# Reflective Variants of Solomonoff Induction and AIXI

**In *Artificial General Intelligence: 8th International Conference, AGI 2015, Berlin, Germany, July 22–25, 2015. Proceedings,* 9205:60–69. Lecture Notes in Artificial Intelligence. Springer International Publishing**

**Benja Fallenstein** and **Nate Soares** and **Jessica Taylor**
Machine Intelligence Research Institute
{benja,nate,jessica}@intelligence.org

## Abstract

Solomonoff induction and AIXI model their environment as an arbitrary Turing machine, but are themselves uncomputable. This fails to capture an essential property of real-world agents, which cannot be more powerful than the environment they are embedded in; for example, AIXI cannot accurately model game-theoretic scenarios in which its opponent is another instance of AIXI.

In this paper, we define *reflective* variants of Solomonoff induction and AIXI, which are able to reason about environments containing other, equally powerful reasoners. To do so, we replace Turing machines by probabilistic oracle machines (stochastic Turing machines with access to an oracle). We then use *reflective oracles*, which answer questions of the form, "is the probability that oracle machine $M$ outputs 1 greater than $p$, when run on this same oracle?" Diagonalization can be avoided by allowing the oracle to answer randomly if this probability is equal to $p$; given this provision, reflective oracles can be shown to exist. We show that reflective Solomonoff induction and AIXI can themselves be implemented as oracle machines with access to a reflective oracle, making it possible for them to model environments that contain reasoners as powerful as themselves.

## 1 Introduction

Legg and Hutter [1] have defined a "Universal measure of intelligence" that describes the ability of a system to maximize rewards across a wide range of diverse environments. This metric is useful when attempting to quantify the cross-domain performance of modern AI systems, but it does not quite capture the induction and interaction problems faced by generally intelligent systems acting in the real world: In the formalism of Legg and Hutter (as in many other agent formalisms) the agent and the environment are assumed to be distinct and separate, while real generally intelligent systems must be able to learn about and manipulate an environment from within.

As noted by Hutter [2], Vallinder [3], and others, neither Solomonoff induction [4] nor AIXI [5] can capture this aspect of reasoning in the real world. Both formalisms require that the reasoner have more computing power than any individual environment hypothesis that the reasoner considers: a Solomonoff inductor predicting according to a distribution over all computable hypotheses is not itself computable; an AIXI acting according to some distribution over environments uses more computing power than any one environment in its distribution. This is also true of computable approximations of AIXI, such as AIXI$^{tl}$. Thus, these formalisms cannot easily be used to make models of reasoners that must reason about an environment which contains the reasoner and/or other, more powerful reasoners. Because these reasoners require more computing power than any environment they hypothesize, environments which contain the reasoner are not in their hypothesis space!

In this paper, we extend the Solomonoff induction formalism and the AIXI formalism into a setting where the agents reason about the environment while embedded within it. We do this by studying variants of Solomonoff induction and AIXI using *probabilistic oracle machines* rather than Turing machines, where a probabilistic oracle machine is a Turing machine that can flip coins and make calls to an oracle. Specifically, we make use of probabilistic oracle machines with access to a "reflective oracle" [6] that answers questions about other probabilistic oracle machines using the same oracle. This allows us to define environments which may contain agents that in turn reason about the environment which contains them.

Section 2 defines reflective oracles. Section 3 gives a definition of Solomonoff induction on probabilistic oracle machines. Section 4 gives a variant of AIXI in this setting. Section 5 discusses these results, along with a number of avenues for future research.

Research supported by the Machine Intelligence Research Institute (intelligence.org). The final publication is available at Springer via `http://dx.doi.org/10.1007/978-3-319-21365-1_7`.

## 2 Reflective Oracles

Our goal is to define agents which are able to reason about environments containing other, equally powerful agents. If agents and environments are simply Turing machines, and two agents try to predict their environments (which contain the other agent) by simply running the corresponding machines, then two agents trying to predict each other will go into an infinite loop.

One might try to solve this problem by defining agents to be Turing machines with access to an oracle, which takes the source code of an oracle machine as input and which outputs what this machine would output when run on the same oracle. (The difference to simply running the machine would be that the oracle would always return an answer, never go into an infinite loop.) Then, instead of predicting the environment by *running* the corresponding oracle machine, agents would query the oracle about this machine. However, it's easy to see that such an oracle cannot exist, for reasons similar to the halting problem: if it existed, then by quining, one could write a program that queries the oracle about *its own* output, and returns 0 iff the oracle says it returns 1, and returns 1 otherwise.

It is possible to get around this problem by allowing the oracle to give random answers in certain, restricted circumstances. To do so, we define agents and environments to be *probabilistic oracle machines*, Turing machines with the ability to act stochastically (by tossing fair coins) and to consult oracles. We consider probabilistic oracle machines to be equipped with advance-only output tapes.

We will write $\mathcal{M}$ for the set of these probabilistic oracle machines. Throughout this paper, an overarrow will be used to denote finite strings, and $\epsilon$ will be used to denote the empty string. Let $\mathbb{B} := \{0, 1\}$ be the set of bits, and $\mathbb{B}^{<\omega}$ denote the set of finite strings of bits. We write $M^O(\overrightarrow{x})$ for a machine $M \in \mathcal{M}$ run on the input $\overrightarrow{x} \in \mathbb{B}^{<\omega}$, using the oracle $O$.

Roughly speaking, a reflective oracle $O$ will answer queries of the form "is the probability that $M^O(\overrightarrow{x})$ outputs 1 greater than $q$?" where $q$ is a rational probability. That is, a query is a triple $(M, \overrightarrow{x}, q) \in \mathcal{M} \times \mathbb{B}^{<\omega} \times \mathbb{Q} \cap [0, 1]$, where $\mathbb{Q} \cap [0, 1]$ is the set of rational probabilities.

More formally, write $\mathbb{P}(M^O(\overrightarrow{x}) = y)$ for the probability that $M^O(\overrightarrow{x})$ outputs at least one bit and that the first bit of output is $y \in \mathbb{B}$. If $M^O(\overrightarrow{x})$ does not always halt before outputting the first bit, then $\mathbb{P}(M^O(\overrightarrow{x}) = 1) + \mathbb{P}(M^O(\overrightarrow{x}) = 0)$ may be less than 1. We assume that the oracle always outputs either 1 or 0, and define distinct calls to the oracle to be stochastically independent (even if they call the oracle on the same query); hence, an oracle's behavior is fully specified by the probabilities $\mathbb{P}(O(M, \overrightarrow{x}, q) = 1)$. Now, we can define reflective oracles as follows:

**Definition** An oracle $O$ is "reflective" if, for all $M \in \mathcal{M}$ and $\overrightarrow{x} \in \mathbb{B}^{<\omega}$, there is some $p \in [0, 1]$ such that

$$\mathbb{P}(M^O(\overrightarrow{x}) = 1) \;\leq\; p \;\leq\; \mathbb{P}(M^O(\overrightarrow{x}) \neq 0)$$

and such that for all $q \in \mathbb{Q} \cap [0, 1]$, the following implications hold:

$$q > p \implies \mathbb{P}(O(M, \overrightarrow{x}, q) = 1) = 1 \tag{1}$$

$$q < p \implies \mathbb{P}(O(M, \overrightarrow{x}, q) = 0) = 1 \tag{2}$$

Note that if $M^O(\overrightarrow{x})$ is guaranteed to output a bit, then $p$ must be exactly the probability $\mathbb{P}(M^O(\overrightarrow{x}) = 1)$ that $M^O(\overrightarrow{x})$ returns 1. If $M^O(\overrightarrow{x})$ sometimes fails to halt, then the oracle can, in a sense, be understood to "redistribute" the probability that the machine goes into an infinite loop between the two possible outputs: it answers queries as if $M^O(\overrightarrow{x})$ outputs 1 with probability p, where $p$ is lower-bounded by the true probability of outputting 1, and upper-bounded by the probability of outputting 1 or looping.

If $q = p$, then $\mathbb{P}(O(M, \overrightarrow{x}, q) = 1)$ may be any number between 0 and 1; this is essential in order to avoid paradox. For example, consider the probabilistic oracle machine which asks the oracle which bit it itself is most likely to output, and outputs the opposite bit. In this case, a reflective oracle may answer 1 with probability 0.5, so that the agent outputs each bit with equal probability. In fact, given this flexibility, a consistent solution always exists.

**Theorem 2.1.** *A reflective oracle exists.*

*Proof.* [6, Appendix A]. □

## 3 Reflective Solomonoff Induction

Using a reflective oracle, it is possible to define a variation on Solomonoff induction defined on probabilistic oracle machines. Define an *environment* to be a probabilistic oracle machine which takes a sequence of bits as input and (probabilistically) produces a single bit of output. We write $\mathbb{B}^{<\omega} \rightsquigarrow \mathbb{B}$ for the type of probabilistic oracle machines run with oracle $O$ which take a finite bit string as input and probabilistically output a single bit. Holding $O$ fixed, one can think of an environment as defining a function of type $\mathbb{B}^{<\omega} \to \Delta(\mathbb{B})$ where $\Delta(\mathbb{B})$ is the set of probability distributions over a single bit. Equivalently, one may see an environment paired with an oracle as a distribution over possibly-infinite bit strings, where strings of bits are generated by running the environment on $\epsilon$ to produce the first bit, and then running it on the first bit to produce the second bit, and then running it on the first two bits to produce the third bit, and so on. What results is a distribution over possibly-infinite bit strings (where the strings may be finite if the environment sometimes goes into an infinite loop rather than producing another output bit).

We will give a variant of Solomonoff induction that predicts observations according to a simplicity distribution over environments, and which is itself a probabilistic oracle machine (implying that it can be embedded into an environment). Roughly speaking, it will take a simplicity distribution, condition it on the observations

seen so far, sample a machine from the resulting distribution, and then use the oracle to output its next bit as if it were that machine. Loosely, this results in a distribution over bits which is 1 according to the probability that a random machine from the updated distribution would next output a 1.

In order to define our variant of Solomonoff induction (and later AIXI) it will be necessary to fix some representation of real numbers. Throughout this paper, real numbers will be represented by infinite sequences of nested closed intervals. To demonstrate, Algorithm 1 describes a probabilistic oracle machine $\mathsf{getProb} : \mathcal{M} \times \mathbb{B}^{<\omega} \times \mathbb{B} \rightsquigarrow \mathbb{R}$ which takes an encoding $M$ of another probabilistic oracle machine, a finite bit string $\overrightarrow{x}$, and a single bit $y$, and uses the oracle $O$ to compute $\mathbb{P}(M^O(\overrightarrow{x}) = y)$. If $M^O(\overrightarrow{x})$ may fail to generate output, $\mathsf{getProb}^O(M, \overrightarrow{x}, 1)$ will return the "redistributed" probability $p$ from Definition 2.

**Algorithm 1:** When run with an oracle $O$, outputs $\mathbb{P}(M^O(\overrightarrow{x}) = y)$ as an infinite sequence of nested intervals.

```
def getProb^O(M, x, y):
    upper ⟵ 1;
    lower ⟵ 0;
    repeat
        middle ⟵ (upper + lower)/2;
        if O(M, x, middle) = y then
        lower ⟵ middle;
        else upper ⟵ middle;
        output (lower, upper);
```

Solomonoff induction on probabilistic oracle machines is given as a function $\mathsf{rSI} : \mathbb{B}^{<\omega} \rightsquigarrow \mathbb{B}$ by Algorithm 2. This function implicitly defines a probability distribution over infinite bitstrings, by providing a way to sample the next bit given the output so far; this allows the conditional probability of the next bit to be computed by getProb. (rSI is defined so that it will always output either 0 or 1, never go into an infinite loop.)

Algorithm 2 makes use of two more helper functions defined in Appendix A, namely $\mathsf{getStringProb} : \mathcal{M} \times \mathbb{B}^{<\omega} \times \mathbb{B}^{<\omega} \rightsquigarrow \mathbb{R}$, which computes the probability that a machine $M$ would output the sequence $\overrightarrow{y}$ conditional on having already outputted $\overrightarrow{x}$, and $\mathsf{flip} : \mathbb{R} \rightsquigarrow \mathbb{B}$ which flips a weighted coin (returning 1 with probability equal to the weight, and 0 otherwise); like getProb, getStringProb uses the "redistributed" probabilities $p$ from Definition 2 if a machine may go into an infinite loop.

With these two helper functions, defining Solomonoff induction on probabilistic oracle machines is straightforward. Using rejection sampling, we sample a machine $M$ with probability proportional to $2^{-\mathsf{len}(M)}\mathsf{getStringProb}^O(M, \epsilon, \overrightarrow{x})$, where $\overrightarrow{x}$ is the string of our observations so far. To do this, we draw $M$ with probability $2^{-\mathsf{len}(M)}$ using the randomMachine function, and then keep it with probability $\mathsf{getStringProb}^O(M, \epsilon, \overrightarrow{x})$. After sampling this machine, we use getProb to sample the next bit in the sequence after our observations.

**Algorithm 2:** Reflective Solomonoff induction for probabilistic oracle machines. It takes a finite bit string and outputs a bit.

```
def rSI^O(x):
    repeat
        M ⟵ randomMachine^O();
        if flip^O(getStringProb^O(M, ε, x)) then
            return flip^O(getProb^O(M, x, 1))
```

Because rSI always terminates, it defines a distribution $\mathsf{rSI} \in \Delta(\mathbb{B}^\omega)$ over infinite bit strings, where $\mathbb{P}_{\mathsf{rSI}}(\overrightarrow{x})$ is the probability that rSI generates the string $\overrightarrow{x}$ (when run on the first $n$ bits to generate the $n+1^{\text{th}}$ bit). This distribution satisfies the essential property of a simplicity distribution, namely, that each environment $M$ is represented somewhere within this distribution.

**Theorem 3.1.** *For each probabilistic oracle machine $M$, there is a constant $C_M$ such that for all finite bit strings $\overrightarrow{x} \in \mathbb{B}^{<\omega}$,*

$$\mathbb{P}_{\mathsf{rSI}}(\overrightarrow{x}) \geq C_M \cdot \mathbb{P}_M(\overrightarrow{x})$$

*where $\mathbb{P}_M(\overrightarrow{x})$ is the probability of $M$ generating the sequence $\overrightarrow{x}$ (when run on the first $n$ bits to generate the $n+1^{th}$ bit).*

*Proof.* First note that

$$\begin{aligned}\mathbb{P}_M(\overrightarrow{x}) &\leq \mathsf{getStringProb}^O(M, \epsilon, \overrightarrow{x}) \\ &= \prod_{i=0}^{\mathsf{len}(\overrightarrow{x})} \mathsf{getProb}^O(M, \overrightarrow{x}_{1:i-1}, \overrightarrow{x}_i),\end{aligned}$$

with equality on the left if $M^O(\overrightarrow{y})$ is guaranteed to produce an output bit for every prefix $\overrightarrow{y}$ of $\overrightarrow{x}$. Then, the result follows from the fact that by construction, sampling a bit string from $\mathsf{rSI}^O$ is equivalent to choosing a random machine $M$ with probability proportional to $2^{-\mathsf{len}(M)}$ and then sampling bits according to $\mathsf{getProb}^O(M, \cdot, \cdot)$.

□

Reflective Solomonoff induction does itself have the type of an environment, and hence is included in the simplicity distribution over environments. Indeed, it is apparent that reflective Solomonoff induction can be used to predict *its own* behavior—resulting in behavior that is heavily dependent upon the choice of reflective oracle and the encoding of machines as bit strings, of course. But more importantly, there are also environments in this distribution which *run Solomonoff induction as a subprocess:* that is, this variant of Solomonoff induction can be used to predict environments that contain Solomonoff inductors.

## 4 Reflective AIXI

With reflective Solomonoff induction in hand, we may now define a reflective *agent*, by giving a variant of AIXI that runs on probabilistic oracle machines. To do this, we fix a finite set $\mathcal{O}$ of observations, together with a prefix-free encoding of observations as bit strings. Moreover, we fix a function $r : \mathcal{O} \to [0,1]$ which associates to each $o \in \mathcal{O}$ a (computable) reward $r(o)$. Without loss of generality, we assume that the agent has only two available actions, 0 and 1.

Reflective AIXI will assume that an environment is a probabilistic oracle machine which takes a finite string of observation/action pairs and produces a new observation; that is, an environment is a machine with type $(\mathcal{O} \times \mathbb{B})^{<\omega} \rightsquigarrow \mathcal{O}$. Reflective AIXI assumes that it gets to choose each action bit, and, given a history $\overrightarrow{oa} \in (\mathcal{O}\times\mathbb{B})^{<\omega}$ and the latest observation $o \in \mathcal{O}$, it outputs the bit which gives it the highest expected (time-discounted) future reward. We will write $r_t(\overrightarrow{oa}) := r(\mathsf{fst}(\overrightarrow{oa}_t))$ for the reward in the $t^{\text{th}}$ observation of $\overrightarrow{oa}$.

To define reflective AIXI, we first need the function step from Algorithm 3, which encodes the assumption that an environment can be factored into a world-part and an agent-part, one of which produces the observations and the other which produces the actions.

**Algorithm 3:** Takes an agent and an environment and the history so far, and computes the next observation/action pair.

**def** $\mathsf{step}^O(world, agent, \overrightarrow{oa})$**:**
  $o \longleftarrow world^O(\overrightarrow{oa})$;
  $a \longleftarrow agent^O(\overrightarrow{oa}, o)$;
  **return** $(o, a)$

Next, we need the function reward from Algorithm 4, which computes the total discounted reward given a world (selecting the observations), an agent (assumed to control the actions), and the history so far. Total reward is computed using an exponential discount factor $0 < \gamma < 1$. We multiply by $1-\gamma$ to make total reward sum to a number between 0 and 1. With this rescaling, total discounted reward starting from step $t$ is no more than $(1-\gamma)\sum_{s=t}^{\infty} \gamma^{s-1} = \gamma^{t-1}$.

**Algorithm 4:** The distribution over real numbers defined by this probabilistic machine is the distribution of the future discounted reward of *agent* interacting with *world*, given that the history $\overrightarrow{oa}$ has already occurred.

**def** $\mathsf{reward}^O(world, agent, \overrightarrow{oa})$**:**
  **for** $n = 1, 2, \ldots$ **do**
    $\overrightarrow{oa} \longleftarrow$
    $\mathsf{append}\left(\overrightarrow{oa}, \mathsf{step}^O(world, agent, \overrightarrow{oa})\right)$;
    $seen \longleftarrow (1-\gamma)\sum_{t=1}^{n} \gamma^{t-1} \cdot r_t(\overrightarrow{oa})$;
    **output** $(seen, seen + \gamma^n)$;

With reward in hand, an agent which achieves the maximum expected (discounted) reward in a given environment $\mu$, can be defined as in $\mathsf{rAI}_\mu$. Algorithm 5 defines a machine $\mathsf{actionReward}^O(a)$, which computes the reward if the agent takes action $a$ in the next timestep and in future timesteps behaves like the optimal agent $\mathsf{rAI}_\mu$. It then defines a machine $\mathsf{difference}^O()$, which computes the difference in the discounted rewards when taking action 1 and when taking action 0, then rescales this difference to the interval $[0,1]$ and flips a coin with the resulting probability. Finally, $\mathsf{rAI}_\mu$ uses the oracle to determine whether the probability that $\mathsf{difference}^O() = 1$ is greater than $1/2$, which is equivalent to asking whether the expectation of $\mathsf{actionReward}^O(1)$ is greater than the expectation of $\mathsf{actionReward}^O(0)$; if the expectations are equal, the oracle may behave randomly, but this is acceptable, since in this case the agent is indifferent between its two actions. Note that Algorithm 5 references its own source code (actionReward passes the source of $\mathsf{rAI}_\mu$ to reward); this is possible by quining (Kleene's second recursion theorem).

**Algorithm 5:** Reflective $\mathrm{AI}_\mu$.

**def** $\mathsf{rAI}_\mu^O(\overrightarrow{oa}, o)$**:**
  **def** $\mathsf{actionReward}^O(a)$**:**
    **return** $\mathsf{reward}^O\left(\mu, \mathsf{rAI}_\mu, \mathsf{append}\left(\overrightarrow{oa}, (o, a)\right)\right)$
  **def** $\mathsf{difference}^O()$**:**
    **return**
    $\mathsf{flip}^O\left(\dfrac{\mathsf{actionReward}^O(1) - \mathsf{actionReward}^O(0) + 1}{2}\right)$
  **return** $O\left(\mathsf{difference}, \epsilon, 1/2\right)$;

We can now obtain a reflective version of AIXI by instantiating the environment $\mu$ in $\mathsf{rAI}_\mu$ to a universal environment $\xi$, which (in analogy with Solomonoff induction) selects a random environment and then behaves like this environment. As in our implementation of Solomonoff induction, we use rejection sampling, sampling a random machine $M$ and keeping it with probability $\mathsf{getHistProb}^O(M, \overrightarrow{oa}, o')$, which computes the probability that environment $M$ will produce an observation starting with prefix $o'$ given the previous history $\overrightarrow{oa}$ (Algorithm 8). $\xi$ will find the next bit of the next observation after any $\overrightarrow{oa}$ sequence followed by a prefix $o'$ of the next observation.

**Algorithm 6:** A variant of reflective Solomonoff induction used by reflective AIXI. It takes a series of observation/action pairs and updates its simplicity distribution according to the likelihood that an environment produced the observations in this sequence (holding the actions fixed).

**def** $\xi^O(\overrightarrow{x})$**:**
  split $\overrightarrow{x}$ into a sequence $\overrightarrow{oa}$ of observations and actions and a prefix $o'$ of the next observation;
  **repeat**
    $M \longleftarrow \mathsf{randomMachine}^O()$;
    **if** $\mathsf{flip}^O(\mathsf{getHistProb}^O(M, \overrightarrow{oa}, o'))$ **then**
    **return** $\mathsf{flip}(\mathsf{getProb}(M, \overrightarrow{oa}o', 1))$;

**def** $\mathsf{rAIXI}^O(\overrightarrow{oa}, o)$**:**
  **return** $\mathsf{rAI}^O_\xi(\overrightarrow{oa}, o))$

## 5 Conclusions

Our model of agents interacting with an environment is quite reminiscent of classical game theory, in which all agents are assumed to be logically omniscient: indeed, reflective oracles can be used to provide new foundations for classical game theory in which the agents are not ontologically distinct from the rest of the game, but rather are ordinary features of the environment [6].

Realistic models of artificial reasoners must dispense with this guarantee of logical omniscience, and consider agents that reason under logical uncertainty. Even reasoners that have perfect knowledge about other agents (for example, reasoners which possess the source code of a different, deterministic agent) may not be able to deduce exactly how that agent will behave, due to computational limitations. Such limitations are not captured by models of reflective AIXI.

Nevertheless, we expect that studying the behavior of powerful reasoners in reflective environments will shed some light on how powerful bounded reasoners can perform well in more realistic settings. These reflective environments provide the beginnings of a suite of tools for studying agents that can reason about the environment in which they are embedded, and which can reason about universes which contain other agents of similar capabilities.

It is our hope that, through studying this simple model of reflective agents, it will be possible to gain insights into methods that agents can use to learn the environment which embeds them (as discussed by Soares [7]), while reasoning well in the presence of agents which are as powerful or more powerful than the reasoner (as discussed by Fallenstein and Soares [8]). For example, these reflective versions of Solomonoff induction and AIXI open up the possibility of studying agents in settings where the agent/environment boundary breaks down (as discussed by Orseau and Ring [9]), or agents in settings containing other similarly powerful agents. A first step in this direction is suggested by a result of [6], which shows that it is possible to define a computable version of reflective oracles, defined only on the set of probabilistic oracles machines whose length is $\leq l$ and which are guaranteed to halt within a time bound $t$; this appears to be exactly what is needed to translate our reflective variant of AIXI into a reflective, computable variant of AIXI$^{tl}$.

## References


[1] Shane Legg and Marcus Hutter. "Universal Intelligence. A Definition of Machine Intelligence". In: *Minds and Machines* 17.4 (2007), pp. 391–444. DOI: 10.1007/s11023-007-9079-x.

[2] Marcus Hutter. "Open Problems in Universal Induction & Intelligence". In: *Algorithms* 2.3 (2009), pp. 879–906. DOI: 10.3390/a2030879.

[3] Aron Vallinder. "Solomonoff Induction: A Solution to the Problem of the Priors?" MA thesis. Lund University, 2012. URL: http://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=3577211&fileOId=3577215.

[4] Ray J. Solomonoff. "A Formal Theory of Inductive Inference. Part I". In: *Information and Control* 7.1 (1964), pp. 1–22. DOI: 10.1016/S0019-9958(64)90223-2.

[5] Marcus Hutter. "Universal Algorithmic Intelligence. A Mathematical Top→Down Approach". In: *Artificial General Intelligence*. Ed. by Ben Goertzel and Cassio Pennachin. Cognitive Technologies. Berlin: Springer, 2007, pp. 227–290. DOI: 10.1007/978-3-540-68677-4_8.

[6] Benja Fallenstein, Jessica Taylor, and Paul Christiano. *Reflective Oracles: A Foundation for Classical Game Theory*. Tech. rep. 2015–7. Berkeley, CA: Machine Intelligence Research Institute, 2015. URL: http://intelligence.org/files/ReflectiveOracles.pdf.

[7] Nate Soares. *Formalizing Two Problems of Realistic World-Models*. Tech. rep. 2015–3. Berkeley, CA: Machine Intelligence Research Institute, 2015. URL: https://intelligence.org/files/RealisticWorldModels.pdf.

[8] Benja Fallenstein and Nate Soares. *Vingean Reflection. Reliable Reasoning for Self-Improving Agents*. Tech. rep. 2015–2. Berkeley, CA: Machine Intelligence Research Institute, 2015. URL: https://intelligence.org/files/VingeanReflection.pdf.

[9] Laurent Orseau and Mark Ring. "Space-Time Embedded Intelligence". In: *Artificial General Intelligence. 5th International Conference, AGI 2012, Oxford, UK, December 8–11, 2012. Proceedings*. Ed. by Joscha Bach, Ben Goertzel, and Matthew Iklé. Lecture Notes in Artificial Intelligence 7716. New York: Springer, 2012, pp. 209–218. DOI: 10.1007/978-3-642-35506-6_22.

# APPENDIX

## A Helper functions

**Algorithm 7:** Computes the probability that machine $M$ outputs $\overrightarrow{y}$ conditional on it already outputting $\overrightarrow{x}$, as a real number represented by an infinite sequence of nested intervals..

**def** $\mathsf{getStringProb}^O(M, \overrightarrow{x}, \overrightarrow{y})$**:**
  **return** $\prod_{i=1}^{\mathsf{len}(\overrightarrow{y})} \mathsf{getProb}^O(M, \overrightarrow{x}\,\overrightarrow{y}_{1:i-1}, \overrightarrow{y}_i)$

**Algorithm 8:** Computes the probability that $M$ would output the observations in $\overrightarrow{oa}$ and the additional observation prefix $o'$, given that the agent responds with the actions in $\overrightarrow{oa}$.

**def** $\mathsf{getHistProb}^O(M, \overrightarrow{oa}, o')$**:**
  **return**
  $\left(\prod_{i=1}^{\mathsf{len}(\overrightarrow{oa})} \mathsf{getStringProb}^O(M, \overrightarrow{oa}_{1:i-1}, \mathsf{fst}(\overrightarrow{oa}_i))\right) \cdot$
  $\mathsf{getStringProb}^O(M, \overrightarrow{oa}, o'))$;

**Algorithm 9:** Generates a random machine $M$ with probability $2^{-\mathsf{len}(M)}$.

**def** $\mathsf{randomMachine}^O()$**:**
  *prefix* $\longleftarrow \epsilon$;
  **repeat**
    **if** *prefix* is a valid machine **then return** *prefix*;
    **else** *prefix* $\longleftarrow \mathsf{append}(prefix, \mathsf{tossCoin}())$;

**Algorithm 10:** Outputs 1 with probability *weight*, 0 otherwise.

**def** $\mathsf{flip}^O(weight)$**:**
  *upper* $\longleftarrow 1$;
  *lower* $\longleftarrow 0$;
  **for** $(l, u)$ **in** *weight* **do**
    *middle* $\longleftarrow {}^{(upper\,+\,lower)}/_2$;
    **if** $\mathsf{tossCoin}() = 1$ **then** *upper* $\longleftarrow$ *middle*;
    **else** *lower* $\longleftarrow$ *middle*;
    **if** *upper* $< l$ **then return** 1;
    **else if** *lower* $> u$ **then return** 0;