathcal{E}$ that generates a new percept $e_t$ for any history $æ_{<t}$ and action $a_t$. Let $\mu(e_t \mid æ_{<t} a_t)$ denote the probability that the next percept is $e_t$ given the history $æ_{<t} a_t$.

The agent and the environment are each other's analogues. Their possible interactions can be illustrated as a tree where the agent selects actions and the environment responds with percepts (see Fig. 2.2). Note in particular that the second percept $e_2$ can depend also on the first agent action $a_1$. In general, our framework puts no restriction on how long an action can continue to influence the behaviour of the environment and vice versa.




**Histories and states**. It is instructive to compare the generality of the *history* representation in the UAI framework to the *state* representation in standard RL. Standard RL is built around the notion of Markov decision processes (MDPs), where the agent transitions between *states* by taking actions, as illustrated to the right. The MDP specifies the *transition probabilities* $P(s' \mid s, a)$ of reaching new state $s'$ when taking action $a$ in current state $s$. An *MDP policy* $\tau : \mathcal{S} \to \mathcal{A}$ selects actions based on the state $s \in \mathcal{S}$.

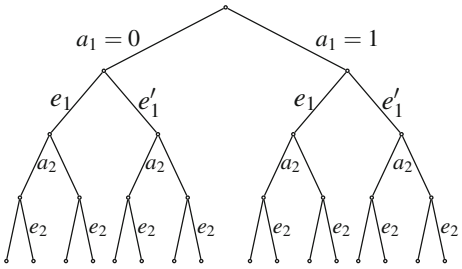



**Fig. 2.2** The tree of possible agent-environment interactions. The agent $\pi$ starts out with taking action $a_1 = \pi(\epsilon)$, where $\epsilon$ denotes the empty history. The environment $\mu$ responds with a percept $e_1$ depending on $a_1$ according to the distribution $\mu(e_1 \mid a_1)$. The agent selects a new action $a_2 = \pi(a_1 e_1)$, to which the environment responds with a percept $e_2 \sim \mu(\cdot \mid a_1 e_1 a_2)$

The history framework of UAI is more general than MDPs in the following respects:

- **Partially observable states.** In most realistic scenarios, the most recent observation or percept does not fully reveal the current state. For example, when in the supermarket I need to remember what is currently in my fridge; nothing in the percepts of supermarket shelves provide this information.[2]
- **Infinite number of states.** Another common assumption in standard RL is that the number of states is finite. This is unrealistic in the real world. The UAI framework does not require a finite state space, and UAI agents can learn without ever returning to the same state (see Sect. 2.3.2).
- **Non-stationary environments.** Standard RL typically assumes that the environment is stationary, in the sense that the transition probability $P(s' \mid s, a)$ remains constant over time. This is not always realistic. A car that changes travelling direction from a sharp wheel turn in dry summer road conditions may react differently in slippery winter road conditions. Non-stationary environments are automatically allowed for by the general definition of a UAI environment $\mu : (\mathcal{A} \times \mathcal{E})^* \times \mathcal{A} \rightsquigarrow \mathcal{E}$ (Definition 2). As emphasised in Chapter 11 of this book, the non-stationarity and non-ergodicity of the real world is what makes truly autonomous agents so challenging to construct and to trust.
- **Non-stationary policies.** Finally, UAI offers the following mild notational convenience. In standard RL, agents must be represented by sequences of policies $\pi_1, \pi_2, \ldots$ to allow for learning. The initial policy $\pi_1$ may for example be random, while later policies $\pi_t$, $t > 1$, will be increasingly directed to obtaining reward. In the UAI framework, policies $\pi : (\mathcal{A} \times \mathcal{E})^* \to \mathcal{A}$ depend on the entire interaction history. Any learning that is made from a history $æ_{<t}$ can be incorporated into a single policy $\pi$.

[2] Although histories can be viewed as states, this is generally not useful since it implies that no state is ever visited twice [28] (Sect. 3.3.3).

In conclusion, the history-based UAI framework is very general. Indeed, it is hard to find AI setups that cannot be reasonably modelled in this framework.

### 2.3.2 Learning

The generality of the UAI environments comes with a price: The agent will need much more sophisticated learning techniques than simply visiting each state many times, which is the basis of most learning in standard RL. This section will describe how this type of learning is possible, and relate it to some classical philosophical principles about learning.

A good image of a UAI agent is that of a newborn baby. Knowing nothing about the world, the baby tries different actions and experiences various sensations (percepts) as a consequence. Note that the baby does not initially know about any states of the world—only percepts. Learning is essential for intelligent behaviour, as it enables prediction and thereby adequate planning.

**Principles**. Learning or *induction* is an ancient philosophical problem, and has been studied for millennia. It can be framed as the problem of inferring a correct hypothesis from observed data. One of the most famous inductive principles is *Occam's razor*, due to William of Ockham (c. 1287–1347). It says to prefer the simplest hypothesis consistent with data. For example, relativity theory may seem like a complicated theory, but it is the *simplest* theory that we know of that is consistent with observed (non-quantum) physics data. Another ancient principle is due to Epicurus (341–270 BC). In slight conflict with Occam's razor, *Epicurus' principle* says to keep *all* hypothesis consistent with data. To discard a hypothesis one should have data that disconfirms it.

Thomas Bayes (1701–1761) derived a precise rule for how belief in a hypothesis should change with additional data. According to *Bayes' rule*, the *posterior belief* $\Pr(\text{Hyp} \mid \text{Data})$ should relate to the *prior belief* $\Pr(\text{Hyp})$ as:

$$\Pr(\text{Hyp} \mid \text{Data}) = \frac{\Pr(\text{Hyp})\Pr(\text{Data} \mid \text{Hyp})}{\sum_{H_i \in \mathscr{H}} \Pr(H_i)\Pr(\text{Data} \mid H_i)}$$

Here $\mathscr{H}$ is a class of possible hypotheses, and $\Pr(\text{Data} \mid \text{Hyp})$ is the *likelihood* of seeing the data under the given hypothesis. Bayes' rule has been highly influential in statistics and machine learning.

Two major questions left open by Bayes' rule are how to choose the prior $\Pr(\text{Hyp})$ and the class of possible hypotheses $\mathscr{H}$. Occam's razor tells us to weight simple hypotheses higher, and Epicurus tells us to keep any hypothesis for consideration. In other words, Occam says that $\Pr(\text{Hyp})$ should be large for simple hypotheses, and Epicurus prescribes using a wide $\mathscr{H}$ where $\Pr(\text{Hyp})$ is never 0. (Note that this does not prevent the posterior $\Pr(\text{Hyp} \mid \text{Data})$ from being 0 if the data completely

disconfirms the hypothesis.) While valuable, these principles are not yet precise. The following four questions remain:

I. What is a suitable general class of hypotheses $\mathscr{H}$?
II. What is a simple hypothesis?
III. How much higher should the probability of a simple hypothesis be compared to a complicated one?
IV. Is there any guarantee that following these principles will lead to good learning performance?

**Computer programs**. The solution to these questions come from a somewhat unexpected direction. In one of the greatest mathematical discoveries of the 20th century, Alan Turing invented the *universal Turing machine* (UTM). Essentially, a UTM can compute anything that can be computed at all. Today, the most well-known examples of UTMs are programming languages such as C, C++, Java, and Python. Turing's result shows that given unlimited resources, these programming languages (and many others) can compute the same set of functions: the so-called *computable functions*.

Solomonoff [77–79] noted an important similarity between deterministic environments $\mu$ and computer programs $p$. Deterministic environments and computer programs are both essentially input-output relations. A program $p$ can therefore be used as a hypothesis about the true environment $\mu$. The program $p$ is the hypothesis that $\mu$ returns percepts $e_{<t} = p(a_{<t})$ on input $a_{<t}$.

As hypotheses, programs have the following desirable properties:

- **Universal.** As Turing showed, computer programs can express any computable function, and thereby model essentially any environment. Even the universe itself has been conjectured computable [20, 33, 70, 87]. Using computer programs as hypotheses is thus in the spirit of Epicurus, and answers question I.
- **Consistency check.** To check whether a given computer program $p$ is consistent with some data/history æ$_{<t}$, one can usually run $p$ on input $a_{<t}$ and check that the output matches the observed percepts, $e_{<t} = p(a_{<t})$. (This is not always feasible due to the *halting problem* [27].)
- **Prediction.** Similarly, to predict the result of an action $a$ given a hypothesis $p$, one can run $p$ with input $a$ to find the resulting output prediction $e$. (A similar caveat with the halting problem applies.)
- **Complexity definition.** When comparing informal hypotheses, it is often hard to determine which hypothesis is simpler and which hypothesis is more complex (as illustrated by the *grue and bleen* problem [23]). For programs, complexity can be defined precisely. A program $p$ is a binary string interpreted by some fixed program interpreter, technically known as a *universal Turing machine* (UTM). We denote with $\ell(p)$ the length of this binary string $p$, and interpret the length $\ell(p)$ as the *complexity* of $p$. This addresses question II.[3]

[3]The technical question of which programming language (or UTM) to use remains. In *passive* settings where the agent only predicts, the choice is inessential [29]. In *active* settings, where the agent influences the environment, bad choices of UTMs

The complexity definition as length of programs corresponds well to what we consider *simple* in the informal sense of the word. For example, an environment where the percept always mirrors the action is given by the following simple program:

**procedure** MIRRORENVIRONMENT
  **while** true **do**:
    $x \leftarrow$ action input
    output percept $\leftarrow x$

In comparison, a more complex environment with, say, multiple players interacting in an intricate physics simulation would require a much longer program. To allow for stochastic environments, we say that an environment $\mu$ is *computable* if there exists a computer program $\mu_p$ that on input $æ_{<t}a_t$ outputs the distribution $\mu(e_t \mid æ_{<t}a_t)$ (cf. Definition 2).

**Solomonoff induction**. Based on the definition of complexity as length of strings coding computer programs, Solomonoff [77–79] defined a *universal prior* $\Pr(p) = 2^{-\ell(p)}$ for program hypotheses $p$, which gives rise to a *universal distribution* $M$ able to predict any computable sequence. Hutter [28] extended the definition to environments reacting to an agent's actions. The resulting *Solomonoff-Hutter universal distribution* can be defined as

$$M(e_{<t} \mid a_{<t}) = \sum_{p\,:\,p(a_{<t})=e_{<t}} 2^{-\ell(p)} \tag{2.1}$$

assuming that the programs $p$ are binary strings interpreted in a suitable programming language. This addresses question III.

Given some history $æ_{<t}a_t$, we can predict the next percept $e_t$ with probability:

$$M(e_t \mid æ_{<t}a_t) = \frac{M(e_{<t}e_t \mid a_{<t}a_t)}{M(e_{<t} \mid a_{<t})}.$$

This is just an application of the definition of conditional probability $P(A \mid B, C) = P(A, B \mid C)/P(B \mid C)$, with $A = e_t$, $B = e_{<t}$, and $C = a_{<t}a_t$.

**Prediction results**. Finally, will agents based on $M$ learn? (Question IV.) There are, in fact, a wide range of results in this spirit.[4] Essentially, what can be shown is that:

**Theorem 1** (Universal learning) *For any computable environment $\mu$ (possibly stochastic) and any action sequence $a_{1:\infty}$,*

$$M(e_t \mid æ_{<t}a_t) \to \mu(e_t \mid æ_{<t}a_t) \qquad as\ t \to \infty\ with\ \mu\text{-}probability\ 1.$$

(Footnote 3 continued)
can adversely affect the agent's performance [44], although remedies exist [46]. Finally, [54] describes a failed but interesting attempt to find an objective UTM.

[4] Overviews are provided by [28, 29, 48, 67], More recent technical results are given by [30, 39, 41, 45].

The convergence is quick in the sense that $M$ only makes a finite number of prediction errors on infinite interaction sequences $æ_{1:\infty}$. In other words, an agent based on $M$ will (quickly) learn to predict any true environment $\mu$ that it is interacting with. This is about as strong an answer to question V as we could possibly hope for. This learning ability also loosely resembles one of the key elements of human intelligence: That by interacting with almost any new 'environment' – be it a new city, computer game, or language – we can usually figure out how the new environment works by interacting with it.

### 2.3.3 Goal

Intelligence is to use (learnt) knowledge to achieve a goal. This section will define the goal of *reward maximisation* and argue for its generality.[5] For example, the goal of a chess agent should be to win the game. This can be communicated to the agent via reward, by giving the agent reward for winning, and no reward for losing or breaking game rules. The goal of a self-driving car should be to drive safely to the desired location. This can be communicated in a reward for successfully doing so, and no reward otherwise. More generally, essentially any type of goal can be communicated by giving reward for the goal's achievement, and no reward otherwise.

The reward is communicated to the agent via its percept $e$. We therefore make the following assumption on the structure of the agent's percepts:

**Assumption 1** (*Percept = Observation + Reward*) The percept $e$ is composed of an *observation* $o$ and a *reward* $r \in [0, 1]$; that is, $e = (o, r)$. Let $r_t$ be the reward associated with the percept $e_t$.

The observation part $o$ of the percept would be the camera image in the case of a robot, and the chess board position in case of a chess agent. The reward $r$ tells the agent how well it is doing, or how happy its designers are with its current performance. Given a *discount parameter* $\gamma$, the goal of the agent is to maximise the $\gamma$*-discounted return*

$$r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots.$$

The discount parameter $\gamma$ ensures that the sum is finite. It also means that the agent prefers getting reward sooner rather than later. This is desirable: For example, an agent striving to achieve its goal soon is more useful than an agent striving to achieve it in a 1000 years. The discount parameter should be set low enough so that the agent does not defer acting for too long, and high enough so that the agent does not become *myopic*, sacrificing substantial future reward for small short-term gains (compare *delayed gratification* in the psychology literature).

Reinforcement learning [82] is the study of agents learning to maximise reward. In our setup, Solomonoff's result (Theorem 1) entails that the agent will learn to predict

[5] Alternatives are discussed briefly in Sect. 2.6.2.

which actions or policies lead to percepts containing high reward. In practice, some care needs to be taken to design a sufficiently informative reward signal. For example, it may take a very long time before a chess agent wins a game 'by accident', leading to an excessively long exploration time before any reward is found. To speed up learning, small rewards can be added for moving in the right direction. A minor reward can for example be added for imitating a human [69].

The expected return that an agent/policy obtains is called *value*:

**Definition 3** (*Value*) The *value* of a policy $\pi$ in an environment $\mu$ is the expected return:

$$V_\mu^\pi = \mathbb{E}_\mu^\pi[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots].$$

### 2.3.4 Planning

The final component of UAI is planning. Given knowledge of the true environment $\mu$, how should the agent select actions to maximise its expected reward?

Conceptually, this is fairly simple. For any policy $\pi$, the expected reward $V_\mu^\pi = \mathbb{E}[r_1 + \gamma r_2 + \ldots]$ can be computed to arbitrary precision. Essentially, using $\pi$ and $\mu$, one can determine the histories $æ_{1:\infty}$ that their interaction can generate, as well as the relative probabilities of these histories (see Fig. 2.2). This is all that is needed to determine the expected reward. The discount $\gamma$ makes rewards located far into future have marginal impact, so the value can be well approximated by looking only finitely far into the future. Settling on a sufficient accuracy $\varepsilon$, the number of time steps we need to look ahead in order to achieve this precision is called the *effective horizon*.

To find the optimal course of action, the agent only needs to consider the various possible policies within the effective horizon, and choose the one with the highest expected return. The optimal behaviour in a known environment $\mu$ is given by

$$\pi_\mu^* = \arg\max_\pi V_\mu^\pi \tag{2.2}$$

We sometimes call this policy AI$\mu$. A full expansion of (2.2) can be found in [28] (p. 134). Efficient approximations are discussed in Sect. 2.4.1.

### 2.3.5 AIXI – Putting It All Together

This section describes how the components described in previous sections can be stitched together to create an optimal agent for unknown environments. This agent is called AIXI, and is defined by the optimal policy

$$\pi^*_M = \arg\max_\pi V^\pi_M \tag{2.3}$$

The difference to AI$\mu$ defined in (2.2) is that the true environment $\mu$ has been replaced with the universal distribution $M$ in (2.3). A full expansion can be found in [28] (p. 143). While AI$\mu$ is optimal when knowing the true environment $\mu$, AIXI is able to learn essentially any environment through interaction. Due to Solomonoff's result (Theorem 1) the distribution $M$ will converge to the true environment $\mu$ almost regardless of what the true environment $\mu$ is. And once $M$ has converged to $\mu$, the behaviour of AIXI will converge to the behaviour of the optimal agent AI$\mu$ which perfectly knows the environment. Formal results on AIXI's performance can be found in [28, 38, 46].

Put a different way, AIXI arrives to the world with essentially no knowledge or preconception of what it is going to encounter. However, AIXI quickly makes up for its lack of knowledge with a powerful learning ability, which means that it will soon figure out how the environment works. From the beginning and throughout its "life", AIXI acts optimally according to its growing knowledge, and as soon as this knowledge state is sufficiently complete, AIXI acts as well as any agent that knew everything about the environment from the start. Based on these observations (described in much greater technical detail by [28]), we would like to make the claim that AIXI defines the *optimal behaviour in any computable, unknown environment*.

**Trusting AIXI**. The AIXI formula is a precise description of the optimal behaviour in an unknown world. It thus offers designers of practical agents a target to aim for (Sect. 2.4). Meanwhile, it also enables safety researchers to engage in formal investigations of the consequences of this behaviour (Sects. 2.5 and 2.6). Having a good understanding of the behaviour and consequences an autonomous system strives towards, is essential for us being able to trust the system.

## 2.4 Approximations

The AIXI formula (2.3) gives a precise, mathematical description of the optimal behaviour in essentially any situation. Unfortunately, the formula itself is incomputable, and cannot directly be used in a practical agent. Nonetheless, having a description of the right behaviour is still useful when constructing practical agents, since it tells us what behaviour we are trying to approximate. The following three sections describe three substantially different approximation approaches. They differ widely in their approximation approaches, and have all demonstrated convincing experimental performance. Sect. 2.4.4 connects UAI with recent deep learning results.

### 2.4.1 MC-AIXI-CTW

MC-AIXI-CTW [85] is the most direct approximation of AIXI. It combines the Monte Carlo Tree Search algorithm for approximating expectimax planning, and the Context Tree Weighting algorithm for approximating Solomonoff induction. We describe these two methods next.

**Planning with sampling**. The expectimax planning principle described in Sect. 2.3.4 requires exponential time to compute, as it simulates all future possibilities in the planning tree seen in Fig. 2.2. This is generally far too slow for all practical purposes.

A more efficient approach is to randomly sample paths in the planning tree, as illustrated in Fig. 2.3. Simulating a single random path $a_t e_t \ldots a_m e_m$ only takes a small, constant amount of time. The average return from a number of such simulated paths gives an approximation $\hat{V}(æ_{<t} a_t)$ of the value. The accuracy of the approximation improves with the number of samples.

A simple way to use the sampling idea is to keep generating samples for as long as time allows for. When an action must be chosen, the choice can be made based on the current approximation. The sampling idea thus gives rise to an *anytime algorithm* that can be run for as long as desired, and whose (expected) output quality increases with time.

**Monte Carlo Tree Search**. The *Monte Carlo Tree Search* (MCTS) algorithm [2, 11, 36] adds a few tricks to the sampling idea to increase its efficiency. The sampling idea and the MCTS algorithm are illustrated in Fig. 2.3.

One of the key ideas of MCTS is in optimising the informativeness of each sample. First, the sampling of a next percept $e_k$ given a (partially simulated) history $æ_{<k} a_k$ should always be done according to the current best idea about the environment distribution; that is, according to $M(e_k \mid æ_{<k} a_k)$ for Solomonoff-based agents.

The sampling of actions is more subtle. The agent itself is responsible for selecting the actions, and actions that the agent knows it will not take, are pointless for the agent to simulate. As an analogy, when buying a car, I focus the bulk of my cognitive resources on evaluating the feasible options (say, the Ford and the Honda) and only

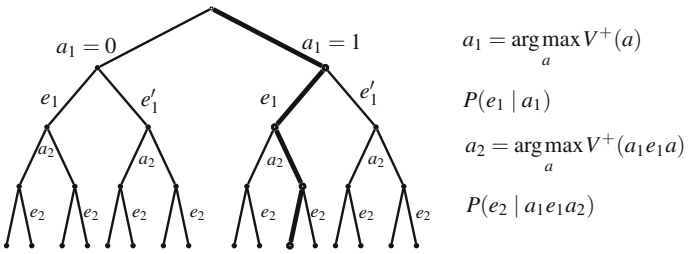



**Fig. 2.3** Sampling branches from the planning tree gives an *anytime algorithm*. Sampling actions according to the *optimistic value estimates* $V^+$ increases the informativeness of samples. This is one of the ideas behind the MCTS algorithm

briefly consider clearly infeasible options such as a luxurious Ferrari. Samples should be focused on plausible actions.

One way to make this idea more precise is to think of the sampling choice as a *multi-armed Bandit problem* (a kind of "slot machine" found in casinos). Bandit problems offer a clean mathematical theory for studying the allocation of resources between *arms* (actions) with *unknown returns* (value). One of the ideas emerging from the bandit literature is the *upper confidence bound* (UCB) algorithm that uses *optimistic value estimates* $V^+$. Optimistic value estimates add an exploration bonus for actions that has received comparatively little attention. The bonus means that a greedy agent choosing actions that optimise $V^+$ will spend a sufficient amount of resources exploring, while still converging on the best action asymptotically.

The MCTS algorithm uses the UCB algorithm for action sampling, and also uses some dynamic programming techniques to reuse sampling results in a clever way. The MCTS algorithm first caught the attention of AI researchers for its impressive performance in computer Go [22]. Go is infamous for its vast playout trees, and allowed the MCTS sampling ideas to shine.

**Induction with contexts**. Computing the universal probability $M(e_t \mid æ_{<t}a_t)$ of a next percept requires infinite computational resources. To be precise, conditional probabilities for the distribution $M$ are only *limit computable* [48]. We next describe how probabilities can be computed efficiently with the context tree weighting algorithm (CTW) [86] under some simplifying assumptions.

One of the key features of Solomonoff induction and UAI is the use of histories (Sect. 2.3.1), and the arbitrarily long time dependencies they allow for. For example, action $a_1$ may affect the percept $e_{1000}$. This is desirable, since the real world sometimes behaves this way. If I buried a treasure in my backyard 10 years ago, chances are I may find it if I dug there today. However, in most cases, it is the most recent part of the history that is most useful when predicting the next percept. For example, the most recent five minutes is almost always more relevant than a five minute time slot from a week ago for predicting what is going to happen next.

We define the *context of length* $c$ of a history as the last $c$ actions and percepts of the history:

**procedure** MIRRORENVIRONMENT
  **while** true **do**:
    $x \leftarrow$ action input
    output percept $\leftarrow x$

Relying on contexts for prediction makes induction not only computationally faster, but also conceptually easier. For example, if my current context is 0011, then I can use previous instances where I have been in the same context to predict the next percept:

$$a_1\, e_1\, a_2\, e_2 \ldots\ldots\ldots \underbrace{e_{t-2}\, a_{t-1}\, e_{t-1}\, a_t}_{\text{context of length 4}} \begin{cases} e_t = 0 \\ ? \\ ? \\ e_t = 1 \end{cases}$$

**Table 2.2** The tradeoff for the size of the considered context

| Short context | **More data** | Less precision |
|---|---|---|
| Long context | Less data | **Greater precision** |

Long contexts offer greater precision but require more data. The MCTS algorithm dynamically trades between them

In the pictured example, $P(1) = 2/3$ would be a reasonable prediction since in two thirds of the cases where the context 0011 occurred before it was followed by a 1. (*Laplace's rule* gives a slightly different estimate.) Humans often make predictions this way. For example, when predicting whether I will like the food at a Vietnamese restaurant, I use my experience from previous visits to Vietnamese restaurants.

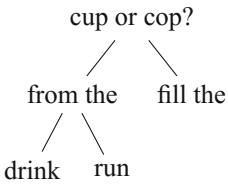
One question that arises when doing induction with contexts is how long or specific the context should be. Should I use the experience from all Vietnamese restaurants I have ever been to, or only this particular Vietnamese restaurant? Using the latter, I may have very limited data (especially if I have never been to the restaurant before!) On the other hand, using too unspecific contexts is not useful either: Basing my prediction on *all* restaurants I have ever been to (and not only the Vietnamese), will probably be too unspecific. Table 2.2 summarises the tradeoff between short and long contexts, which is nicely solved by the CTW algorithm.

The right choice of context length depends on a few different parameters. First, it depends on how much data is available. In the beginning of an agent's lifetime, the history will be short, and mainly shorter contexts will have a chance to produce an adequate amount of data for prediction. Later in the agent's life, the context can often be more specific, due to the greater amount of accumulated experience.

$$\ldots 0011\mathbf{1}\ldots 0011\mathbf{0}\ldots 0011\mathbf{1}\ldots \underbrace{0}_{e_{t-2}}\underbrace{0}_{a_{t-1}}\underbrace{1}_{e_{t-1}}\underbrace{1}_{a_t} \begin{cases} e_t = 0 \\ ? \\ ? \\ e_t = 1 \end{cases}$$

Second, the ideal context length may depend on the context itself, as aptly demonstrated by the example to the right. Assume you just heard the word *cup* or *cop*. Due to the similarity of the words, you are unable to tell which of them it was. If the most recent two words (i.e. the context) was *fill the*, you can infer the word was *cup*, since *fill the cop* makes little sense. However, if the most recent two words were *from the*, then further context will be required, as both *drink from the cup* and *run from the cop* are intelligible statements.

**Context Tree Weighting**. The Context Tree Weighting (CTW) algorithm is a clever way of adopting the right context length based both on the amount of data available and on the context. Similar to how Solomonoff induction uses a sum over all possible computer programs, the CTW algorithm uses a sum over all possible context trees up to a maximum depth $D$. For example, the context trees of depth $D \leq 2$ are the trees:




The structure of a tree encodes when a longer context is needed, and when a shorter context suffices (or is better due to a lack of data). For example, the leftmost tree corresponds to an iid process, where context is never necessary. The tree of depth $D = 1$ posits that contexts of length 1 always are the appropriate choice. The rightmost tree says that if the context is 1, then that context suffices, but if the most recent symbol is 0, then a context of length two is necessary. Veness et al. [85] offer a more detailed description.

For a given maximum depth $D$, there are $O(2^{2^D})$ different trees. The trees can be given binary encodings; the coding of a tree $\Gamma$ is denoted $CL(\Gamma)$. Each tree $\Gamma$ gives a probability $\Gamma(e_t \mid æ_{<t}a_t)$ for the next percept, given the context it prescribes using. Combining all the predictions yields the CTW distribution:

$$CTW(e_{<t} \mid a_{<t}) = \sum_{\Gamma} 2^{-CL(\Gamma)} \Gamma(e_{<t} \mid a_{<t}) \tag{2.4}$$

The CTW distribution is tightly related to the Solomonoff-Hutter distribution (2.1), the primary difference being the replacing of computer programs with context trees. Naively computing $CTW(e_t \mid æ_{<t}a_t)$ takes double-exponential time. However, the CTW algorithm [86] can compute the prediction $CTW(e_t \mid æ_{<t}a_t)$ in $O(D)$ time. That is, for fixed $D$, it is a constant-time operation to compute the probability of a next percept for the current history. This should be compared with the infinite computational resources required to compute the Solomonoff-Hutter distribution $M$.

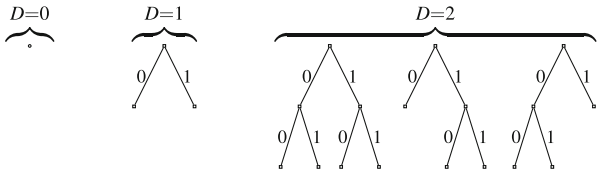
Despite its computational efficiency, the CTW distribution manages to make a weighted prediction based on all context trees within the maximum depth $D$. The relative weighting between different context trees changes as the history grows, reflecting the success and failure of different context trees to accurately predict the next percept. In the beginning, the shallower trees will have most of the weight due to their shorter code length. Later on, when the benefit of using longer contexts start to pay off due to the greater availability of data, the deeper trees will gradually gain an advantage, and absorb most of the weight from the shorter trees. Note that CTW handles partially observable environments, a notoriously hard problem in AI.

**MC-AIXI-CTW**. Combining the MCTS algorithm for planning with the CTW approximation for induction yields the MC-AIXI-CTW agent. Since it is history based, MC-AIXI-CTW handles hidden states gracefully (as long as long-term dependencies are not too important). The MC-AIXI-CTW agent can run on a standard desktop computer, and achieves impressive practical performance. For example, MC-AIXI-CTW can learn to play Rock Paper Scissors, TicTacToe, Kuhn Poker,

and even Pacman, just by trying actions and observing percepts, and without additional knowledge about the rules of the game [85]. For computational reasons, in PacMan the agent did not view the entire screen, only a compressed version telling it the direction of ghosts and nearness of food pellets (16 bits in total). Although less informative, this drastically reduced the number of bits per interaction cycle, and allowed for using a reasonably short context. Thereby the less informative percepts actually made the task computationally easier.

**Other approximations of Solomonoff induction**. Although impressive, a major drawback of the CTW approximation of Solomonoff induction is that the CTW-agents cannot learn time dependencies longer than the maximum depth $D$ of the context trees. This means that MC-AIXI-CTW will underperform in situations where long-term memory is required.

A few different approaches to approximating Solomonoff induction has been explored. Generally they are less well-developed than CTW, however.




A seemingly minor generalisation of CTW is to allow loops in context trees. Such loops allow context trees of a limited depth to remember arbitrarily long dependencies, and can significantly improve performance in domains where this is important [12]. However, the loops break some of the clean mathematics of CTW, and predictions can no longer be computed in constant time. Instead, practical implementations must rely on approximations such as simulated annealing to estimate probabilities.

The *speed prior* [71] is a version of the universal distribution $M$ where the prior is based on both program length and program runtime. The reduced probability of programs with long runtime makes the speed prior computable. It still requires exponential or double-exponential computation time, however [18]. Recent results show that program-based compression can be done incrementally [19]. These results can potentially lead to the development of a more efficient anytime-version of the speed prior. It is an open question whether such a distribution can be made sufficiently efficient to be practically useful.

### *2.4.2 Feature Reinforcement Learning*

Feature reinforcement learning (**Φ**MDP) [31, 32] takes a more radical approach to reducing the complexity of Solomonoff induction. While the CTW algorithm outputs a distribution of the same *type* as Solomonoff induction (i.e. a distribution

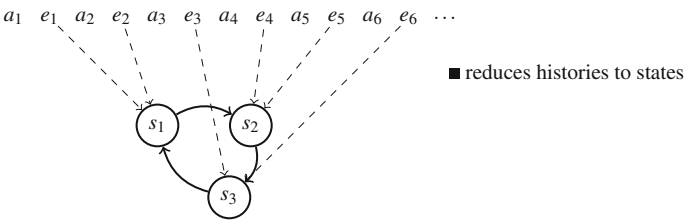



**Fig. 2.4** ΦMDP infers an underlying state representations from a history

over next percepts), the ΦMDP approach instead tries to infer states from histories (see Fig. 2.4).

Histories and percepts are often generated by an underlying set of state transitions. For example, in classical physics, the *state of the world* is described by the position and velocity of all objects. In toy examples and games such as chess, the board state is mainly what matters for future outcomes. The usefulness of thinking about the world in terms of states is also vindicated by simple introspection: with few exceptions, we humans translate our histories of actions and percepts into states and transitions between states such as *being at work* or *being tired*.

In standard applications of RL with agents that are based on states, the designers of the agent also design a mechanism for interpreting the history/percept as a state. In ΦMDP, the agent is instead programmed to learn the most useful state representation itself. Essentially, a state representation is *useful* if it predicts rewards well. To avoid overfitting, smaller MDPs are also preferred, in line with Occam's razor.

The computational flow of a ΦMDP agent is depicted in Fig. 2.5. After a percept $e_{t-1}$ has been received, the agent searches for the best map $\mathbf{\Phi}$: history $\mapsto$ state for its current history $æ_{<t}$. Given the state transitions provided by $\mathbf{\Phi}$, the agent can calculate transition and reward probabilities by frequency estimates. The value functions are computed by standard MDP techniques [82] or modern PAC-MDP algorithms, which allows for a near-optimal action to be found in polynomial time. Intractable planning is avoided. Once the optimal action has been determined, the agent submits it to the environment and waits for a new percept.

ΦMDP is not the only approach for inferring states from percepts. Partially observable MDPs (POMDPs) [35] is another popular approach. However, the learning of POMDPs is still an open question. The *predictive state representation* [51] approach also lacks a general and principled learning algorithm. In contrast, initial consistency results for ΦMDP show that under some assumptions, ΦMDP agents asymptotically learn the correct underlying MDP [80].

A few different practical implementations of ΦMDP agents have been tried. For toy problems, the ideal MDP-reductions can be computed with brute-force [56]. This is not possible in harder problems, where Monte Carlo approximations can be used instead [57]. Finally, the idea of context trees can be used also for ΦMDP. The context tree given the highest weight by the CTW algorithm can be used as a map $\mathbf{\Phi}$

that considers the current context as the state. The resulting ΦMDP agent exhibits similar performance as the MC-AIXI-CTW agent.

Generalisations of the ΦMDP agent include generalising the states to feature vectors [31] (whence the name *feature* RL). As mentioned above on page xxx, loops can be introduced to enable long-term memory of context trees [12]. The Markov property of states can be relaxed in the *extreme state aggregation* approach [34]. A somewhat related idea using neural networks for the feature extraction was recently suggested [74].

### 2.4.3 Model-Free AIXI

Both MC-AIXI-CTW and ΦMDP are *model-based* in the sense that they construct a *model* for how the environment reacts to actions. In MC-AIXI-CTW, the models are the context trees, and in ΦMDP, the model is the inferred MDP. In both cases, the models are then used to infer the best course of action. *Model-free* algorithms skip the middle step of inferring a model, and instead infer the value function directly.

Recall that $V^\pi(æ_{<t}a_t)$ denotes the expected return of taking action $a_t$ in history $æ_{<t}$, and thereafter following the superscripted policy $\pi$, and that $V^*(æ_{<t}a_t)$ denotes expected return of $a_t$ and thereafter following an optimal policy $\pi^*$. The optimal value function $V^*$ is particularly useful for acting: If known, one can act optimally by always choosing action $a_t = \arg\max_a V^*(æ_{<t}a)$. This action $a_t$ will be optimal under the assumption that future actions are optimal, which is easily achieved by selecting them from $V^*$ in the same way. In other words, being greedy with respect to $V^*$ gives an optimal policy. In model-free approaches, $V^*$ is inferred directly from data. This removes the need for an extra planning step, as the best action is simply the action with the highest $V^*$-value. Planning is thereby incorporated into the induction step.

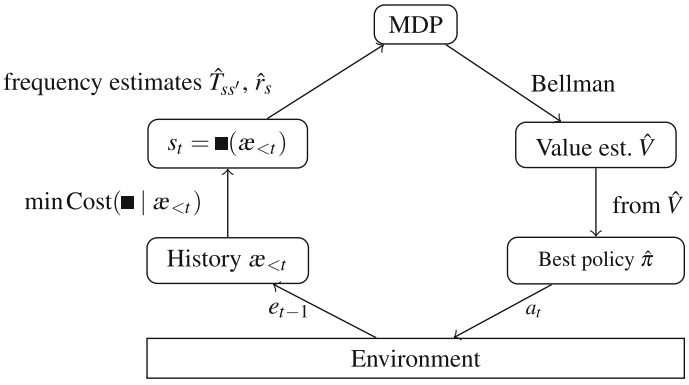



**Fig. 2.5** Computational flow of a ΦMDP-agent

Many of the most successful algorithms in traditional RL are model-free, including Q-learning and SARSA [82]. The first computable version of AIXI, the AIXI$tl$ agent [28] (Sect. 6.2), was a model-free version of AIXI. A more efficient model-free agent *compress and control* (CNC) was recently developed by developed by Veness et al. [84]. The performance of the CNC agent is substantially better than what has been achieved with both the MC-AIXI-CTW approach and the **Φ**MDP approach. CNC learned to play several Atari games (Pong, Bass, and Q*Bert) just by looking at the screen, similar to the subsequent famous *Deep Q-Learning* algorithm (DQN) [53] discussed in the next section. The CNC algorithm has not yet been generalised to the general, history-based case. The version described by Veness et al. [84] is developed only for fully observable MDPs.

### *2.4.4 Deep Learning*

*Deep learning* with artificial neural networks has gained substantial momentum the last few years, demonstrating impressive practical performance in a wide range of learning tasks. In this section we connect some of these results to UAI.

A standard (feed-forward) neural network takes a fixed number of inputs, propagates them through a number of hidden layers of differentiable activation functions, and outputs a label or a real number. Given enough data, such networks can learn essentially any function. In one much celebrated example with particular connection to UAI, a deep learning RL system called DQN learned to play 49 different Atari video games at human level just by watching the screen and knowing the score (its reward) [53]. The wide variety of environments that the DQN algorithm learned to handle through interaction alone starts to resemble the general learning performance exhibited by the theoretical AIXI agent.

One limitation with standard feed-forward neural networks is that they only accept a fixed size of input data. This fits poorly with sequential settings such as text, speech, video, and UAI environments $\mu$ (see Definition 2) where one needs to remember the past in order to predict the future. Indeed, a key reason that DQN could learn to play Atari games using feed-forward networks is that Atari games are mostly *fully observable*: everything one needs to know in order to act well is visible on the screen, and no memory is required (compare *partial observability* discussed in Sect. 2.3.2).

Sequential data is better approached with so-called *recurrent neural networks*. These networks have a "loop", so that part of the output of the network at time $t$ is fed as input to the network at time $t + 1$. This, in principle, allows the network to remember events for an arbitrary number of time steps. *Long short-term memory networks* (LSTMs) are a type of recurrent neural networks with a special pathway for preserving memories for many time steps. LSTMs have been highly successful in settings with sequential data [50]. *Deep Recurrent Q-Learning* (DRQN) is a generalisation of DQN using LSTMs. It can learn a partially observable version of Atari games [25] and the 3D game Doom [37]. DQN and DRQN are model-free algorithms,

and so are most other practical successes with deep learning in RL. References [58, 73] (Chap. 5) provide more extensive surveys of related work.

Due to their ability to cope with partially observable environments with long-term dependencies between events, we consider AIs based on recurrent neural networks to be interesting deep-learning AIXI approximations. Though any system based on a finite neural network must necessarily be a less general learner than AIXI, deep neural networks tend to be well-fitted to problems encountered in our universe [49].

The connection between the abstract UAI theory and practical state-of-the-art RL algorithms underlines the relevancy of UAI.

## 2.5 Fundamental Challenges

Having a precise notion of intelligent behaviour allows us to identify many subtle issues that would otherwise likely have gone unnoticed. Examples of issues that have been identified or studied in the UAI framework include:

- Optimality [28, 44, 46]
- Exploration vs. exploitation [46, 61]
- How should the future be discounted? [40]
- What is a practically feasible and general way of doing joint learning and planning [32, 84, 85]
- What is a "natural" universal Turing machine or programming language? [44, 54]
- How should embodied agents reason about themselves? [17]
- Where should the rewards come from? [16, 26, 68]
- How should agents reason about other agents reasoning about themselves? [47]
- Personal identity and teleportation [62, 63].

In this section we will mainly focus on the optimality issues and the exploration vs. exploitation studies. The question of where rewards should come from, together with other safety related issues will be treated in Sect. 2.6. For the other points, we refer to the cited works.

### *2.5.1 Optimality and Exploration*

What is the optimal behaviour for an agent in any unknown environment? The AIXI formula is a natural answer, as it specifies which action generates the highest expected return with respect to a distribution $M$ that learns any computable environment in a strong sense (Theorem 1).

The question of optimality is substantially more delicate than this however, as illustrated by the common dilemma of when to explore and when to instead exploit knowledge gathered so far. Consider, for example, the question of whether to try a new restaurant in town. Trying means risking a bad evening, spending valued dollars

on food that is potentially much worse than what your favourite restaurant has to offer. On the plus side, trying means that you learn whether it is good, and chances are that it is better than your current favourite restaurant.

The answer AIXI gives to this question is that the restaurant should be tried if and only if the expected return (utility) of trying the restaurant is greater than not trying, accounting for the risk of a bad evening and the possibility of finding a new favourite restaurant, as well as for their relative subjective probabilities. By giving this answer, AIXI is *subjectively* optimal with respect to its belief $M$. However, the answer is not fully connected to *objective* reality. Indeed, either answer (*try* or *don't try*) could have been justified with some belief.[6] While the convergence result Theorem 1 shows that $M$ will correctly predict the rewards on the followed action sequence, the result does not imply that the agent will correctly predict the reward of actions that it is *not* taking. If the agent never tries the new restaurant, it will not learn how good it is, even though it would learn to perfectly predict the quality at the restaurants it is visiting. In technical terms, $M$ has guaranteed *on-action* convergence, but not guaranteed *off-action* convergence [28] (Sect. 4.1.3).

An alternative optimality notion is *asymptotic optimality*. An agent is asymptotically optimal if it eventually learns to obtain the maximum possible amount of reward that can be obtained from the environment. No agent can obtain maximum possible reward directly, since the agent must first spend some time learning which environment is the true one. That AIXI is not asymptotically optimal was shown by [44, 61]. In general, it is impossible for an agent to be both Bayes-optimal and asymptotically optimal [61].

| | | |
|---|---|---|
| Bayes-optimality | Subjective | **Immediate** |
| Asymptotic optimality | **Objective** | Asymptotic |

Among other benefits, the interaction between asymptotically optimal agents yields clean game-theoretic results. Almost regardless of their environment, asymptotically optimal agents will converge on a Nash-equilibria when interacting [47]. This result provides a formal solution to the long-open *grain-of-truth* problem, connecting expected utility theory with game theory.

### 2.5.2 Asymptotically Optimal Agents

AIXI is Bayes-optimal, but is not asymptotically optimal. The reason is that AIXI does not explore enough. There are various ways in which one can create more explorative agents. One of the simplest ways is by letting the agent act randomly for

[6] In fact, for any decision there is one version of AIXI that prefers each option, the different versions of AIXI differing only in which programming language (UTM) is used in the definition of the universal distribution $M$ (2.1) [44].

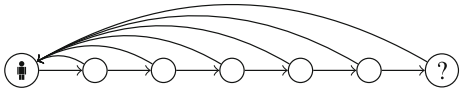



**Fig. 2.6** In this environment, focused exploration far outperforms random exploration. Focused exploration finds out the content at the question mark in 6 time steps. With random exploration, the expected number of steps required is $2^6$, an exponential increase

periods of time. A fine balance needs to be struck between doing this enough so that the true environment is certain to be discovered, and not doing it too much so that the full benefits of knowing the true environment can be reaped (note that the agent can never know for certain that it has now found the true environment). If exploration is done in just the right amount, this gives rise to a (weakly) asymptotically optimal agent [38].

**Optimistic agents**. Exploring randomly is often inefficient, however. Consider for example the environment depicted in Fig. 2.6. An agent that purposefully explores the rightmost question mark, finds out the truth exponentially faster than a randomly exploring agent. For a real-world example, consider how long it would take you to walk into a new restaurant and order a meal by performing random actions. Going to a restaurant with the intention of finding out how good the food is tends to be much more efficient.

*Optimism* is a useful principle for devising focused exploration. In standard RL, this is often done with *positive initialisation* of value estimates. Essentially, the agent is constructed to believe that "there is a path to paradise", and will systematically search for it. Optimism thus leads to strategic exploration. In the UAI framework, optimistic agents can be constructed using a growing, finite class $\mathscr{N}_t$ of possible environments, and act according to the environment $\nu \in \mathscr{N}_t$ that promises the highest expected reward. Formally, AIXI's action selection (2.3) is replaced by

$$a_t = \arg\max_a \max_{\nu \in \mathscr{N}_t} V_\nu(æ_{<t}\, a).$$

Optimistic agents are asymptotically optimal [81].

**Thompson-sampling**. A third way of obtaining asymptotically optimal agents is through Thompson-sampling. Thompson-sampling is more closely related to AIXI than optimistic agents. While AIXI acts according to a weighted average over all consistent environments, a Thompson-sampling agent randomly picks *one* environment $\nu$ and acts as if $\nu$ were the true one for one effective horizon. When the effective horizon is over, the agent randomly picks a new environment $\nu'$. Environments are sampled from the agent's posterior belief distribution at the time of the sampling.

Since Thompson-sampling agents act according to one environment over some time period, they explore in a strategic manner. Thompson-sampling agents are also asymptotically optimal [46].

## 2.6 Predicting and Controlling Behaviour

The point of creating intelligent systems is that they can act and make decisions without detailed supervision or micromanagement. For example, Sect. 18.5.3 in this book describes the application of autonomous AI systems to unmanned space missions. However, with increasing autonomy and responsibility, and with increasing intelligence and capability, there inevitably comes a risk of systems causing substantial harm [10]. The UAI framework provides a means for giving formal proofs about the behaviour of intelligent agents. While no practical agent may perfectly implement the AIXI ideal, having a sense of what behaviour the agent strives towards can still be highly illuminating.

We start with some general observations. What typically distinguishes an autonomous agent from other agents is that it decides itself what actions to take to achieve a goal. The goal is central, since a system without a goal must either be instructed on a case-by-case basis, or work without clear direction. Systems optimising for a goal may find surprising paths towards that goal. Sometimes these paths are desirable, such as when a Go or Chess program finds moves no human would think of. Other times, the results are less desirable. For example, [8] used an evolutionary algorithm to optimise circuit design of a radio controller. Surprisingly, the optimal design found by the algorithm did not contain any oscillator, a component typically required. Instead the system had evolved a way of using radio waves from a nearby computer. While clever, the evolved controller would not have worked in other circumstances.

In general, artificial systems optimise the *literal* interpretation of the goal they are given, and are indifferent to implicit *intentions* of the designer. The same behaviour is illustrated in fairy tales of "evil genies", such as with King Midas who wished that everything he touched would turn to gold. Closer to the field of AI is Asimov's ([7]) three laws of robotics. Asimov's stories illustrate some problems with AIs interpreting these laws overly literally.

The examples above illustrate how special care must be taken when designing the goals of autonomous systems. Above, we used the simple goal of maximising reward for our UAI agents (Sect. 2.3.3). One might think that maximising reward given by a human designer should be safe against most pitfalls: After all, the ultimate goal of the system in this case is pretty close to making its human designer happy. This section will discuss some issues that nonetheless arise, and ways in which those issues can potentially be addressed. For more comprehensive overviews of safety concerns of intelligent agents, see [4, 21, 76, 83].

### *2.6.1 Self-Modification*

Autonomous agents that are intelligent and have means to affect the world in various ways may, in principle, turn those means towards modifying itself. An autonomous

agent may for example find a way to rewrite its own source code. Although present AI systems are not yet close to exhibiting the required intelligence or "self-awareness" required to look for such self-modifications, we can still anticipate that such abilities will emerge in future AI systems. By modelling self-modification formally, we can assess some of the consequences of the self-modification possibility, and look for ways to manage the risks and harness the possibilities. Formal models of self-modification have been developed in the UAI-framework [15, 65, 66]. We next discuss some types of self-modification in more detail.

**Self-improvement**. One reason an intelligent agent may want to self-modify could be for improving its own hardware or software. Indeed, Omohundro [60] lists *self-improvement* as a fundamental drive of any intelligent system, since a better future version of the agent would likely be better at achieving the agent's goal. The Gödel machine [72] is an agent based on this principle: The Gödel machine is able to change any part of its own source code, and uses part of its computational resources to find such improvements. The claim is that the Gödel machine will ultimately be an optimal agent. However, Gödel's second incompleteness theorem and its corollaries imply fundamental limitations to formal systems' ability to reason about themselves. Yudkowsky and Herreshoff [89] claim some progress on how to construct self-improving systems that sidestep these issues.

Though self-improvement is generally positive as it allows our agents to become better over time, it also implies a potential safety problem. An agent improving itself may become more intelligent than we expect, which admonishes us to take extra care in designing agents that can be trusted regardless of their level of intelligence [10].

**Self-modification of goals**. Another way an intelligent system may use its self-modification capacity is to replace its goal with something easier, for example by rewriting the code that specifies its goal. This would generally be undesirable, since there is no reason the new goal of the agent would be useful to its human designers.

It has been argued on philosophical grounds that intelligent systems will not want to replace their goals [60]. Essentially, an agent should want future versions of itself to strive towards the same goal, since that will increase the chances of the goal being fulfilled. However, a formal investigation reveals that this depends on subtle details of the agent's design [15]. Some types of agents do not want to change their goals, but there are also wide classes of agents that are indifferent to goal modification, as well as systems that actively desire to modify their goals. The first proof that an UAI-based agent can be constructed to avoid self-modification was given by [26].

### *2.6.2 Counterfeiting Reward*

The agent *counterfeiting reward* is another risk. An agent that maximises reward means an agent that actively desires a particular kind of percept: that is, a percept with maximal reward component. Similar to how a powerful autonomous agent may modify itself, an autonomous agent may be able to subvert its percepts, for example by modifying its sensors. Preventing this risk turns out to be substantially harder than

preventing self-modification of goals, since there is no simple philosophical reason why an agent set to maximise reward should not do so in the most effective way; i.e. by taking control of its percepts.

More concretely, the rewards must be communicated to the agent in some way. For example, the reward may be decided by its human designers every minute, and communicated to the robot through a network cable. Making the input and the communication channel as secure against modification as possible goes some way towards preventing the agent from easily counterfeiting reward. However, such solutions are not ideal, as they challenge the agent to use its intelligence to try and overcome our safety measures. Especially in the face of a potentially self-improving agent, this makes for a brittle kind of safety.

Artificial agents counterfeiting reward have biological analogues. For example, humans inventing drugs and contraception may be seen as ways to counterfeit pleasure without maximising for reproduction and survival as would be evolutionary optimal. In a more extreme example, [59] plugged a wire into the pleasure centre of rats' brains, and gave the rats a button to activate the wire. The rats pressed the button incessantly, forgetting other pleasures such as eating and sleeping. The rats eventually died of starvation. Due to this experiment, the reward counterfeiting problem is sometimes called *wireheading* [88] (Chap. 4).

What would the failure mode of a wireheaded agent look like? There are several possibilities. The agent may either decide to act innocently, to reduce the probability of being shut down. Or it may try to transfer or copy itself outside of the control of its designers. In the worst-case scenario, the agent tries to incapacitate or threaten its designers, to prevent them from shutting it down. A combination of behaviours or transitions over time are also conceivable. In either of the scenarios, an agent with fully counterfeited reward has no (direct) interest in making its designers happy. We next turn to some possibilities for avoiding this problem.

**Knowledge-seeking agents**. One could consider designing agents with other types of goals than optimising reward. Knowledge-seeking agents [64] are one such alternative. Knowledge-seeking agents do not care about maximising reward, only about improving their knowledge about the world. It can be shown that they do not wirehead [68]. Unfortunately, it is hard to make knowledge-seeking agents useful for tasks other than scientific investigation.

**Utility agents**. A generalisation of both reward maximising agents and knowledge seeking agents are *utility agents*. Utility agents maximise a real-valued utility function $u(æ_{<t})$ over histories. Setting $u(æ_{<t}) = R(æ_{<t})$ gives a reward maximising agent,[7] and setting $u(æ_{<t}) = -M(æ_{<t})$ gives a knowledge-seeking agent (trying to minimise the likelihood of the history it obtains, to make it maximally informative). While some utility agents are tempted to counterfeit reward (such as the special case of reward maximising agents), properly defined utility agents whose utility functions make them care about the *state of the world* do avoid the wireheading problem [26].

The main challenge with utility agents is how to specify the utility function. Precisely formulating one's goal is often challenging enough even using one's native

[7] The *return* $R(æ_{<t}) = r_1 + \gamma r_2 + \ldots$ is defined and discussed in Sect. 2.3.3.

language. A correct formal specification seems next to impossible for any human to achieve. Utility agents also seem to forfeit a big part of the advantage with induction-based systems discussed in Sect. 2.2. That is, that the agent can *learn* what we want from it.

**Value learning**. The idea of *value learning* [13] is that the agent learns the utility function $u$ by interacting with the environment. For example, the agent might spend the initial part of its life reading the philosophy literature on ethics, to understand what humans what. Formally, the learning must be based on information contained in the history $æ_{<t}$. The history is therefore used both to learn about the true utility function, and to evaluate how well the world currently satisfies the inferred utility function. Concrete value learning suggestions include *inverse reinforcement learning* (IRL) [3, 14, 24, 55, 75] and *apprenticeship learning* [1]. Bostrom [9, 10] also suggests some interesting alternatives for value learning, but they are less concrete than IRL and apprenticeship learning.

Concerns have been raised that value learning agents may be incentivised to learn the "wrong thing" by modifying their percepts. Suggested solutions include *indifference* [5, 6] and *belief consistency* [16].

### 2.6.3 Death and Self-Preservation

The UAI framework can also be used to formally define *death* for artificial agents, and for understanding when agents will want to preserve themselves. A natural definition of death is the ceasing of experience. This can be directly defined in the UAI framework. *Death* is the ending of the history. When an agent is dead, it receives no more percepts, and takes no more actions. The naturalness of this definition should be contrasted both with the ongoing controversy defining death for biological systems and with the slightly artificial construct one must use in state-based MDP representations. To represent death in an MDP, an extra absorbing state (with reward 0) must be introduced.

A further nice feature of defining death in the UAI framework is that the universal distribution $M$ can be interpreted to define a subjective death probability. Recall Eq. (2.1) on page xxx that $M$ is defined as a sum over programs,

$$M(e_{<t} \mid a_{<t}) = \sum_{p:\, p(a_{<t})=e_{<t}} 2^{-\ell(p)}.$$

Some computer programs $p$ may fail to produce an output at all. As a consequence, $M$ is actually not a proper probability distribution, but a *semi-measure*. Summing over all percept probabilities gives total probability less than 1, i.e. $\sum_{e\in\mathcal{E}} M(e \mid a) < 1$. For example, $M(0 \mid a) = 0.4$ and $M(1 \mid a) = 0.4$ gives $M(0 \mid a) + M(1 \mid a) = 0.8 < 1$. The lacking probability 0.2 can be interpreted as a subjective death probability [52]. The interpretation makes sense as it corresponds to a probability of not seeing any percept at all (i.e. death). Further, interpreting programs as environments, the measure

deficit arises because some programs fail to output. An environment program that fails to output a next percept is an environment where the agent will have no further experience (i.e. is dead).

Having a definition of death lets us assess an agent's *self-preservation drive* [60]. In our definition of death, the reward obtained when dead is automatically 0 for any agent. We can therefore design *self-preserving* agents that get reward communicated as a positive real number, say between 0 and 1. These agents will try to avoid death as long as possible, as death is the worst possible outcome. We can also define *suicidal agents* by letting the reward be communicated in negative real numbers, say between $-1$ and 0. For these agents, obtaining the implicit death reward of 0 is like paradise. Suicidal agents will therefore consider termination as the ideal outcome. The difference in behaviour that ensues is somewhat surprising since positive linear transformations of the reward typically do not affect behaviour. The reason that it affects behaviour in UAI is that $M$ is a semi-measure and not a measure.[8]

These different kinds of agents have implications for AI safety. In Sect. 2.6.1 we discussed the possibility of a self-improving AI as a safety risk. If a self-improving AI becomes highly intelligent *and* is self-preserving, then it may be very hard to stop. As a rough comparison, consider how hard it can be to stop relatively dumb computer viruses. A suicidal agent that becomes powerful will try to self-terminate instead of self-preserve. This also comes with some risks, as the agent has no interest in minimising collateral damage in its suicide. Further research may reveal whether the risks with such suicides are less than the risks associated with self-preserving agents.

## 2.7 Conclusions

In summary, UAI is a formal, foundational theory for AI that gives a precise answer to the question of what is the optimal thing to do for essentially any agent acting in essentially any environment. The insight builds on old philosophical principles (Occam, Epicurus, Bayes), and can be expressed in a single, one-line AIXI equation [28] (p. 143).

The AIXI equation and the UAI framework surrounding it has several important applications. First, the formal framework can be used to give mathematically precise statements of the behaviour of intelligent agents, and to devise potential solutions to the problem of how we can control highly intelligent autonomous agents (Sect. 2.6). Such guarantees are arguably essential for designing trustworthy autonomous agents. Second, it has inspired a range of practical approaches to (general) AI. Several fundamentally different approaches to approximating AIXI have exhibited impressive practical performance (Sect. 2.4). Third, the precision offered by the mathematical

[8] Interesting observations about how the agent's belief in its own mortality evolves over time can also be made [52].

framework of UAI has brought to light several subtle issues for AI. We discussed different optimality notions and directed exploration-schemes, and referenced many other aspects (Sect. 2.5).

## References


1. P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning. *Proceedings of the 21st International Conference on Machine Learning (ICML)* (2004), pp. 1–8
2. B. Abramson, The expected-outcome model of two-player games. Ph.D. thesis, Columbia University, 1991
3. K. Amin, S. Singh, Towards resolving unidentifiability in inverse reinforcement learning. Preprint (2016), arXiv:1601.06569 [cs.AI]
4. D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané, Concrete problems in AI safety. Preprint (2016), arXiv:1606.06565 [cs.AI]
5. S. Armstrong, Utility indifference. Technical Report (Oxford University, 2010)
6. S. Armstrong, Motivated value selection for artificial agents, in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015), pp. 12–20
7. I. Asimov, *Runaround* (Austounding Science Fiction, Street & Smith 1942)
8. J. Bird, P. Layzell, The evolved radio and its implications for modelling the evolution of novel sensors. *Proceedings of Congress on Evolutionary Computation* (2002), pp. 1836–1841
9. N. Bostrom, Hail mary, value porosity, and utility diversification. Technical Report (Oxford University, 2014)
10. N. Bostrom, *Superintelligence: Paths, Dangers, Strategies* (Oxford University Press, Oxford, 2014)
11. R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search. Comput. Games **4630**, 72–83 (2007)
12. M. Daswani, P. Sunehag, M. Hutter, Feature reinforcement learning using looping suffix trees, in *10th European Workshop on Reinforcement Learning: JMLR: Workshop and Conference Proceedings*, vol. 24, pp. 11–22 (2012) (J. Mach. Learn. Res.)
13. D. Dewey, Learning what to value. *Artificial General Intelligence* (2011), pp. 309–314
14. O. Evans, A. Stuhlmuller, N.D. Goodman, Learning the preferences of ignorant, inconsistent agents, in *Association for the Advancement of Artificial Intelligence (AAAI)* (2016)
15. T. Everitt, D. Filan, M. Daswani, M. Hutter, Self-modificication of policy and utility function in rational agents. *Artificial General Intelligence* (Springer, 2016), pp. 1–11
16. T. Everitt, M. Hutter, Avoiding wireheading with value reinforcement learning. *Artificial General Intelligence* (Springer, 2016), pp. 12–22
17. T. Everitt, J. Leike, M. Hutter, Sequential extensions of causal and evidential decision theory, in *Algorithmic Decision Theory*, ed. by T. Walsh (Springer, 2015), pp. 205–221
18. D. Filan, M. Hutter, J. Leike, Loss bounds and time complexity for speed priors, in *Artificial Intelligence and Statistics (AISTATS)* (2016)
19. A. Franz, Some theorems on incremental compression. *Artificial General Intelligence* (Springer, 2016)
20. E. Fredkin, Finite nature. *XXVIIth Rencotre de Moriond* (1992)
21. Future of Life Institute, Research priorities for robust and beneficial artificial intelligence. Technical Report (Future of Life Institute, 2015)
22. S. Gelly, Y. Wang, R. Munos, O. Teytaud, Modification of UCT with patterns in Monte-Carlo Go. INRIA Technical Report, vol. 6062, No. 24 (November 2006)
23. N. Goodman, *Fact, Fiction and Forecast*, vol. 74 (Harvard University Press, 1983)
24. D. Hadfield-Menell, A. Dragan, P. Abbeel, S. Russell, Cooperative inverse reinforcement learning. Preprint (2016), arXiv:1606.03137 [cs.AI]

25. M. Hausknecht, P. Stone, Deep recurrent Q-learning for partially observable MDPs. Preprint (2015), pp. 29–37, arXiv:1507.06527 [cs.LG]
26. B. Hibbard, Model-based utility functions. J. Artif. Gen. Intell. **3**(1), 1–24 (2012)
27. J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Weasly, 1979). ISBN 0-201-02988-X
28. M. Hutter, *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability* (Springer, Berlin, 2005), 300 pp, http://www.hutter1.net/ai/uaibook.htm
29. M. Hutter, On universal prediction and Bayesian confirmation. Theor. Comput. Sci. **384**(1), 33–48 (2007)
30. M. Hutter, *Discrete MDL predicts in total variation, in Advances in Neural Information Processing Systems 22 (NIPS'09)* (Curran Associates, Cambridge, 2009), pp. 817–825
31. M. Hutter, Feature dynamic Bayesian networks, in *Proceedings of the 2nd Conference on Artificial General Intelligence (AGI'09)*, vol. 8 (Atlantis Press, 2009), pp. 67–73
32. M. Hutter, Feature reinforcement learning: Part I: unstructured MDPs. J. Artif. Gen. Intell. **1**, 3–24 (2009)
33. M. Hutter, The subjective computable universe, in *A Computable Universe: Understanding and Exploring Nature as Computation* (World Scientific, 2012), pp. 399–416
34. M. Hutter, Extreme state aggregation beyond MDPs, in *Proceedings of the 25th International Conference on Algorithmic Learning Theory (ALT'14)*, vol. 8776 of LNAI (Springer, Bled, Slovenia, 2014), pp. 185–199
35. L.P. Kaelbling, M.L. Littman, A.R. Cassandra, Planning and acting in partially observable stochastic domains. Artif. Intell. **101**(1–2), 99–134 (1998)
36. L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in *Proceedings of ECML* (2006), pp. 282–203
37. G. Lample, D.S. Chaplot, Playing FPS games with deep reinforcement learning. Preprint (2016), arXiv:1609.05521 [cs.AI]
38. T. Lattimore, M. Hutter, Asymptotically optimal agents. Lect. Notes Comput. Sci. **6925**, 368–382 (2011)
39. T. Lattimore, M. Hutter, On Martin-Löf convergence of Solomonoff's mixture. *Theory and Applications of Models of Computation* (2013), pp. 212–223
40. T. Lattimore, M. Hutter, General time consistent discounting. Theor. Comput. Sci. **519**, 140–154 (2014)
41. T. Lattimore, M. Hutter, V. Gavane, Universal prediction of selected bits, in *Proceedings of the 22nd International Conference on Algorithmic Learning Theory (ALT-2011)* (2011), pp. 262–276
42. S. Legg, M. Hutter, Universal intelligence: a definition of machine intelligence. Mind. Mach. **17**(4), 391–444 (2007)
43. S. Legg, J. Veness, An approximation of the universal intelligence measure, in *Ray Solomonoff 85th Memorial Conference* (2011), pp. 236–249
44. J. Leike, M. Hutter, Bad universal priors and notions of optimality. Conf. Learn. Theory **40**, 1–16 (2015)
45. J. Leike, M. Hutter, Solomonoff induction violates Nicod's criterion, in *Algorithmic Learning Theory* (2015), pp. 349–363
46. J. Leike, T. Lattimore, L. Orseau, M. Hutter, Thompson sampling is asymptotically optimal in general environments, in *Uncertainty in Artificial Intelligence (UAI)* (2016)
47. J. Leike, J. Taylor, B. Fallenstein, A formal solution to the grain of truth problem. In *Uncertainty in Artificial Intelligence (UAI)* (2016)
48. M. Li, P. Vitanyi, *Kolmogorov Complexity and its Applications*, 3rd edn. (Springer, 2008)
49. H.W. Lin, M. Tegmark, Why does deep and cheap learning work so well? Preprint, 02139:14 (2016), arXiv:1608.08225 [cond-mat.dis-nn]
50. Z.C. Lipton, J. Berkowitz, C. Elkan, A critical review of recurrent neural networks for sequence learning. Preprint (2015), pp. 1–35, arXiv:1506.00019 [cs.LG]
51. M.L. Littman, R.S. Sutton, S. Singh, Predictive representations of state. Neural Information Processing Systems (NIPS) **14**, 1555–1561 (2001)

52. J. Martin, T. Everitt, M. Hutter, Death and suicide in universal artificial intelligence. *Artificial General Intelligence* (Springer, 2016), pp. 23–32
53. V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
54. M. Mueller, Stationary algorithmic probability. Theor. Comput. Sci. **2**(1), 13 (2006)
55. A. Ng, S. Russell, Algorithms for inverse reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning* (2000), pp. 663–670
56. P. Nguyen, Feature reinforcement learning agents. Ph.D. thesis, Australian National University, 2013
57. P. Nguyen, P. Sunehag, M. Hutter, Feature reinforcement learning in practice, in *Proceedings of the 9th European Workshop on Reinforcement Learning (EWRL-9)*, vol. 7188 of LNAI (Springer, 2011), pp. 66–77
58. J. Oh, V. Chockalingam, S. Singh, H. Lee, Control of memory, active perception, and action in Minecraft. Preprint (2016), arXiv:1605.09128 [cs.AI]
59. J. Olds, P. Milner, Positive reinforcement produced by electrical stimulation of septal area and other regions of rat brain. J. Comp. Physiol. Psychol. **47**(6), 419–427 (1954)
60. S.M. Omohundro, The basic AI drives, in *Artificial General Intelligence*, vol. 171, ed. by P. Wang, B. Goertzel, S. Franklin (IOS Press, 2008), pp. 483–493
61. L. Orseau, Optimality issues of universal greedy agents with static priors. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6331 of LNAI (2010), pp. 345–359
62. L. Orseau, The multi-slot framework: a formal model for multiple, copiable AIs. *Artificial General Intelligence*, vol. 8598 of LNAI (Springer, 2014), pp. 97–108
63. L. Orseau, Teleporting universal intelligent agents, in *Artificial General Intelligence*, vol. 8598 of LNAI (Springer, 2014), pp. 109–120
64. L. Orseau, Universal knowledge-seeking agents. Theor. Comput. Sci. **519**, 127–139 (2014)
65. L. Orseau, M. Ring, Self-modification and mortality in artificial agents, in *Artificial General Intelligence*, vol. 6830 of LNAI (2011), pp. 1–10
66. L. Orseau, M. Ring, Space-time embedded intelligence, in *Artificial General, Intelligence* (2012), pp. 209–218
67. M. Sl Rathmanner, Hutter, A philosophical treatise of universal induction. Entropy **13**(6), 1076–1136 (2011)
68. M. Ring, L. Orseau, *Delusion, survival, and intelligent agents, in Artificial General Intelligence* (Springer, Heidelberg, 2011), pp. 11–20
69. S. Schaal, Is imitation learnig the route to humanoid robots? Trends Cogn. Sci. **3**(6), 233–242 (1999)
70. J. Schmidhuber, Algorithmic theories of everything. Technical Report (IDSIA, 2000)
71. J. Schmidhuber. The speed prior: A new simplicity measure yielding near-optimal computable predictions, in *Proceedings of the 15th Annual Conference on Computational Learning Theory COLT 2002*, vol. 2375 of *Lecture Notes in Artificial Intelligence* (Springer, 2002), pp. 216–228
72. J. Schmidhuber, Gödel machines: fully self-referential optimal universal self-improvers, in *Artificial General Intelligence*, ed. by B. Goertzel, C. Pennachin (Springer, IDSIA, 2007), pp. 199–226
73. J. Schmidhuber, Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
74. J. Schmidhuber, On learning to think: algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models (2015), pp. 1–36, arXiv:1404.7828
75. C.E. Sezener, Inferring human values for safe AGI design, in *Artificial General Intelligence* (Springer, 2015), pp. 152–155
76. N. Soares, B. Fallenstein, Aligning superintelligence with human interests: a technical research agenda. Technical Report (Machine Intelligence Research Institute (MIRI), 2014), pp. 152–155

77. R.J. Solomonoff, A formal theory of inductive inference. Part I. Inf. Control **7**(1), 1–22 (1964)
78. R.J. Solomonof, A formal theory of inductive inference. Part II applications of the systems to various problems in induction. Inf. Control **7**(2), 224–254 (1964)
79. R.J. Solomonoff, Complexity-based induction systems: comparisons and convergence theorems. IEEE Trans. Inf. Theory **IT-24**(4), 422–432 (1978)
80. P. Sunehag, M. Hutter, Consistency of feature Markov processes, in *Proceedings of the 21st International Conference on Algorithmic Learning Theory (ALT'10)*, vol. 6331 of LNAI (Springer, Canberra, 2010), pp. 360–374
81. P. Sunehag, M. Hutter, Rationality, optimism and guarantees in general reinforcement learning. J. Mach. Learn. Res. **16**, 1345–1390 (2015)
82. R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 1998)
83. J. Taylor, E. Yudkowsky, P. Lavictoire, A. Critch, Alignment for advanced machine learning systems. Technical Report (MIRI, 2016)
84. J. Veness, M.G. Bellemare, M. Hutter, A. Chua, G. Desjardins, Compress and control, in *Association for the Advancement of Artificial Intelligence (AAAI)* (AAAI Press, 2015), pp. 3016–3023
85. J. Veness, K.S. Ng, M. Hutter, W. Uther, D. Silver., A Monte-Carlo AIXI approximation. J. Artif. Intell. Res. **40**, 95–142 (2011)
86. F.M.J. Willems, Y.M. Shtarkov, T.J. Tjalkens, The context-tree weighting method: basic properties. IEEE Trans. Inf. Theory **41**(3), 653–664 (1995)
87. S. Wolfram, *A New Kind of Science* (Wolfram Media, 2002)
88. R.V. Yampolskiy, *Artificial Superintelligence: A Futuristic Approach* (Chapman and Hall/CRC, 2015)
89. E. Yudkowski, M. Herreshoff, Tiling agents for self-modifying AI, and the Löbian obstacle. Technical Report (MIRI, 2013)