Journal of Artificial General Intelligence 4(1) 31–48, 2013
DOI: 10.2478/jagi-2013-0003

Submitted 2012-06-30
Accepted 2013-11-25

A Measure of Real-Time Intelligence

Vaibhav Gavane VAIBHAV.GAVANE@GMAIL.COM
School of Computing Science and Engineering
VIT University
Vellore - 632014, Tamil Nadu, India

Editor: Jürgen Schmidhuber

Abstract

We propose a new measure of intelligence for general reinforcement learning agents, based on the notion that an agent's environment can change at any step of execution of the agent. That is, an agent is considered to be interacting with its environment in real-time. In this sense, the resulting intelligence measure is more general than the universal intelligence measure (Legg and Hutter, 2007) and the anytime universal intelligence test (Hernández-Orallo and Dowe, 2010). A major advantage of the measure is that an agent's computational complexity is factored into the measure in a natural manner. We show that there exist agents with intelligence arbitrarily close to the theoretical maximum, and that the intelligence of agents depends on their parallel processing capability. We thus believe that the measure can provide a better evaluation of agents and guidance for building practical agents with high intelligence.

Keywords: intelligence measure, reinforcement learning, real-time computation, algorithmic probability

1. Introduction

Reinforcement learning is a general framework in which a wide variety of problems in artificial intelligence can be expressed, including pattern recognition, classification, optimization, and game playing (Hutter, 2005). All these problems can be expressed within the framework as environments that interact with agents and reward them according to the correctness or optimality of their actions. Given the expressive power of the framework, it is of interest to find agents that can perform well in a wide class of environments. This requires that a reasonable performance or intelligence measure for agents be established, both to evaluate existing agents and to provide guidance for building more intelligent agents.

Legg and Hutter (2007) proposed the *universal intelligence measure*, which measures the intelligence of an agent by the expected value of its policy in all reward summable computable environments with respect to an algorithmic probability distribution (Li and Vitányi, 2008, §4.3.3). However, the model of agent-environment interactions is such that agents are allowed to take any amount of time to complete an action in a cycle, and the environments are also completely unaffected by the response times of agents. Consequently, the universal intelligence of agents is independent of their computational complexities. According to Legg and Hutter (2007), an intelligence measure should be defined this way in order to maintain its applicability to (hypothetical) agents with incomputable policies, but it is conceded that the efficiency of agents is important.

Hernández-Orallo and Dowe (2010) considered response times to be a significant factor in a test of intelligence for agents, and proposed a modification of the universal intelligence measure that (apart from other changes) adjusts the intelligence of an agent according to its response times. In a finite testing period in which an agent's interactions with an environment are simulated, the actual number of interactions that are used to calculate the value of the agent in the environment is proportional to the time from the beginning until the last completed action in that period. However, any kind of explicit adjustment of the value introduces an arbitrariness in the resulting intelligence measure. The problem is that the response-times-dependent performance of an agent is not properly reflected in the intelligence test, since the simulated environments remain unaware of the response times of agents, with the result that the perceptions of an agent are still independent of its response times.

In this paper, we consider a simple and natural way to factor the effect of response times into an intelligence measure. We consider every agent as interacting with its environment in real-time. More precisely, each possible output (including null, if necessary) at a step of execution of an agent (where "step" depends on the machine model of the agent) is interpreted as an action of the agent, to which the environment responds. Similarly, each input available at a step of execution of the agent is a perception, which can be lost if not read by the agent. This is obviously the case for agents in the form of robots with sensors for input and actuators for output. However, it is applicable to any agent that interacts even indirectly with the universe, such as an agent that only trades in a financial market. Like Legg and Hutter (2007), we assume that the universe is (probabilistically) computable, and that the prior probability of a computable environment is given by an algorithmic probability distribution. Consequently, at a "high" level, the intelligence measure developed in this paper resembles the universal intelligence measure, but since the agents are interpreted at their machine level in the former, overall the two measures are very different. We show that there exist agents with intelligence arbitrarily close to the theoretical maximum, and that the intelligence of agents depends on their parallel processing capability. Thus, we expect this measure to be more useful for evaluating and constructing practical agents.

The rest of this paper is organized as follows. Section 2 describes general reinforcement learning agents and optimal policies for arbitrary environments. Section 3 describes algorithmic probability distributions and their suitability for weighing computable environments. In Section 4 we develop a measure of real-time intelligence using a variant of Turing machines and compare it with the universal intelligence measure and the anytime universal intelligence test. In Section 5 we prove theoretical results using the developed measure. Section 6 concludes the paper.

2. General Reinforcement Learning Agents

A general reinforcement learning agent is an information processing system. The inputs to an agent are called *perceptions*, and its outputs are called *actions*. Each perception consists of an *observation* and a *reward*. In each *cycle*, an agent receives a perception from its *environment*, processes it, and performs an action. The input-output behavior of an agent is called its *policy*. The *value* of an agent's policy in an environment is the expected total reward that the agent receives from the environment. The policy of an agent, and its environment may depend on the complete history of interactions between the agent and environment. In general, both an agent and its environment can be probabilistic, but for convenience we consider only deterministic agents in this paper.

We now give formal definitions, mostly following Hutter (2005) and Legg and Hutter (2007). However, we defer a formal definition of agents until Section 4, and we also do not give a computable basis for environments until Section 3. We do this because it is useful to define arbitrary (in)computable policies and environments for mathematical analysis.

Definition 1 (Sequences) *For a nonempty finite set $\mathcal{B}$, the sets $\mathcal{B}^n$ and $\mathcal{B}^*$ denote the sets of all n-length and finite sequences over $\mathcal{B}$. ϵ denotes the empty sequence. For an $x \in \mathcal{B}^*$, x_n is the nth symbol/word of the sequence, $x_{1:n} := x_1 \ldots x_n$, and $x_{<n} := x_{1:n-1}$. $\ell(x)$ denotes the length of x. If $n > \ell(x)$ or $n < 1$ then $x_n := \epsilon$.*

Definition 2 (Perception/Action Space) *A perception space $\mathcal{P}$ is a nonempty finite set such that $\mathcal{P} = \mathcal{O} \times \mathcal{R}$ and $\mathcal{R} \subset \mathbb{Q} \cap [0,1]$, where $\mathcal{O}$ is the observation space and $\mathcal{R}$ is the reward space. For a perception $x \in \mathcal{P}$, $o(x)$ and $r(x)$ denote its observation and reward components. An action space $\mathcal{A}$ is any nonempty finite set.*

We have restricted the perception/action space to be finite since we assume that the universe is discrete, which implies that in a fixed length of time an agent can receive/send only a finite amount of information. We will usually assume without explicitly stating that $|\mathcal{O}|, |\mathcal{R}|, |\mathcal{A}| \geq 2$, which is required for non-trivial results. Also, for convenience we will assume that $\max \mathcal{R} = 1$ and $\min \mathcal{R} = 0$.

Consider the interaction of an environment and an agent with perception space $\mathcal{P}$ and action space $\mathcal{A}$. Let $x_n \in \mathcal{P}$ be the perception generated by the environment in the nth cycle, and let $y_n \in \mathcal{A}$ be the action of the agent. That is, in the nth cycle, first the environment generates x_n based on the interaction history $x_1 y_1 \ldots x_{n-1} y_{n-1}$, and then the agent performs the action y_n based on the history $x_1 y_1 \ldots x_{n-1} y_{n-1} x_n$.

Definition 3 (Policy) *The policy of an agent with perception space $\mathcal{P}$ and action space $\mathcal{A}$ is a function $p : \mathcal{P}^* \to \mathcal{A}^*$ such that $\forall (n \in \mathbb{N})\ \forall (x \in \mathcal{P}^n)\ [p(x)_{<n} = p(x_{<n})]$. Then, for a sequence of perceptions $x_1 x_2 x_3 \ldots$, the nth action of the agent is $y_n := p(x)_n \equiv p(x_{1:n})_n$. Since we are considering only deterministic agents, it is not necessary to include the actions of the agent in the domain of its policy.*

We need to define probabilistic environments more generally in terms of probability semimeasures (Li and Vitányi, 2008, §4.3); this is necessary for environments defined using algorithmic probability distributions.

Definition 4 (Probability Semimeasures) *A function $\mu : \mathcal{B}^* \to [0,1]$ is a continuous probability semimeasure if $\mu(\epsilon) \leq 1$ and $\forall (x \in \mathcal{B}^*)\ [\sum_{b \in \mathcal{B}} \mu(b \mid x) \leq 1]$; if equality holds, then μ is a continuous probability measure. A function $P : \mathcal{B}^* \to [0,1]$ is a discrete probability semimeasure if $\sum_{x \in \mathcal{B}^*} P(x) \leq 1$; if equality holds then P is a discrete probability measure.*

Definition 5 (Environment) *An environment is a conditional probability semimeasure $\mu : (\mathcal{P} \times \mathcal{A})^* \to [0,1]$ such that $\forall (n \in \mathbb{N})\ \forall (x \in \mathcal{P}^n)\ \forall (y \in \mathcal{A}^n)\ [\mu(x_1 y_1 \ldots x_n y_n) = \mu(x \mid y) = \mu(x \mid y_{<n}) \geq \sum_{b \in \mathcal{P}} \mu(xb \mid y)]$. Such a semimeasure is called* chronological *(Hutter, 2005). To avoid clutter, we have not defined the domain of environments as $\{\epsilon\} \cup \mathcal{P} \times (\mathcal{A} \times \mathcal{P})^*$, but specified $\mu(x \mid y) = \mu(x \mid y_{<n})$ in the condition to indicate that the environment generates the perception first in a cycle. An environment μ is deterministic if $\forall (y \in \mathcal{A}^*)\ \forall (x \in \mathcal{P}^{\ell(y)+1})\ [\mu(x \mid y) = 1$ or $\mu(x \mid y) = 0]$.*

The (total) value of an agent's policy in an environment μ is the μ-expected sum of rewards that the agent receives over an infinite number of interaction cycles, which may not exist for arbitrary environments. Legg and Hutter (2007) use only reward summable environments to define the universal intelligence measure, that is, those environments in which the value of every policy is finite. However, due to the finiteness of the perception space and hence the reward space, every deterministic reward summable environment generates a nonzero reward only for a finite number of cycles for any policy, that is, every agent eventually becomes completely useless. While there exist probabilistic environments for which this is not the case, the implications for deterministic environments are more important (we will justify this in Section 4). We thus take the alternative approach of discounting. In order to obtain a finite value for every environment and policy, we discount the rewards using a summable function. The drawback of discounting is that it introduces an arbitrariness in the resulting intelligence measure.

Definition 6 (Discounted Value) *For a function $\gamma : \mathbb{N} \to (0, 1]$ such that $\sum_{j=1}^{\infty} \gamma(j) \leq 1$, the γ-discounted value of an agent's policy $p : \mathcal{P}^* \to \mathcal{A}^*$ in an environment $\mu : (\mathcal{P} \times \mathcal{A})^* \to [0, 1]$ for m cycles is defined as*

$$V_{m\gamma}^{p\mu} := \sum_{x \in \mathcal{P}^m} \mu(x \mid p(x_{<m})) \sum_{j=1}^{m} \gamma(j) r(x_j).$$

Since the reward space $\mathcal{R} \subset [0, 1]$ and $\gamma(j) > 0$, $V_{m\gamma}^{p\mu}$ is monotonically increasing in m. And since $\sum_{j=1}^{\infty} \gamma(j) \leq 1$, the total value $V_{\gamma}^{p\mu} := \lim_{m\to\infty} V_{m\gamma}^{p\mu} \leq 1$.

For each environment μ, and discount function γ, there exists an *optimal* policy p_{γ}^{μ} such that for any policy p, $V_{\gamma}^{p_{\gamma}^{\mu}\mu} \geq V_{\gamma}^{p\mu}$. In fact, deterministic optimal policies are also optimal among all probabilistic policies (Hutter, 2005, prob. 4.2 and p. 160). A formal proof of existence of optimal policies is given by Lattimore and Hutter (2011, thm. 8).

Definition 7 (Optimal Policy) *For an environment $\mu : (\mathcal{P} \times \mathcal{A})^* \to [0, 1]$, and discount function $\gamma : \mathbb{N} \to [0, 1]$, the optimal policy $p_{\gamma}^{\mu} : \mathcal{P}^* \to \mathcal{A}^*$ is defined as $p_{\gamma}^{\mu} := \operatorname{argmax}_p V_{\gamma}^{p\mu}$.*

For each policy p, and discount function γ, there exists an environment μ such that $p = p_{\gamma}^{\mu}$ (see Hutter, 2005, §3.3.1). An agent with policy p_{γ}^{μ} interacting with any environment ρ selects an action in each cycle that is expected to maximize the future rewards with respect to μ, conditioned with the history of interactions between the agent and ρ. If ρ is different from μ then the agent may not perform well in ρ. Indeed, for each environment μ there exists an environment ρ such that $V_{\gamma}^{p_{\gamma}^{\mu}\rho} = 0$ and $V_{\gamma}^{p_{\gamma}^{\rho}\rho} > 0$ (Hutter, 2005, §5.3.2).

3. Algorithmic Probability Distributions

A general intelligence measure for agents needs to consider the performance of agents in a wide range of environments. The set of all computable environments is sufficiently large to contain models from all valid theories of the universe (Hutter, 2005, §3.2.9) and thus it is reasonable to consider this set. Since there are an infinite number of computable environments, the sum of values of a policy in all computable environments may be infinite. Hence, analogous to discounting, the value of a policy in each environment needs to be weighed according to a summable function. Like the discount function, the weight function also introduces a parameter in the intelligence measure. However, the choice of algorithmic probability distributions can be justified for weighing environments on the basis of their invariance (within a multiplicative constant) under reparametrization and regrouping (Hutter, 2007).

Algorithmic probability can be viewed as a formalization of Occam's razor: the assumption that the simplest hypothesis is most likely to be true. Note that this does not mean discarding the hypotheses that are not the simplest; we can still keep all consistent hypotheses in accordance with Epicurus's principle, but we consider complex hypotheses less likely. So far, the empirical evidence suggests that the universe is simple. The fundamental theories of the universe consistent with the evidence, viz. general relativity and quantum mechanics, consist of simple equations. However, it is always possible that the universe is more complex than it seems and that the current theories may not hold in the future.

The notion of complexity is closely related to that of information. In general, it requires more words to describe a complex object than a simple one. However, since the description of objects depends on the language that is used, complexity cannot be an absolute property. Nevertheless, *universal languages*, that is, languages of universal Turing machines, yield complexity measures that are equivalent and optimal up to an additive constant. Formally, a complexity measure is defined for sequences. For this reason, we do not consider the complexity of environments (that is, functions) directly, rather we consider the complexity of encodings of Turing machines (or equivalently, encodings of indices in a recursive enumeration of all Turing machines) that approximate the environments.

In this section, we mostly follow the terminology and definitions of Li and Vitányi (2008).

Definition 8 (Approximable Functions) *A function $f : \mathcal{B}^* \to \mathbb{R}$ is approximable if there exists a total recursive function $\phi : \mathcal{B}^* \times \mathbb{N} \to \mathbb{Q}$ such that $\forall(x \in \mathcal{B}^*)$ $[\lim_{n\to\infty} \phi(x,n) = f(x)]$. An approximable function f is lower (resp. upper) semicomputable if there exists a total recursive approximator ϕ for it that is also monotonically increasing (resp. decreasing) in n. A function f is computable if it is both lower and upper semicomputable.*

The main difference between a function that is computable and a function that is only approximable (or semicomputable) is that for the latter an upper bound on $|f(x) - \phi(x,n)|$ cannot be found in general.

For each perception space $\mathcal{P}$ and action space $\mathcal{A}$, the set of all lower semicomputable environments is recursively enumerable (Hutter, 2005, §5.10). However, the set of all computable environments is not recursively enumerable (Li and Vitányi, 2008, lemma 4.5.2). Since we would like the intelligence measure to be at least approximable, we use the larger class of lower semicomputable environments for defining it.

We need to define a complexity measure using a variant of Turing machines; this is necessary for algorithmic probability (see Li and Vitányi, 2008, p. 198).

Definition 9 (Prefix Machines) *A prefix machine is a Turing machine with a unidirectional input tape, a unidirectional output tape and a bidirectional work tape. For convenience, we assume that the input/output tape alphabet is binary. For a prefix machine T with an input sequence beginning with p, if T halts after having read exactly p and written a sequence x, then this is denoted by $T(p) = x$. This can be interpreted as: p is a description in the language of T for the sequence x.*

A prefix machine U is universal if for each prefix machine T there exists a sequence q such that for all p, $U(qp) = T(p)$. That is, U first reads the encoding q of a prefix machine T from its input tape, and thereafter simulates T with the rest of the input. Hence, for every description p in the language of T, qp is a description in the language of U.

Definition 10 (Prefix Complexity) *The prefix complexity of a sequence x with respect to a prefix machine T is defined as $K_T(x) := \min\{\ell(p) : T(p) = x\}$. In general, K_T is an upper semicomputable function.*

For a universal prefix machine U and a prefix machine T with encoding q in the language of U, $K_U(x) \leq \ell(q) + K_T(x)$. That is, U requires at most $\ell(q)$ more bits than T to describe any sequence describable by T. Hence, for any two universal prefix machines U_1 and U_2, there exist constants c_1 and c_2 such that $c_1 \leq K_{U_1}(x) - K_{U_2}(x) \leq c_2$.

Definition 11 (Algorithmic Probability) *The algorithmic probability of a sequence x with respect to a universal prefix machine U is defined as $R_U(x) := 2^{-K_U(x)}$. For all U, R_U is only lower semicomputable. Also, R_U is a discrete probability semimeasure but not a probability measure.*

By definition, a sequence with a low prefix complexity has a high algorithmic probability. Although the probability does depend on the choice of the universal prefix machine, from the additive optimality of prefix complexity, it follows that for any two universal prefix machines U_1 and U_2, there exist constants c_1 and c_2 such that $c_1 \leq R_{U_1}(x)/R_{U_2}(x) \leq c_2$.

Let $T_0, T_1, T_2, \ldots$ be an enumeration of all Turing machines, representing approximators of all lower semicomputable environments $\mu_0, \mu_1, \mu_2, \ldots$ (with repetition). Let $\langle\cdot\rangle : \mathbb{N} \to \{0,1\}^*$ be a recursive encoding of numbers as binary sequences. Then, the algorithmic probability of an environment μ_i with respect to a universal prefix machine U is given by $R_U(\langle i \rangle)$. Note that the algorithmic probability of μ_i changes by only a constant multiplicative factor for any other universal prefix machine or recursive encoding scheme.

4. Real-Time Intelligence

We now develop a measure of real-time intelligence. The central idea is to interpret each step of execution of an agent as a cycle of the agent. What an agent can do in one step depends on its machine model. Accordingly, the intelligence measure is built up in two layers: the machine-dependent lower layer interprets agents one step per interaction cycle, and the machine-independent upper layer simulates environments and drives the overall interactions.

The machine model of an agent is required to be synchronous, so that each step of the agent is executed in a fixed physical time period. This time period is then interpreted as the sampling period of the perceptions and actions of the agent. Therefore, the real-time intelligence of two agents can be meaningfully compared only if their time periods are the same, although their machine models need not be the same.

In an ideal case, the given machine model is already synchronous: each machine has a clock that oscillates at a constant time period, and one step of the machine is performed in one cycle of the clock. Then, we can simply take the time period of its machine as the time period of the agent. The specification of a synchronous machine model need not include any particular value(s) for the time period. A given machine of a machine model may be runnable at different time periods by tweaking its clock. However, different values of time period for the same machine instantiate different agents.

In practice, machine models are often asynchronous. These models include, but are not limited to, those with a dynamic clock, multiple clocks, or no clock at all. The concept of a (global) clock and/or step may not exist in such models, but in our approach these concepts are necessary to evaluate an agent's real-time intelligence. Therefore, we need to "wrap" an asynchronous machine inside a synchronizer (Rabaey, Chandrakasan, and Nikolic, 2003, §10.5). In this case, a "step" is defined as the operation performed by the overall machine in one time period of the clock of the synchronizer. A synchronizer may be made up of multiple physical units but all units should be synchronized by the same clock.

In the rest of the paper, we consider in detail, agents that are modeled by synchronous multitape Turing machines similar to those defined by Rosenberg (1967), with a fixed time period $\mathcal{T}$. However, the results that are obtained are not really specific to such agents; analogous results hold for multihead Turing machines (Vitányi, 1980). For convenience and clarity, we consider only deterministic agents. The generalization to probabilistic agents is straightforward by defining them in terms of probabilistic Turing machines.

We first provide an informal description of the machine model, called *transducer*. A transducer t is a Turing machine with $k \in \mathbb{N}$ work tapes. The tapes are read/written by k separate heads that are housed in the *finite control* of t. The alphabet W of all the tapes is the same, and contains a blank symbol $\beta \in W$ that is used to initialize all squares of all the tapes. At any time, the finite control of t exists (for convenience, we usually just say that t exists) in one of the states of a set S, which is partitioned into the set of *autonomous* states S_a and the set of *polling* states S_p. If the current state of t is polling, then t reads an externally provided/available symbol. Depending on the current state and inputs, t may or may not itself output a symbol. Initially, the state of t is $s_0 \in S$ (that is, the initial state may be either autonomous or polling).

In a single step or time period $\mathcal{T}$, the transducer t does the following. First, if its current state s is in S_p then it reads a symbol $b \in \Sigma$, where Σ is the set of input symbols of t. Then, depending on s, b (if $s \in S_p$), and the k symbols under the read/write heads of the tapes, t (i) does or does not output a symbol $a \in \Delta$, where Δ is the set of output symbols of t, (ii) for each tape, writes a symbol in W on the square under the read/write head and either does not shift the tape or shifts the tape by one square to the left or to the right, and (iii) sets its current state to some $s' \in S$.

Definition 12 (Transducer) *Formally, a transducer t is specified by a tuple $(k, W, \beta, S_a, S_p, s_0, \Sigma, \Delta, \delta, \tau)$, where $\delta : (S_a \times W^k) \cup (S_p \times \Sigma \times W^k) \to S \times W^k \times \{\leftarrow, \rightarrow, \nleftrightarrow\}^k$ is the transition function, and $\tau : (S_a \times W^k) \cup (S_p \times \Sigma \times W^k) \to \Delta \cup \{\epsilon\}$ is the output function. If the current state of t is in S_p then it reads one of the symbols in Σ which is used as a parameter for δ and τ, otherwise the two-parameter form of the functions is used. If t does not output in a step then the result of τ is ϵ. The symbols $\leftarrow, \rightarrow, \nleftrightarrow$ denote the left shift, right shift and no shift operations for the tapes.*

A configuration or instantaneous description of t is a tuple $(q, (u, i), (v, j), \dots) \in S \times (W^* \times \mathbb{N})^k$ where q is the state and $(u, i), (v, j), \dots$ are the pairs of tape sequence and head position such that $1 \le i \le \ell(u)$ & $1 \le j \le \ell(v)$ & $\cdots$. The initial configuration of t is $(s_0, (\beta, 1), (\beta, 1), \dots)$. The set of all valid configurations of t is denoted by C. For $c, c' \in C$, $c \vdash_a^b c'$ denotes that t changes its configuration from c to c' in a single step, with $b \in \Sigma$ if the state component of c is in S_p and $b = \epsilon$ otherwise, and $a \in \Delta \cup \{\epsilon\}$.

Some remarks on the physical limits of machine models:

- The maximum distance that a signal can travel in time $\mathcal{T}$ is $c\mathcal{T}$, where c is the speed of light. Therefore, the finite control of transducers must be contained within a sphere of diameter $c\mathcal{T}$. This implies that as $k + |W| + |S|$ increases, the diameters of the tapes and other components of the finite control become ever smaller to ensure that the overall volume of the finite control remains bounded. (This is analogous to the reduction of the size of transistors in integrated circuits.) The transducer model by itself does not specify any limits on the physical dimensions of transducers. However, our primary assumption that the universe is discrete (likely at the Planck scale) actually implies that the scalability of every model is bounded.

- The measure of scalability of a model is specific to the model, and is a function of the time period and the sizes of input/output spaces. But insofar as real-time intelligence is concerned, the scalabilities of two models for a given time period and sizes of input/output spaces are comparable according to the maximally-intelligent agents that are possible for the models with the given parameters. In practice, determining accurate bounds on the scalability of a model is extremely difficult, and further, determining the maximally-intelligent agent among even a finite number of agents is only an approximable process, since the intelligence measure (Definition 14) is only lower semicomputable.

- It is not necessary that the size of the machines of every model be restricted in the same way as in the transducer model. For example, consider an extension of the transducer model such that multiple transducers can be pipelined in a manner similar to synchronous digital circuits. That is, the output of the first transducer is the input to the second one and so on (assuming there are appropriate input/output interfaces between each successive pair), but the time period of each transducer and also that of the overall machine still remains the same: $\mathcal{T}$. Now, since there is no requirement that the input to the first transducer be processed and carried through the last one within time $\mathcal{T}$, there is no limit in principle to the number of transducers that can be pipelined, and hence no limit to the size of the machines of this model. However, due to the phenomenon of cosmological expansion, it is practically impossible to hold together an arbitrarily large number of transducers indefinitely. Indeed, for the same reason it is also impossible for any transducer to maintain even a single (potentially) infinite tape (Aaronson and Ambainis, 2005, p. 52).

We define the *interface* of an agent based on a transducer with mappings between the perception/action spaces and the transducer's input/output sets (including null). On an abstract level, the interface is a specific interpretation of the transducer's inputs/outputs in terms of perceptions/actions. On a physical level, it may be composed of a variety of devices, such as analog/digital converters, asynchronous/synchronous converters, or even direct connections to sensors/actuators. Note that the interface does not change the time period of the agent; the time period is determined solely by the transducer.

Definition 13 (Agent) *For a perception space $\mathcal{P}$ and action space $\mathcal{A}$, an agent T is specified by a transducer $t := (k, W, \beta, S_a, S_p, s_0, \Sigma, \Delta, \delta, \tau)$, an input mapping $f : \mathcal{P} \to \Sigma$ and an output mapping $g : \Delta \cup \{\epsilon\} \to \mathcal{A}$. The mappings specify the interface of the agent with its environment.*

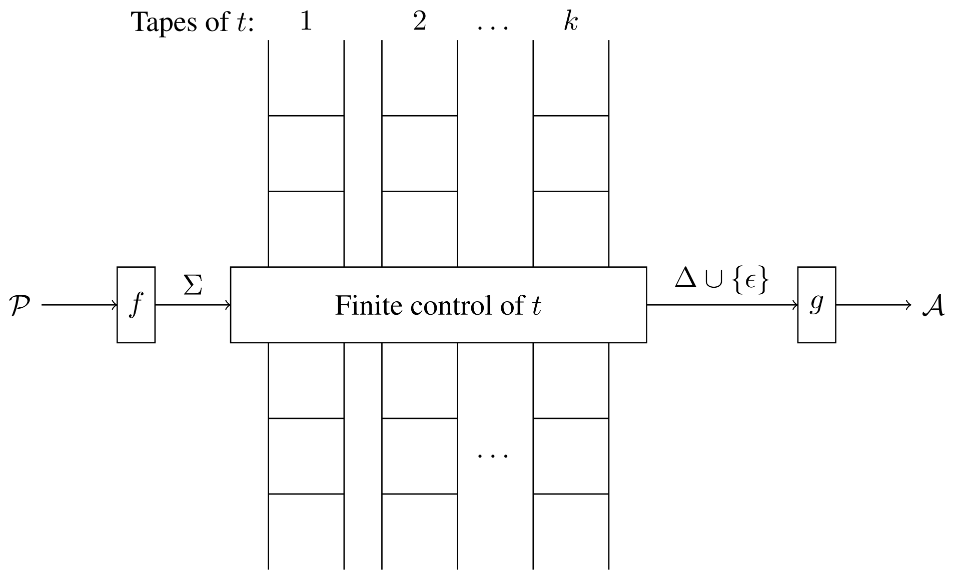

Figure 1: An agent $T := (t, f, g)$ as defined in Definition 13.

Note that $|\mathcal{P}|$ need not be equal to $|\Sigma|$ and $|\Delta \cup \{\epsilon\}|$ need not be equal to $|\mathcal{A}|$. In general, f and g need not be injective or surjective.

The agent T is interpreted in the lower layer of the intelligence measure as follows. In each cycle, a perception $x \in \mathcal{P}$ is received from the upper layer. Let s be the current state of t and $w \in W^k$ the symbols under the read/write heads of t. If $s \in S_p$ then $d := \delta(s, f(x), w)$ and $a := \tau(s, f(x), w)$, otherwise x is discarded and $d := \delta(s, w)$ and $a := \tau(s, w)$. Then, the configuration of t is changed according to d, and the action $g(a)$ is returned to the upper layer. If s is a halting state, that is, δ and τ are undefined, then the action $g(\epsilon)$ is returned.

Formally, the policy $p_T : \mathcal{P}^* \to \mathcal{A}^*$ of the agent T is defined as $p_T(x) = p'_T(x, (s_0, (\beta, 1), (\beta, 1), \dots))$ where the function $p'_T : \mathcal{P}^* \times C \to \mathcal{A}^*$ is defined as

$$p'_T(x, c) := \begin{cases} \epsilon & \text{if } x = \epsilon \\ g(a) \cdot p'_T(x_{2:\ell(x)}, c') & \text{if } \exists (c' \in C)\ \exists (a \in \Delta \cup \{\epsilon\})\ [c \vdash_a^{f(x_1)} c'] \\ g(a) \cdot p'_T(x_{2:\ell(x)}, c') & \text{if } \exists (c' \in C)\ \exists (a \in \Delta \cup \{\epsilon\})\ [c \vdash_a c'] \\ g(\epsilon)^{\ell(x)} & \text{otherwise.} \end{cases}$$

By definition, p_T is real-time computable in the length of its input. Note that in the third condition of p'_T, the state of the transducer of T is autonomous, which causes it to miss or lose the current perception. An environment does not know the internal state or configuration of T, it only knows T via the black-box interfaces f and g.

It is easy to see that for each agent $T := (t, f, g)$ there exists an agent $T' := (t', f', g')$ such that $p_T = p_{T'}$ and the mappings f' and g' are bijective. Also, we could have simplified the definitions above by considering only real-time transducers, that is, transducers for which every state is a polling state and which output a symbol at every step. However, our purpose is to explicitly show how a Turing-complete model is interpreted in the *physically more general* real-time setting.

Note that this doesn't mean that the environments also have to be real-time computable. In fact, if we required the environments to also be real-time computable then we would be making a stronger assumption about the whole universe (that is, agent + environment) than the assumption we made and justified in Section 3. In general, the execution of one step of an agent can require one or more steps of execution of its entire universe. Because of this, an agent may perceive its environment executing "faster" than itself, even if the computational complexity of the environment is higher. A similar view of local versus universal time is expressed by Schmidhuber (1997, p. 202).

Orthogonal to the assumption of general computability of the universe is the assumption that the universe may be probabilistic. In the common interpretation of quantum theory, the "collapse of the wave function" is probabilistic. However, it is possible that the collapse actually occurs only pseudo-randomly and that Heisenberg's uncertainty principle only applies to observers within the universe (Schmidhuber, 1997, p. 203). As such, it is consistent with the current physical theories to assume that the universe is deterministic and computable. Therefore, the specific implications of our other choices in the intelligence measure for deterministic environments are more important than general implications for all environments.

In the upper layer of the intelligence measure, the value of an agent T in an environment μ is given by the value of its policy p_T in μ. We have discussed in Section 2 that every agent eventually stops receiving a positive reward in any deterministic reward summable environment, which is at odds with the fact that it is not always true for probabilistic environments. Since we consider deterministic environments more important as discussed above, we have taken the approach of discounting for obtaining finite values. We have also discussed in Section 3 about the choice of algorithmic probability distributions for weighing the value of agents in different environments. So, we can now give a formal definition of the intelligence measure.

Definition 14 *Let $\mathcal{P}$ be a perception space, $\mathcal{A}$ an action space, γ a lower semicomputable discount function, and U a universal prefix machine. Let $\mu_0, \mu_1, \mu_2, \ldots$ be an enumeration of all lower semicomputable environments over $\mathcal{P}$ and $\mathcal{A}$. Let $\langle\cdot\rangle : \mathbb{N} \to \{0,1\}^*$ be a recursive encoding function. Then, the intelligence of an agent T is defined as*

$$\mathcal{I}(T) := \sum_{i \in \mathbb{N}} R_U(\langle i \rangle) V_\gamma^{p_T \mu_i}.$$

$\mathcal{I}$ is only lower semicomputable. Note that even if we restrict γ to be computable and choose a computable weight function instead of R_U, we would not be able to obtain a computable $\mathcal{I}$ by considering only computable environments since the set of computable environments is not recursively enumerable.

The definition of $\mathcal{I}$ is similar in appearance to that of the universal intelligence measure Υ. However, the two measures are fundamentally different. Υ is defined for arbitrary *policies* while $\mathcal{I}$ is defined for *agents*, and agents are always interpreted by $\mathcal{I}$ such that the policy of every agent is real-time computable. Note that by the assumption of the computability of the universe, which is made by Legg and Hutter (2007) and also us, it follows that every physically realizable agent can indeed be interpreted this way. Another difference is that since Υ considers only (reward summable) computable environments, which are not recursively enumerable, Υ is not even approximable.

The anytime universal intelligence test also uses Υ as its basis. The test was designed with the focus on feasible implementation, and accordingly, many of the changes to Υ (in particular, the restriction to reward-sensitive and balanced environments) are made for the sole purpose of making the evaluation of agents more practical. The focus of this paper is to develop a better theory of intelligence of agents. The main theoretical difference between Υ and the anytime test is that in the latter an agent's response times (time taken to perform an action) affect its intelligence. However, the effect of response times varies with the amount of time that an agent's interaction with an environment is simulated: the number of completed interaction cycles in the testing period that are actually used to calculate the value of the agent in the environment is proportional to the time from the beginning until the last completed action in that period (see Hernández-Orallo and Dowe, 2010, def. 16). While the individual rewards are not scaled (as in discounting) in this method, the adjustment of the value is still done "externally", and is thus necessarily arbitrary. And since the environments still remain unaware of the response times of agents, the perceptions of an agent are independent of its response times. If the motivation behind taking response times into account is that the performance of an agent depends on its response times because environments react differently to different response times, then the agent-environment interactions should be simulated in that way in the first place.

Considering the effect of response times at the finest level of granularity (w.r.t. a machine model or time period, and not necessarily at the extreme Planck scale) naturally leads to the real-time interaction model, where the concept of response time itself is superseded. As the policy of every agent is considered to be real-time computable, *some* action is performed by an agent at every step, and hence the response time (in terms of the number of steps) is always 1, and thus the value adjustment of Hernández-Orallo and Dowe (2010, def. 16) becomes redundant. However, this does not mean that all agents are equally fast; the notion of speed of agents in the real-time setting is more subtle. Informally, the speed of an agent depends on its parallel processing capability. In terms of the machine model that we have used to define the agents, this capability refers, primarily, to the number of tapes of the transducer of an agent. There is a hierarchy of agents and (real-time computable) environments according to the number of tapes, and at each level of the hierarchy, no agent at that level can perform optimally in any environment above that level. But since the number of states and the size of the tape alphabet also contribute to the parallelism of a transducer, the number of tapes alone does not set an upper limit on the overall intelligence of agents. We show these results in the next section.

5. Theoretical Results

In this section, "agent" always refers to that which is formally defined by Definition 13; the theorems do not consider any physical bounds on the scalability of transducers.

It is convenient to express $\mathcal{I}$ in a different form for proving theoretical results. Let p be an arbitrary (in)computable policy. Then,

$$\sum_{i\in\mathbb{N}} R_U(\langle i\rangle)V_\gamma^{p\mu_i} = \sum_{i\in\mathbb{N}} R_U(\langle i\rangle)\lim_{m\to\infty} V_{m\gamma}^{p\mu_i} = \sum_{i\in\mathbb{N}} \lim_{m\to\infty} R_U(\langle i\rangle)V_{m\gamma}^{p\mu_i}. \tag{1}$$

Since the reward space $\mathcal{R}\subset[0,1]$, for all $i\in\mathbb{N}$, $V_{m\gamma}^{p\mu_i}$ is monotonically increasing in m. And since the discount function γ is summable, for all $i\in\mathbb{N}$, $\lim_{m\to\infty} V_{m\gamma}^{p\mu_i}$ exists. Then, the same holds for

$R_U(\langle i \rangle)V_{m\gamma}^{p\mu_i}$. Therefore, by Lebesgue's monotone convergence theorem,

$$(1) \;=\; \lim_{m\to\infty} \sum_{i\in\mathbb{N}} R_U(\langle i \rangle)V_{m\gamma}^{p\mu_i}. \tag{2}$$

Define a conditional probability semimeasure $\xi : (\mathcal{P}\times\mathcal{A})^* \to [0,1]$ as

$$\xi(z) := \sum_{i\in\mathbb{N}} R_U(\langle i \rangle)\mu_i(z). \tag{3}$$

By definition, ξ is a universal lower semicomputable environment (Hutter, 2005, §5.1.2). Then, by expanding $V_{m\gamma}^{p\mu_i}$ in (2) according to Definition 6, rearranging the sums, and using (3),

$$(2) \;=\; \lim_{m\to\infty} V_{m\gamma}^{p\xi} \;=\; V_{\gamma}^{p\xi}. \tag{4}$$

Therefore, $\mathcal{I}$ can be expressed as

$$\mathcal{I}(T) \;=\; V_{\gamma}^{p_T\xi}. \tag{5}$$

That is, the intelligence of an agent T can be expressed as the value of its policy p_T in a single environment: ξ, which is only lower semicomputable.

Now, for the environment ξ and discount function γ, there exists an optimal policy $p_\gamma^\xi := \operatorname{argmax}_p V_\gamma^{p\xi}$ (see Definition 7). We show that there exist agents with intelligence arbitrarily close to $V_\gamma^{p_\gamma^\xi\xi}$.

Theorem 15 *For each $\varepsilon > 0$, there exists an agent M such that $V_\gamma^{p_\gamma^\xi\xi} - \mathcal{I}(M) < \varepsilon$.*

Theorem 15 does not immediately follow from (5), since p_γ^ξ is optimal among *all* policies while the policies p_T of transducers are specifically real-time computable. The main idea of the proof below is that the value of p_γ^ξ can be approximated to arbitrary precision by a policy that mimics it up to a finite number of cycles. Since the approximation can be achieved in a finite number of cycles, even a transducer without any tapes (which is a finite automaton) can be used to compute the approximating policy. We define such a transducer by hard-coding the optimal sequence of actions for each possible sequence of perceptions up to the number of cycles required.

Proof Let $\varepsilon > 0$. Since $V_{m\gamma}^{p_\gamma^\xi\xi}$ is monotonically increasing in m, there exists an $n \geq 1$ such that

$$V_\gamma^{p_\gamma^\xi\xi} - V_{n\gamma}^{p_\gamma^\xi\xi} < \varepsilon. \tag{6}$$

That is, the optimal policy p_γ^ξ achieves (in the expected sense, as usual) a value that deviates less than ε from the maximum value in a finite number of cycles n.

We now define an agent $M := (t, f, g)$ whose policy p_M is the same as p_γ^ξ for the first n cycles. Let the set of inputs Σ and outputs Δ of t be such that $|\Sigma| = |\mathcal{P}|$ and $|\Delta| = |\mathcal{A}| - 1$. Let the interface mappings $f : \mathcal{P} \to \Sigma$ and $g : \Delta \cup \{\epsilon\} \to \mathcal{A}$ be bijective, and let $f^{-1} : \Sigma \to \mathcal{P}$ and $g^{-1} : \mathcal{A} \to \Delta \cup \{\epsilon\}$ be their inverses. The number of tapes $k := 0$. The state set $S := S_p := \{s_x : x \in \bigcup_{i=0}^{n-1} \Sigma^i\} \cup \{s_\hbar\}$, where $\hbar \notin \Sigma$, and the initial state is s_ϵ. The transition function δ and the output function τ are defined as

$$\delta(s_x, b, \epsilon) := \begin{cases} (s_{xb}, \epsilon, \epsilon) & \text{if } x \in \bigcup_{i=0}^{n-2} \Sigma^i \\ (s_\hbar, \epsilon, \epsilon) & \text{otherwise.} \end{cases}$$

$$\tau(s_x, b, \epsilon) := \begin{cases} g^{-1}(p^{\xi}_{\gamma}(f^{-1}(x_1)f^{-1}(x_2)\cdots f^{-1}(x_{\ell(x)})f^{-1}(b))_{\ell(xb)}) & \text{if } x \neq \hbar \\ \epsilon & \text{otherwise.} \end{cases}$$

In the first $n-1$ cycles (or steps), t traverses the state tree from the root s_ϵ to a node s_x where x depends on the perceptions that are received from the environment, and generates outputs (including null) that correspond to the actions of p^{ξ}_{γ} for the same sequence of perceptions. In the nth cycle, it transitions to a non-halting and non-outputting state $s_\hbar$. The action of M in the nth cycle is the same as that of p^{ξ}_{γ}, but this is actually unnecessary since the reward in the nth cycle only depends on the actions in the first $n-1$ cycles.

Then, the policy p_M of M is such that $\forall(x \in \mathcal{P}^n)\ [p_M(x) = p^{\xi}_{\gamma}(x)]$. Also, $V^{p_M\xi}_{m\gamma}$ monotonically increases in m. Hence,

$$V^{p_M\xi}_{\gamma} \;\geq\; V^{p_M\xi}_{n\gamma} \;=\; V^{p^{\xi}_{\gamma}\xi}_{n\gamma}. \tag{7}$$

Therefore, by (5), (7), and (6): $\mathcal{I}(M) \;=\; V^{p_M\xi}_{\gamma} \;\geq\; V^{p_M\xi}_{n\gamma} \;=\; V^{p^{\xi}_{\gamma}\xi}_{n\gamma} \;>\; V^{p^{\xi}_{\gamma}\xi}_{\gamma} - \varepsilon.$ ■

Since for all $M, \mathcal{I}(M) \leq \sup_T\{\mathcal{I}(T)\} \leq V^{p^{\xi}_{\gamma}\xi}_{\gamma}$, Theorem 15 implies that $\sup_T\{\mathcal{I}(T)\} = V^{p^{\xi}_{\gamma}\xi}_{\gamma}$. However, there does not exist an agent M such that $V^{p_M\xi}_{\gamma} = V^{p^{\xi}_{\gamma}\xi}_{\gamma}$, since p_M is always real-time computable while the optimal policy p^{ξ}_{γ} is not real-time computable. Thus, there does not exist a maximally intelligent agent.

Corollary 16 *For every agent M, there exists an agent M' such that $\mathcal{I}(M') > \mathcal{I}(M)$.*

Proof Let M be any agent and p_M its policy, which is real-time computable. Since the environment ξ is only lower semicomputable, the optimal policy p^{ξ}_{γ} is not real-time computable. Hence,

$$\mathcal{I}(M) \;=\; V^{p_M\xi}_{\gamma} \;<\; V^{p^{\xi}_{\gamma}\xi}_{\gamma}.$$

Then, by Theorem 15, there exists an agent M' such that: $V^{p^{\xi}_{\gamma}\xi}_{\gamma} - \mathcal{I}(M') \;<\; V^{p^{\xi}_{\gamma}\xi}_{\gamma} - \mathcal{I}(M).$ ■

The agent M that is defined in the proof of Theorem 15 cannot be effectively constructed since p^{ξ}_{γ} is only approximable. In any case, for a sufficiently small value of ε, the state tree of the transducer t of M is too large to be realizable within the physical bounds imposed on transducers. Practical agents necessarily need to be forgetful; they cannot store and act according to *every* perception. Also, practical agents are usually not designed to "burn out" within a fixed number of cycles.

Nevertheless, theoretical agents like M that are optimal for a finite number of cycles provide an important indication of what is necessary to achieve a high value of intelligence. As $\varepsilon \to 0$ at least either the size of the state set or the number of tapes or the size of the tape alphabet of the transducer of such agents must grow unboundedly since p^{ξ}_{γ} is not (real-time) computable. The size of the state set, the number of tapes, and the size of the tape alphabet indicates the amount of parallelism of an agent.

It is easy to show that for any agent M with a transducer having state set S, tape alphabet W and k tapes, there exists an agent M' such that $p_{M'} = p_M$, and the transducer of M' has a single state, two tape symbols and $k\lceil\log_2|W|\rceil + \lceil\log_2|S|\rceil$ tapes. That is, states and tape alphabets can always be substituted with tapes. However, the converse is not true. The following theorem shows this.

Theorem 17 *For each $k \geq 1$, there exists a computable environment $\mu : (\mathcal{P} \times \mathcal{A})^* \to [0,1]$ and a k-tape agent M such that $V_\gamma^{p_M \mu} = V_\gamma^{p_\gamma^\mu \mu}$ and for each $(k-1)$-tape agent T, $V_\gamma^{p_T \mu} < V_\gamma^{p_\gamma^\mu \mu}$.*

Aanderaa (1974) showed that for each $k \geq 1$, there exists a language L over an alphabet Γ such that $|\Gamma| = 3k$ and L is decidable in real-time by a k-tape Turing machine (and even a k-stack machine) but not by any $(k-1)$-tape Turing machine. The main idea of the proof below is to define a computable environment μ that generates observations randomly but rewards deterministically as follows. A randomly generated observation sequence is interpreted as consisting of encoded sequences of Γ^* separated by end markers. The environment rewards agents based on whether they correctly decide if an encoded sequence corresponds to a sequence in L. As long as a fixed-length code is used (cf. Paul, Seiferas, and Simon, 1981, thm. 3), for each $(k-1)$-tape agent there exists a sequence in Γ^* which the agent fails to decide, and whose prior probability of occurring in encoded form at the beginning of an observation sequence is non-zero. On the other hand, since L can be decided using k stacks, a k-tape agent can be constructed that can create k virtual stacks on its k tapes in a single step, and thus be ready for deciding the next sequence as soon as it has decided the current one.

Proof Let $k \geq 1$. Define $\Gamma := \{\alpha_i, \eta_i, \theta_i : 1 \leq i \leq k\}$, and let $\#$ denote the end marker symbol. The language $L \subset \Gamma^*$ is defined by a transducer, which we also name L for convenience. The number of tapes of L is k, the tape alphabet $W := \{\alpha_i, \eta_i : 1 \leq i \leq k\} \cup \{\beta\}$, where β is the blank symbol, the set of inputs $\Sigma := \Gamma \cup \{\#\}$, the set of outputs $\Delta := \{0, 1\}$, the state set $S := S_p := \{s_{lc} : 0 \leq l \leq 2 \ \& \ c \in W\}$, where $s_{0\beta}$ is the initial state. The transition function δ is defined by Algorithm 1 and the output function τ is defined as

$$\tau(s_{lc}, b, w) := \begin{cases} 0 & \text{if } b = \# \ \& \ (l = 0 \text{ or } l = 2) \\ 1 & \text{if } b = \# \ \& \ l = 1 \\ \epsilon & \text{otherwise.} \end{cases}$$

L works as a set of k stacks, in which elements are pushed or popped according to the input. When L receives an input b that is either α_i or η_i, it pushes b on the ith stack by shifting the ith tape to the left and storing b in the variable c that is a part of all the states; if c already contains the top of some stack j, then it is simultaneously written onto the jth tape. When L receives θ_i, it pops the ith stack. But if the ith stack is empty then L records this exception in the state variable l by setting it to 2 and thereafter does not change l until $\#$ is received. If $l \neq 2$, then if the top of the ith stack was α_i then $l := 0$, and if the top was η_i then $l := 1$. When L receives the end marker $\#$, it resets the state to the initial state and also writes β on the squares under the read/write heads of all the tapes to create new stacks.

The language of L is defined as follows. Let C be the set of all valid configurations of L, and let $c_0 := (s_{0\beta}, (\beta, 1), (\beta, 1), \dots)$. For $w \in \Gamma^*$, $w \in L$ iff there exist $c_i \in C : 1 \leq i \leq \ell(w) + 1$ such that $\forall (1 \leq i \leq \ell(w)) \ [c_{i-1} \vdash^{w_i} c_i] \ \& \ c_{\ell(w)} \vdash_1^{\#} c_{\ell(w)+1}$. In general, let the language of any transducer with the same set of inputs and outputs be defined in this way. Then, for any such transducer t with $k-1$ tapes, the language of t cannot be L (Aanderaa, 1974).

Algorithm 1 $\delta(s_{lc}, b, w)$

```
if $\exists(i)\ [b = \alpha_i \text{ or } b = \eta_i]$
  if $\exists(j)\ [c = \alpha_j \text{ or } c = \eta_j]$
    $(s_{lb}, w_{<j}cw_{j+1:k}, \nleftrightarrow^{i-1}\leftarrow\nleftrightarrow^{k-i})$
  else
    $(s_{lb}, w, \nleftrightarrow^{i-1}\leftarrow\nleftrightarrow^{k-i})$
else if $\exists(i)\ [b = \theta_i]$
  if $l = 2$
    $(s_{2c}, w, \nleftrightarrow^{k})$
  else if $c = \alpha_i$
    $(s_{0\beta}, w, \nleftrightarrow^{i-1}\rightarrow\nleftrightarrow^{k-i})$
  else if $c = \eta_i$
    $(s_{1\beta}, w, \nleftrightarrow^{i-1}\rightarrow\nleftrightarrow^{k-i})$
  else if $w_i = \alpha_i$
    $(s_{0c}, w, \nleftrightarrow^{i-1}\rightarrow\nleftrightarrow^{k-i})$
  else if $w_i = \eta_i$
    $(s_{1c}, w, \nleftrightarrow^{i-1}\rightarrow\nleftrightarrow^{k-i})$
  else
    $(s_{2c}, w, \nleftrightarrow^{k})$
else
  $(s_{0\beta}, \beta^k, \nleftrightarrow^{k})$
```

We encode the symbols of $\Gamma \cup \{\#\}$ with a fixed-length code $\mathcal{C} := \mathcal{O}^d$, where $d := \lceil \log_{|\mathcal{O}|}(3k + 1)\rceil$ is the decoding delay. If $|\mathcal{C}| > 3k + 1$ then we use all extra codes for $\#$. Let $D : \mathcal{C} \to \Gamma \cup \{\#\}$ be the decoding function. Also, let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two distinct elements of $\mathcal{A}$.

Define an agent $M := (t, f, g)$. The number of tapes k, the tape alphabet W, and the set of outputs Δ of t is the same as that of L. The set of inputs $\Sigma := \mathcal{O}$ and the input mapping $f : \mathcal{P} \to \Sigma$ is defined as $f(b) := o(b)$. The state set $S := S_p := \{s_{lcx} : 0 \leq l \leq 2 \ \&\ c \in W \ \&\ x \in \bigcup_{i=0}^{d-1} \Sigma^i\}$, where $s_{0\beta\epsilon}$ is the initial state. For $x \in \bigcup_{i=0}^{d-2} \Sigma^i$, the transition function δ is defined as $\delta(s_{lcx}, b, w) := (s_{lc(xb)}, w, \nleftrightarrow^k)$. For $x \in \Sigma^{d-1}$, $\delta(s_{lcx}, b, w)$ is defined essentially the same as Algorithm 1; all occurrences of b are replaced with $D(xb)$, and the additional variable x in all the succeeding states is set to ϵ. The output function τ and the output mapping $g : \Delta \cup \{\epsilon\} \to \mathcal{A}$ are defined as

$$\tau(s_{lcx}, b, w) := \begin{cases} 0 & \text{if } xb \in \mathcal{C} \ \&\ D(xb) = \# \ \&\ (l = 0 \text{ or } l = 2) \\ 1 & \text{if } xb \in \mathcal{C} \ \&\ D(xb) = \# \ \&\ l = 1 \\ \epsilon & \text{otherwise} \end{cases}$$

$$g(a) := \begin{cases} \mathcal{A}_1 & \text{if } a = 1 \\ \mathcal{A}_2 & \text{otherwise.} \end{cases}$$

For $w \in \Gamma^*$, define $E(w) := \{x \in \mathcal{P}^{d\cdot(\ell(w)+1)} : \forall(1 \leq i \leq \ell(w))\ [D(o(x_{1+d\cdot(i-1)}) \cdots o(x_{d\cdot i})) = w_i] \ \&\ D(o(x_{1+d\cdot\ell(w)}) \cdots o(x_{d\cdot(\ell(w)+1)})) = \#\}$. Then, by definition of M, $w \in L$ iff $\forall(x \in E(w))\ [p_M(x)_{\ell(x)} = \mathcal{A}_1]$. The number of observation symbols d that are used to encode each symbol in $\Gamma \cup \{\#\}$ can be interpreted as a fixed delay (in terms of k) in which an agent is allowed to process the symbol. Paul, Seiferas, and Simon (1981, thm. 3) have shown that the result of Aanderaa (1974) holds for such a fixed delay of processing each symbol. Therefore, for any agent T with $k-1$ tapes, $\exists(w \in \Gamma^*)\ \exists(x \in E(w))\ [p_T(x)_{\ell(x)} \neq p_M(x)_{\ell(x)}]$. In particular, the same holds for the set $E_0(w) := \{x \in E(w) : \forall(1 \leq i \leq \ell(x))\ [r(x_i) = 0]\}$.

Algorithm 2 $\mu(x_n \mid y_{<n}, x_{<n})$

```
Require: $n > 0 \Rightarrow \mu(x_{n-1} \mid y_{<n-1}, x_{<n-1}) > 0$
  if $n = 0$
    1
  else if $(n-1)/d \in \mathbb{N}^+$ & $D(o(x_{n-d}) \cdots o(x_{n-1})) = \#$
    if $y_{n-1} = p_M(x_{<n})_{n-1}$ & $r(x_n) = 1$ or $y_{n-1} \neq p_M(x_{<n})_{n-1}$ & $r(x_n) = 0$
      $|\mathcal{O}|^{-1}$
    else
      0
  else if $r(x_n) = 0$
    $|\mathcal{O}|^{-1}$
  else
    0
```

We now define a computable environment $\mu : (\mathcal{P} \times \mathcal{A})^* \to [0,1]$ for which p_M is optimal for all discount functions. The observation sequences generated by μ are sampled from the uniform measure $\lambda : \mathcal{O}^* \to [0,1]$ defined as $\lambda(\sigma) := |\mathcal{O}|^{-\ell(\sigma)}$. A positive reward is generated only in a cycle immediately following a sequence of cycles in which the observation sequence encodes a sequence $w \in \Gamma^*$ followed by $\#$, and only if the previous action of the agent matches that of M.

The environment μ is factorizable (Hutter, 2005, §4.3.1) since $\forall(\sigma_1, \sigma_2 \in \mathcal{O}^*)\ [\lambda(\sigma_1\sigma_2) = \lambda(\sigma_1)\lambda(\sigma_2)]$ & $\forall(w \in \Gamma^*)\ \forall(x \in E(w))\ \forall(z \in \mathcal{P}^*)\ [p_M(xz) = p_M(x)p_M(z)]$.

Let T be a $k-1$ tape agent, and let $w \in \Gamma^*$ and $x \in \mathcal{P}^*$ such that $x \in E_0(w)$ and $p_T(x)_{\ell(x)} \neq p_M(x)_{\ell(x)}$. Then, for each $b \in \mathcal{O}$, $\mu(x \cdot (b,0) \mid p_T(x)) = \mu(x \cdot (b,1) \mid p_M(x)) = |\mathcal{O}|^{-\ell(x)-1}$. Hence, $V_\gamma^{p_T\mu} \leq V_\gamma^{p_M\mu} - \sum_{b \in \mathcal{O}} |\mathcal{O}|^{-\ell(x)-1}\gamma(\ell(x)+1)$. ■

Theorem 17 does not imply that there exists a k-tape agent that is more intelligent than every $(k-1)$-tape agent, since by Corollary 16 and the proof of Theorem 15, for every agent T with k tapes there exist agents with fewer tapes that are more intelligent than T. However, in general, such agents also require a larger state set than that of T, which again amounts to an increase in parallelism.

6. Conclusion

We have developed an intelligence measure $\mathcal{I}$ that considers an agent's real-time performance in all (lower semi)computable environments, favoring the performance in simpler environments over complex ones in accordance with Occam's razor. We have compared $\mathcal{I}$ with related measures proposed by Legg and Hutter (2007) and Hernández-Orallo and Dowe (2010) and shown how it can give a more accurate evaluation of physically realizable agents. However, $\mathcal{I}$ necessarily requires a detailed machine model of the agent to be measured.

The measure is best used in a process of constructing agents with successively higher intelligence. Such a process need not be formally defined or directly use the measure to generate the agents themselves (which is impractical anyway). A process may use multiple machine models but the choice of the time period needs to be fixed. Finer machine models allow more efficient physical scaling and hence higher parallelism, although it may be possible to increase parallelism without scaling. For example, in digital circuits the propagation delay through logic gates and interconnecting wires can be reduced by using better semiconductors and conductors, and hence more of them can be accommodated within a fixed time period. A process also need not use the intelligence measure alone to select machine models and agents; it may for instance, take the expected power consumption into account as well.

Acknowledgments

Thanks to Tor Lattimore and anonymous reviewers for valuable feedback on earlier drafts.

References

Aanderaa, S. O. 1974. On k-tape versus $(k-1)$-tape real-time computation. In Karp, R. M., ed., *Complexity of Computation (SIAM-AMS Proceedings)*, volume 7, 75–96. American Mathematical Society, Providence, Rhode Island.

Aaronson, S., and Ambainis, A. 2005. Quantum search of spatial regions. *Theory of Computing* 1:47–79.

Hernández-Orallo, J., and Dowe, D. L. 2010. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence* 174(18):1508–1539.

Hutter, M. 2005. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin.

Hutter, M. 2007. On universal prediction and Bayesian confirmation. *Theoretical Computer Science* 384(1):33–48.

Lattimore, T., and Hutter, M. 2011. Time consistent discounting. In Kivinen, J.; Szepesvári, C.; Ukkonen, E.; and Zeugmann, T., eds., *Proceedings of The 22nd International Conference on Algorithmic Learning Theory (ALT'11)*, volume 6925 of *LNAI*, 383–397. Espoo, Finland: Springer, Berlin.

Legg, S., and Hutter, M. 2007. Universal intelligence: A definition of machine intelligence. *Minds & Machines* 17(4):391–444.

Li, M., and Vitányi, P. M. B. 2008. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, New York, third edition.

Paul, W. J.; Seiferas, J. I.; and Simon, J. 1981. An information-theoretic approach to time bounds for on-line computation. *Journal of Computer and System Sciences* 23:108–126.

Rabaey, J. M.; Chandrakasan, A.; and Nikolic, B. 2003. *Digital Integrated Circuits*. Prentice Hall, second edition.

Rosenberg, A. L. 1967. Real-time definable languages. *Journal of the Association for Computing Machinery* 14(4):645–662.

Schmidhuber, J. 1997. A computer scientist's view of life, the universe, and everything. In Freksa, C.; Jantzen, M.; and Valk, R., eds., *Foundations of Computer Science: Potential - Theory - Cognition*, volume 1337 of *Lecture Notes in Computer Science*. Springer, Berlin. 201–208.

Vitányi, P. M. B. 1980. On the power of real-time Turing machines under varying specifications. In *Proceedings of the 7th International Colloquium on Automata, Languages and Programming (ICALP'80)*, volume 85 of *Lecture Notes in Computer Science*, 658–671. Springer, Berlin.