

# The AI Definition and a Program Which Satisfies this Definition

Dimiter Dobrev  
Institute of Mathematics and Informatics  
Bulgarian Academy of Sciences  
*d@dobrev.com*

We will consider all policies of the agent and will prove that one of them is the best performing policy. While that policy is not computable, computable policies do exist in its proximity. We will define AI as a computable policy which is sufficiently proximal to the best performing policy. Before we can define the agent's best performing policy, we need a language for description of the world. We will also use this language to develop a program which satisfies the AI definition. The program will first understand the world by describing it in the selected language. The program will then use the description in order to predict the future and select the best possible move. While this program is extremely inefficient and practically unusable, it can be improved by refining both the language for description of the world and the algorithm used to predict the future. This can yield a program which is both efficient and consistent with the AI definition.

## 1. Introduction

Once, I was talking to a colleague and he told me: *'Although we may create AI someday, it will be a grossly inefficient program as we will need an infinitely fast computer to run it'*. My answer was: *'You just give me this inefficient program which is AI, and I will improve it so that it becomes a true AI which can run on a real-world computer'*.

Today, in this paper I will deliver the kind of program I asked my colleague to give me at that time. I will set out an inefficient program which satisfies the AI definition. I will go further and suggest some ideas and guidance on how this inefficient program can be improved to become a real program which runs in real time. My hope is that some readers of this paper will succeed to do this and deliver the AI we are looking for.

How inefficient is the program described here? In theory, there are only two types of programs – ones which halt and ones which run forever. In practice however, some programs will halt somewhere in the future, but they are so inefficient that we can consider them as programs which run forever. This is the case with the program described here — formally it halts, but its inefficiency makes it unusable (unless the computer is infinitely fast or the world is extremely simple).

**What is the definition of AI?** We will define AI as a policy. An agent who follows this policy will cope sufficiently well. This is true for any world, provided however that there are not any fatal errors in that world. If a fatal error is possible in a given world, the agent may not perform well in that particular world, but his average performance over all possible worlds will still be sufficiently good.

Which worlds we will consider as possible? The world's policies are continuum many. If we do not have any clues as to what the world should be, then we cannot have a clue about what the expected success of the agent should look like. We will assume that the world can be described and such description is as simple as possible (this assumption is known as *Occam's razor*). In

other words, we will choose a language for description of worlds and will limit our efforts only to the worlds described by that language. The worlds whose description is simpler (shorter) will be preferred (will carry more weight).

This paper will consider several languages for description of the world. The first language will describe deterministic worlds. This language will describe the world by means of a computable function, which will take the state of the world and the action of the agent as input and return the new state of the world and the next observation as output. If we know the initial state of the world and agent's actions, this function will give us the life of the agent in that world.

The second language will describe non-deterministic worlds – again by a computable function, but with one additional argument. This argument will be randomness. In this case, we will need to know one more thing in order to obtain the agent's life in that world. We will need to know what that randomness has been.

We will define AI by these two languages and will make the assumption that these two definitions are identical. We will make even the assumption that the AI definition does not depend on our choice of language for description of worlds, and all languages produce the same definition of AI.

On the basis of these two languages we will make two programs which satisfy the AI definition. These two programs will calculate approximately the same policy, but their efficiency would be dramatically different. Therefore, the choice of language for description of the world will not affect the AI definition, but will have a strong impact on the efficiency of the AI obtained through the chosen language.

### Contributions

This paper improves the AI definition initially provided by Hernández-Orallo et al. in 1998 [3] and then substantially improved by Marcus Hutter in 2000 [4]. More precisely, this paper introduces two improvements:

**1. An AI definition which does not depend on the complexity of the world, nor on the length of life.** Papers [3, 4] do provide an AI definition, however, the assumption there is that the complexity of the world and the length of life are limited by some constants and these constants are parameters of the definition.

**2. An AI definition which does not depend on the language for description of the world.** The language in [3, 4] is fixed. Thus, papers [3, 4] imply that there is only one possible way to describe the world.

## 2. Terms of the problem

Let the agent have  $n$  possible actions and  $m$  possible observations. Let  $\Sigma$  and  $\Omega$  be the sets of actions and respectively observations. In the observations set there will be two special observations. These will be the observations *good* and *bad*, and they will provide rewards 1 and -1. All other observations in  $\Omega$  will provide reward 0.

We will add another special observation – *finish*. The agent will never see that observation ( $finish \notin \Omega$ ), but we will need it when we come to define the model of the world. The model will predict *finish* when it breaks down and becomes unable to predict anything more. For us the *finish* observation will not be the end of life, but rather a leap in the unknown. We expect our AI

to avoid such leaps in the unknown and for this reason the reward given by the *finish* observation will be -1.

**Definition 1:** The tree of all possibilities is an infinite tree. All vertices which sit at an even-number depth level and are not leaves will be referred to as action vertices and those at odd-number depth levels will be observation vertices. From each action vertex there will depart  $n$  arrows which correspond to the  $n$  possible actions of the agent. From each observation vertex there will depart  $m+1$  arrows which correspond to the  $m$  possible observations of the agent and the observation *finish*. The arrow which corresponds to *finish* will lead to a leaf. All other arrows lead to vertices which are not leaves.

**Definition 2:** In our terms the world will be a 3-tuple  $\langle S, s_0, f \rangle$ , where:

1.  $S$  is a finite or countable set of internal states of the world;
2.  $s_0 \in S$  is the initial state of the world; and
3.  $f: S \times \Sigma \rightarrow \Omega \times S$  is a function which takes a state and an action as input and returns an observation and a new state of the world.

The  $f$  function cannot return observation *finish* (it is predicted only when  $f$  is not defined and there is not any next state of the world). What kind of function is  $f$  – computable, deterministic or total? The answer to each of these three questions can be *Yes*, but it can also be *No*.

**Definition 3:** A deterministic policy of the agent is a function which assigns a certain action to each action vertex.

**Definition 4:** A non-deterministic policy of the agent is a function which assigns one or more possible actions to each action vertex.

When the policy assigns all possible actions at a certain vertex (moment) we will say that at that moment the policy does not know what to do. We will not make a distinction between an agent and the policy of that agent. A union of two policies will be the policy which we get when choose one of these two policies and execute it without changing that policy. Allowing a change of the chosen policy will lead to something else.

**Definition 5:** Life in our terms will be a path in the tree of all possibilities which starts from the root.

Each life can be presented by a sequence of actions and observations:

$$a_1, o_1, \dots, a_b, o_b, \dots$$

We will not make a distinction between a finite life and a vertex in the tree of all possibilities because there is a one-to-one correspondence between these two things.

**Definition 6:** The length of life will be  $t$  (the number of observations). Therefore, the length of life will be equal to the length of the path divided by two.

**Definition 7:** A completed life is one which cannot be extended. In other words, it will be an infinite life or a life ending with the observation *finish*.

When we let an agent in a certain world, the result will be a completed life. If the agent is non-deterministic then the result will not be unique. The same applies when the world is non-deterministic.

### 3. The grade

Our aim is to define the agent's best performing policy. For this purpose we need to assign some grade to each life. This grading will give us a linear order by which we will be able to determine the better life in any pair of lives.

Let us first determine how to measure the success of each life  $L$ . For a finite life, we will count the number of times we have had the observation *good*, and will designate this number with  $L_{good}(L)$ . Similar designations will be assigned to the observations *bad* and *finish*. Thus, the success of a finite life will be:

$$Success(L) = \frac{L_{good}(L) - L_{bad}(L) - L_{finish}(L)}{|L|}$$

Let us put  $L_i$  for the beginning of life  $L$  with a length of  $i$ . The  $Success(L)$  for infinite life  $L$  will be defined as the limit of  $Success(L_i)$  when  $i$  tends to infinity. If this sequence is not convergent, we will take the arithmetic mean between the limit inferior and limit superior.

$$Success(L) = \frac{1}{2} \cdot \left( \liminf_{i \rightarrow \infty} (Success(L_i)) + \limsup_{i \rightarrow \infty} (Success(L_i)) \right)$$

By doing this we have related each life to a number which belongs to the interval  $[-1, 1]$  and represents the success of this life. Why not use the success of life for the grade we are trying to find? This is not a good idea because if a world is free from fatal errors then the best performing policy will not bother about the kind of moves it makes. There would be one and only one maximum success and that success would always be achievable regardless of the number of errors made in the beginning. If there are two options which yield the same success in some indefinite time, we would like the best performing policy to choose the option that will yield success faster than the other one. Accordingly, we will define the grade of a completed life as follows:

**Definition 8:** The grade of infinite life  $L$  will be a sequence which starts with the success of that life and continues with the rewards obtained at step  $i$ :

$$Success(L), reward(o_1), reward(o_2), reward(o_3), \dots$$

**Definition 9:** The grade of finite and completed life  $L$  will be the same sequence, but in this sequence for  $i > t$  the members  $reward(o_i)$  will be replaced with  $Success(L)$ :

$$Success(L), reward(o_1), \dots, reward(o_t), Success(L), Success(L), \dots$$

(In other words, the observations that come after the end of that finite life will receive some expectation for a reward and that expectation will be equal to the success of that finite life.)

In order to compare two grades, we will take the first difference. This means that the first objective of the best performing policy will be the success of entire life, but its second objective will be to achieve a better reward as quickly as possible.

## 4. The expected grade

**Definition 10:** For each deterministic policy  $P$  we will determine  $grade(P)$ : the grade we expect for the life if policy  $P$  is executed.

We will determine the expected grade at each vertex  $v$  assuming that we have somehow reached  $v$  and will from that moment on execute policy  $P$ . The expected grade of  $P$  will be the one which we have related to the root.

We will provide a rough description of how we relate vertices to expected grades. Then we will provide a detailed description of the special case in which we look for the best grade, i.e. the expected grade of the best performing policy.

Rough description:

1. Let  $v$  be an action vertex.

Then the grade of  $v$  will be the grade of its direct successor which corresponds to action  $P(v)$ .

2. Let  $v$  be an observation vertex.

2.1. Let there be one possible world which is a model of  $v$ .

If we execute  $P$  in this world we will get one possible life. Then the grade of  $v$  will be the grade of that life.

2.2. Let there be many possible worlds.

Then each world will give us one possible life and the grade  $v$  will be the mean value of the grades of the possible lives.

The next section provides a detailed description of the best performing policy. The main difference is that when  $v$  is an action vertex, the best performing policy always chooses the highest expected grade among the expected grades of all direct successors.

## 5. The best performing policy

As mentioned above, we should have some clue about what the world looks like before can have some expectation about the success of the agent. We will assume that the world can be described by some language for description of worlds.

Let us take the standard language for description of worlds. In this language the world is described by a computable function (this is the case in [3, 4]). We will describe the computable function  $f$  by using a Turing machine. We will describe the initial state of the world as a finite word over the machine alphabet. What we get is a computable and deterministic world which in the general case is not a total one.

**Definition 11:** A world of complexity  $k$  will be a world in which:

1. The  $f$  function is described by a Turing machine with  $k$  states.

2. The alphabet of that machine contains  $k+1$  symbols  $(\lambda_0, \dots, \lambda_k)$ .

3. The initial state of the world is a word made of not more than  $k$  letters. The alphabet is  $\{\lambda_1, \dots, \lambda_k\}$ , i.e. the alphabet of the machine without the blank symbol  $\lambda_0$ .

Here we use the same  $k$  for three different things as we do not need to have different constants.

We will identify the best performing policy for the worlds of complexity  $k$  (importantly, these worlds are finitely many). For this purpose we will assign to each observation vertex its best grade (or the expected grade if the best performing policy is executed from that vertex onwards).

Let us have life  $a_1, o_1, \dots, a_t, o_t, a_{t+1}$ .

Let this life run through the vertices  $v_0, w_1, v_1, \dots, w_t, v_t, w_{t+1}$ ,

where  $v_0$  is the root,  $v_i$  are the action vertices and  $w_i$  are the observation vertices.

Now we have to find out how many models of complexity  $k$  are there for vertex  $v_t$ .

**Definition 12:** A deterministic world is a model of  $v_t$  when in that world the agent would arrive at  $v_t$  if he executes the corresponding actions  $(a_1, \dots, a_t)$ . The models of each action vertex are identical with the models of its direct successors.

**Definition 13:** The best performing policy for the worlds of complexity  $k$  will be the one which always chooses the best grade (among the best grades of the direct successors).

**Definition 14:** The best grade of vertex  $w_{t+1}$  is determined as follows:

**Case 1.** Vertices  $v_t$  and  $w_{t+1}$  do not have any model of complexity  $k$ .

In this case the best grade for  $w_{t+1}$  will be *undef*. At this vertex the policy will not know what to do (across the entire subtree of  $v_t$ ) because the best grade for all successor vertices will be *undef*.

If we do not want to introduce an *undef* grade, we can use the lowest possible grade – the sequence of countably many -1s. The maximal grade will be chosen among the vertices which are different from *undef*. Replacing *undef* with the lowest possible grade will give us the same result.

**Case 2.** Vertices  $v_t$  and  $w_{t+1}$  have one model of complexity  $k$ .

Let this model be  $D$ . In this case there are continuum many paths through  $w_{t+1}$  such that  $D$  is model of all those paths. From these paths (completed lives) we will select the set of the best paths. The grade we are looking for is the grade of these best paths. Each of these paths is related to a deterministic policy of the agent. We will call them the best performing policies which pass through vertex  $w_{t+1}$ .

This is the procedure by which we will construct the set of best deterministic policies: Let  $P_0$  be the set of all policies which lead to  $w_{t+1}$ . We take the success of each of these policies in the world  $D$ . We create the subset  $P_1$  of the policies which achieve the maximum success. Then we reduce  $P_1$  by selecting only the policies which achieve the maximum for  $reward(o_{t+2})$  and obtain subset  $P_2$ . Then we repeat the procedure for each  $i > 2$ . In this way we obtain the set of the best deterministic policies  $P$ . (The best ones of those which pass through vertex  $w_{t+1}$  as well as the best ones for the paths which pass through vertex  $w_{t+1}$ . As regards the other paths, it does not matter how the policy behaves there.)

$$P = \bigcap_{i=0}^{\infty} P_i$$

We can think of  $P$  as one non-deterministic policy. Let us take some  $p \in P$ . This will give us the best grade:

$$Success(p), reward(o_{t+1}), reward(o_{p,t+2}), reward(o_{p,t+3}), \dots$$

Here we drop out the members  $reward(o_i)$  at  $i \leq t$  because they are uniquely defined by  $v_t$ . The next member depends on  $w_{t+1}$  and  $D$ , but does not depend on  $p$ . The remaining members depend on  $p$ .

Another way to express the above formula is:

$$\max_{p \in P_0} Success(p), reward(o_{t+1}), \max_{p \in P_1} reward(o_{p,t+2}), \max_{p \in P_2} reward(o_{p,t+3}), \dots$$

**Case 3.** Vertices  $v_t$  and  $w_{t+1}$  have a finite number of models of complexity  $k$ .

Let the set of these models be  $M$ . Again, there are continuum many paths through  $w_{t+1}$  such that each of these paths has a model in  $M$ . These paths again form a tree, but while in case 2 the branches occurred only due to a different policy of the agent, in this case some branches may occur due to a different model of the world. Again, we have continuum many deterministic policies, but now they will correspond to subtrees (not to paths) because there can be branches because of the model. Again we will try to find the set of best performing deterministic policies and the target grade will be mean grade of those policies (the mean grade in  $M$ ).

We will again construct the set of policies  $P_j$ . Here  $P_1$  will be the set of policies for which the mean success reaches its maximum. Accordingly,  $P_2$  will be the set of policies for which the mean  $reward(o_{t+2})$  reaches its maximum and so on. This is how the resultant grade will look like:

$$\max_{p \in P_0} \sum_{m \in M} q_m \cdot Success(m, p), \sum_{m \in M} q_m \cdot reward(o_{m,t+1}), \max_{p \in P_1} \sum_{m \in M} q_m \cdot reward(o_{m,p,t+2}), \dots$$

If we take some  $p \in P$ , the resultant grade will look like this:

$$\sum_{m \in M} q_m \cdot Success(m, p), \sum_{m \in M} q_m \cdot reward(o_{m,t+1}), \sum_{m \in M} q_m \cdot reward(o_{m,p,t+2}), \dots$$

Here  $q_i$  are the weights of the worlds which have been normalized in order to become probabilities. In this case we assume that the worlds have equal weights, i.e.:

$$q_i = \frac{1}{|M|}$$

■

What we have described so far looks like an algorithm, however, rather than an algorithm, it is a definition because it contains uncomputable steps. The so described policy is well defined, even

though it is uncomputable. Now, from the best grade for complexity  $k$ , how can we obtain the best grade for any complexity?

**Definition 15:** The best grade at vertex  $v$  will be the limit of the best grades at vertex  $v$  for the worlds of complexity  $k$  when  $k$  tends to infinity.

How shall we define the limit of a sequence of grades? The number at position  $i$  will be the limit of the numbers at position  $i$ . When the sequence is divergent, we will take the arithmetic mean between the limit inferior and limit superior.

**Definition 16:** The best performing policy will be the one which always chooses an action which leads to the highest grade among the best grades of the direct successors.

What makes the best performing policy better than the best performing policy for worlds of complexity  $k$ ? The first policy knows what to do at every vertex, while the latter does not have a clue at the majority of vertices because they do not have any model of complexity  $k$ . The first policy can offer a better solution than the latter policy even for the vertices at which the latter policy knows what to do because the first policy also considers models of complexity higher than  $k$ . Although at a first glance we do not use Occam's razor (because all models have equal weights), in earnest we do use Occam's razor because the simpler worlds are calculated by a greater number of Turing machines, meaning that they have a greater weight.

## 6. The AI definition

**Definition 17:** AI will be a computable policy which is sufficiently proximal to the best performing policy.

At this point we must explain what makes a policy proximal to another policy and how proximal is proximal enough. We will say that two policies are proximal when the expected grades of these two policies are proximal.

**Definition 18:** Let  $A$  and  $B$  be two policies and  $\{a_n\}$  and  $\{b_n\}$  are their expected grades. Then the difference between  $A$  and  $B$  will be  $\{\varepsilon_n\}$ , where:

$$\varepsilon_n = \sum_{i=0}^n \gamma^i (a_i - b_i) = \varepsilon_{n-1} + \gamma^n (a_n - b_n)$$

Here  $\gamma$  is a discount factor. Let  $\gamma=0.5$ . We have included a discount factor because we want the two policies to be proximal when they behave in the same way for a long time. The later the difference occurs in time, the less impact it will have.

When  $n$  goes up,  $|\varepsilon_n|$  may go up or down. We have made the definition in this way because we want the difference to be small when the expected grade of policy  $A$  hovers around the expected grade of policy  $B$ . I.e., if for  $n-1$  the higher expected grade is that of  $A$  and for  $n$  the higher expected grade is that of  $B$ , then in  $\varepsilon_n$  the increase will offset the decrease and vice versa.

**Definition 19:** We will say that  $|A-B| < \varepsilon$  if  $\forall n |\varepsilon_n| < \varepsilon$ .

## 7. A program which satisfies the definition

We will describe an algorithm which represents a computable policy. Each action vertex relates to an uncompleted life and the algorithm will give us some action by which this life can continue. This algorithm will be composed of two steps:

**1. The algorithm will answer the question ‘What is going on?’** It will answer this question by finding the first  $k$  for which the uncompleted life has a model. The algorithm will also find the set  $M$  (the set of all models of the uncompleted life, the complexity of which is  $k$ ). Unfortunately, this is uncomputable. To make it computable we will try to find efficient models with complexity  $k$ .

**Definition 20:** An efficient model with complexity  $k$  will be a world of complexity  $k$  (definition 11), where the Turing machine uses not more than  $1000.k$  steps in order to make one step of the life (i.e. to calculate the next observation and the next internal state of the world). When the machine makes more than  $1000.k$  steps, the model will return the observation *finish*.

The number  $1000$  is some parameter of the algorithm, but we assume this parameter is not very important. If a vertex has a model with complexity  $k$ , but does not have an efficient model with complexity  $k$ , then  $\exists n (n > k)$  such that the vertex has an efficient model with complexity  $n$ .

**2. The algorithm will answer the question ‘What should I do?’** For this purpose we will run  $h$  steps in the future over all models in  $M$  and over all possible actions of the agent. In other words, we will walk over one finite subtree and will calculate *best* for each vertex of the subtree (this is the best expected grade up to a leaf). Then we will choose an action which leads to the maximum by *best* (this is the best partial policy).

**Definition 21:** A partial subtree of vertex  $v_t$  over  $M$  with depth  $h$  will be the subtree of  $v_t$  composed of the vertices which i) have a depth not more than  $2h$  and ii) have a model in  $M$ .

**Definition 22:** The grade up to a leaf of vertex  $v_{t+i}$  to the leaf  $v_{t+j}$  will be:

Case 1. If  $j=h$ , this will be the sequence:

$$\text{Success}(v_{t+j}), \text{reward}(o_{t+i+1}), \dots, \text{reward}(o_{t+j})$$

Case 2. If  $j < h$ , then the sequence in case 1 will be extended by  $h-j$  times *Success*( $v_{t+j}$ ). The purpose of this extension is to ensure that the length of the grade up to a leaf will always be  $h-i+1$ .

**Definition 23:** The best expected grade up to a leaf (this is *best*):

1. Let  $v_{t+i}$  be an action vertex.

1.1. If  $v_{t+i}$  is a leaf, then  $\text{best}(v_{t+i})$  will be the grade up to a leaf of  $v_{t+i}$  to the leaf  $v_{t+i}$ .

1.2. If  $v_{t+i}$  is not a leaf then:

$$\text{best}(v_{t+i}) = \max_{a \in \Sigma} \text{best}(w_a)$$

By  $w_a$  here we designate the direct successor of  $v_{t+i}$  resulting from action  $a$ . The same applies accordingly to  $v_o$  below.

2. Let  $w_{t+i}$  be an observation vertex. Then:

$$best(w_{t+i}) = \sum_{o \in \Omega'} p_o \cdot (reward(o) \text{ insert\_at\_1\_in } best(v_o))$$

Thus, we take the *best* of the direct successor  $v_o$  and extend it by one by inserting  $reward(o)$  at position 1. Here  $\Omega' = \Omega \cup \{finish\}$  and  $p_o$  is the probability of the next observation being  $o$ . Let  $M(v)$  be the set of the models of  $v$ . Then:

$$p_o = \frac{(\sum_{m \in M(v_o)} q_m)}{(\sum_{m \in M(w_{t+i})} q_m)} = \frac{|M(v_o)|}{|M(w_{t+i})|}$$

In this formula  $q_m$  are the weights of the models. The last equality is based on the assumption that all models have equal weights. If  $M(v_o) = \emptyset$  then  $p_o = 0$  and it will not be necessary to calculate  $best(v_o)$ . ■

So far we showed how the best partial policy is calculated. Will that be the policy of our algorithm? The answer is *No* because we want to allow for some tolerance.

If two policies differ only slightly in the first coordinates of their expected grades, then a minor increase of  $h$  is very likely to reverse the order of preferences. Therefore, for a certain policy to be preferred, it should be substantially better (i.e. the difference at some of the coordinates should be greater than  $\varepsilon$ ).

We will define the best partial policy with tolerance  $\varepsilon$  and that will be the policy of our algorithm.

## 8. The tolerance $\varepsilon$

We will modify the above algorithm by changing the *best* function. While the initial *best* function returns the best grade, the modified function will return the set of best grades with tolerance  $\varepsilon$ .

How shall we modify the search for the maximum grade to a search for a set of grades? The previous search looked at the first coordinate and picked the grades with the highest value at that coordinate. The search then went on only within these grades to find the ones with the highest value of the second coordinate and so on until it settles for a single grade. The modified search will pick i) the grades with the highest value of the first coordinate and ii) the grades which are at distance  $\varepsilon$  from the maximum value. Let  $E_0$  be the initial set of grades. Let in  $E_0$  there be  $n$  grades, all of them with length  $m+1$ . We will construct the sequence of grade sets  $E_0, \dots, E_{m+1}$  ( $E_{i+1} \subseteq E_i$ ) and the last set  $E_{m+1}$  will be the target set of best grades with tolerance  $\varepsilon$ . Let  $E_0 = \{G_1, \dots, G_n\}$  and  $G_j = g_{j0}, \dots, g_{jm}$ . We will also construct the target grade  $\alpha$  ( $\alpha = \alpha_0, \dots, \alpha_m$ ). The target set of grades  $E_{m+1}$  will be comprised of the grades at distance  $\varepsilon$  from  $\alpha$ .

**Definition 24:** The target grade  $\alpha$  and the target set  $E_{m+1}$  are obtained as follows:

$$\begin{aligned} \alpha_0 &= \max_{G_j \in E_0} g_{j0} \\ E_1 &= \{ G_j \in E_0 \mid \alpha_0 - g_{j0} < \varepsilon \} \\ \alpha_1 &= \max_{G_j \in E_1} g_{j1} \end{aligned}$$

$$E_2 = \{ G_j \in E_1 \mid (\alpha_0 - g_{j0}) + \gamma \cdot (\alpha_1 - g_{j1}) < \varepsilon \}$$

Here  $\gamma$  is again a discount factor. Thus, we have modified the way in which the maximum is calculated. We also need to modify the sum of the grades.

Now the individual grades will be replaced with sets of grades. We will develop all possible combinations and calculate the sum for each combination. The resulting set will be the set of all sums for all possible combinations.

The only remaining thing to do now is to select the next move. We will take the sets of grades provided by the *best* function for the direct successors of  $v_t$ . Then we will make the union of these sets and from that union we will calculate the set of best grades with tolerance  $\varepsilon$ . Finally, we will select one of the actions which take us to one of these best grades.

## 9. Is this AI?

Does the algorithm described above satisfy our AI definition? Before that we must say that the algorithm depends on the parameters  $h$  and  $\varepsilon$ . In order to reduce the number of parameters, we will assume that  $\varepsilon$  is a function of  $h$ . For example, this function can be  $\varepsilon = h^{-0.5}$ .

**Statement 1:** When the value of  $h$  is sufficiently high, the described algorithm is sufficiently proximal to the best performing policy.

Let the best performing policy be  $P_{best}$ , and the policy calculated by the above algorithm with parameter  $h$  be  $P_h$ . Then statement 1 can be expressed as follows:

$$\forall \varepsilon > 0 \exists n \forall h > n ( |P_{best} - P_h| < \varepsilon )$$

Although we cannot prove this statement, we can assume that when  $h$  tends to infinity then  $P_h$  tends to the best performing policy for the worlds the complexity of which is  $k$ . When  $t$  tends to infinity,  $k$  will reach the complexity of the world or tend to infinity. These reflections make us believe that the above statement is true.

## 10. A world with randomness

The first language for description of worlds which discussed here describes deterministic worlds. But, if the world involves some randomness, then the description obtained by using that language would be very inaccurate. Accordingly, we will add randomness to the language for description of worlds. This would improve the language and make it much more expressive.

The new language will also describe the world by a computable function. However, this function will have one additional argument – randomness. By randomness we will mean the result from rolling a dice. Let the complexity of the world be  $k$ . Then the dice will have  $k$  faces and can accordingly return  $k$  possible results. The probabilities of occurrence of one of these results will be  $p_1, \dots, p_k$ .

**Definition 25:** A model of life until moment  $t$  with complexity  $k$  will be a world with complexity  $k$  and randomness with a length of  $t$ . We want that life to be generated by that model and that randomness. The randomness will be some word  $R$  of length  $t$ . The  $R$  letters will be those from the Turing machine alphabet except  $\lambda_0$ .

The weight of the model is the probability of occurrence of  $R$ .

**Definition 26:** The weight of the model will be  $p_1^{L_{\lambda_1}(R)} \cdot \dots \cdot p_k^{L_{\lambda_k}(R)}$ .

We will set the probabilities  $p_1, \dots, p_k$  of the model such that the probability of occurrence of  $R$  becomes maximal:

$$p_i = \frac{L_{\lambda_i}(R)}{|R|}$$

Thus, we will end up with some low-weight models where the probability of occurrence of the life represented by the model is very low, and some heavy-weight models in which the probability of occurrence is higher.

## 11. A definition with randomness

Similar to the process described above, we will define the best performing policy for the models the complexity of which is  $k$ . (An important element here is that these models have different weights.) We will develop the policy which represents the limit when  $k$  tends to infinity, and that will be the best performing policy. Again, AI will be defined as a computable policy which is sufficiently proximal to the best performing policy.

**Statement 2:** The two AI definitions are identical.

This means that the best performing policy for worlds without randomness is the same as the best performing policy for worlds with randomness. Before we can prove this statement, we need to prove that:

**Statement 3:** If we have some word  $\omega$  over the alphabet  $\{0, 1\}$  such that the instances of 1 occur with a probability of  $p$ , and if we make a natural extension of this word, then the next letter will be 1 with probability  $p$ .

What is a natural extension? Let us take the first (simplest) Turing machine which generates  $\omega$ . The natural extension will be the extension generated by that Turing machine.

While we cannot prove statement 3, we can offer two ideas about how to prove it:

The first idea is a practical experiment. We will write a program which finds the natural extension of a sequence and then we will run a series of experiments. We will keep feeding into the program various  $\omega$  words where 1 occurs with probability  $p$ . Then we will check the extensions and will calculate the average probability for all these experiments. If the experiments are many and if the average probability obtained from these experiments is  $p$ , then we can assume that statement 3 is true.

The second idea is to prove the statement by theoretical reasoning. Let us have a computable function  $f$  from  $\mathbb{N}$  to  $\mathbb{N}$ . Suppose we start from the number  $n$ . The resultant sequence will be  $\{f^i(n)\}$ . We will convert this sequence into sequence  $\{b_i\}$  which is made of instances of 0 and 1.

The number  $b_i$  will be zero iff  $f^i(n)$  is an even number. Let  $\omega$  be some beginning of  $\{b_i\}$ . What do we expect the next member of  $\{b_i\}$  to be?

Case 1. Sequence  $\{b_i\}$  is cyclic and has the form  $\omega_1\omega_2^*$ . Let  $\omega$  be longer than  $\omega_1$ . Then there is some beginning of  $\omega_2$  which is part of  $\omega$  and for that beginning the instances of 1 occur with probability  $p$ .

Case 2. Sequence  $\{f^i(n)\}$  has a long beginning in which odd numbers occur with probability  $p$ . We do not have a reason to expect that the  $p$  probability will change.

## 12. A program with randomness

We will develop a program which satisfies an AI definition based on models with randomness. We will proceed in the similar way as above, but with some differences.

We will not search for the first  $k$  for which there is a model until moment  $t$  with complexity  $k$  since such a model exists for very low value of  $k$ . Instead, we will assume that  $k$  is fixed and  $k$  is parameter of the algorithm.

The first step will be to find all models of complexity  $k$  of vertex  $v_t$ . The second step will be to run at depth level  $h$  across a partial subtree of vertex  $v_t$  over i) all discovered models, ii) over all possible actions of the agent and iii) over all probabilities  $R_1R_2$ , where  $R_1$  is the probability of the model and  $R_2$  is the probability after  $t$ . Here  $R_1$  is fixed (it is determined by the model), and  $R_2$  runs over all possibilities.

The next statement will be similar to statement 1:

**Statement 4:** When the values of  $k$  and  $h$  are sufficiently high, the described algorithm is sufficiently proximal to the best performing policy.

We assert that when the values of the parameters are sufficiently high, both algorithms will calculate approximately the same policy. However, are the two algorithms equally efficient?

In practice both algorithms are infinitely inefficient, however, the second algorithm is far more efficient than the first one. We will look at three cases:

1. Let us have a simple deterministic world. By *simple* we mean that its complexity  $k$  is very low. In this case the first algorithm will be slightly more efficient because it will find the model quickly. The second algorithm will find the same model because the deterministic models are a subset of the non-deterministic ones.

2. Let us have a deterministic world which is not simple, i.e. its complexity  $k$  is high. In this case the first algorithm will need a huge amount of time in order to find a model of the world. Moreover, rather than the real model of the world, it will probably find some simplified explanation. That simplified explanation will model the life until moment  $t$ , but after a few more steps the model will err. The second algorithm will also find a simplified explanation of the world, but that simplified explanation will be non-deterministic. While both algorithms will predict the future with some degree of error, the description which includes randomness will be better and more accurate. Moreover, the description with randomness will be much simpler (with smaller  $k$ ).

3. Let us have a world with randomness. In this case the second algorithm has a major advantage. It will find the non-deterministic model of the world and will begin predicting the future in the best possible way. It may appear that the first algorithm will not get there at all, but this is not the case. It will get there, too, but much later and not so successfully. The non-deterministic model consists of a computable function  $f$  and randomness  $R$ . There exists a computable function  $g$  which generates  $R$ . The composition of  $f$  and  $g$  will be a deterministic model of the world at moment  $t$ . Certainly, after a few more steps  $g$  will diverge from the actual randomness and  $f^o g$  will not be a model of the world anymore. Then we will have to find another function  $g$ . All this means that a deterministic function can describe a world with randomness, but such description will be very ungainly and will work only until some moment  $t$ . The non-deterministic model gives us a description which works for any  $t$ .

The conclusion is that the choice of language for description of the world is very important. Although these two languages provide identical AI definitions, the programs developed on the basis of each language differ substantially in terms of efficiency.

### 13. A world with many agents

The world with randomness can be imagined as a world with one additional agent who plays randomly. Let us assume that there are many agents in the world and each of these agents belongs to one of the following three types:

1. Friends, i.e. agents who help us.
2. Foes, i.e. agents who try to disrupt us.
3. Agents who play randomly.

Let the number of additional agents be  $a$  (all excluding the protagonist). Let each additional agent have  $k$  possible moves ( $k$  is the complexity of the world). We will assume that the protagonist (that's us) will play first and the other agents will play after us in a fixed order. We assume that each additional agent can see everything (the internal state of the world, the model including the number of agents and the type of each agent, i.e. friend or foe, as well as the moves of the agents who have played before him). We will also assume that the agents are very smart and capable to calculate which move is the best and which move is the worst.

The model of the world will again be a Turing machine, but that machine will have more arguments (the internal state of the world and the move of the protagonist, plus the moves of all other agents). The model will also include the type of each agent, i.e. friend or foe. Furthermore, the model of life until moment  $t$  will include the moves of all  $a$  agents at all steps until  $t$ .

Once again, we will develop an AI definition on the basis of this new and more complicated language. We will continue with the assumption that the third definition is identical to the previous two. We will also develop a program which looks for a model of the world in the set of worlds with many agents. In the end of the day we will see that the new language is far more expressive: If we have at least one foe in the world this way of describing the world is much more adequate and, accordingly, the AI program developed on the basis of that language is far more efficient.

## 14. Conclusion

We examined three languages for description of the world. On the basis of each language, we developed an AI definition and assumed that all three definitions are the same. Now we will make an even stronger assertion:

**Statement 5:** The AI definition does not depend on the language for description of the world on the basis of which the definition has been developed.

We cannot prove this statement although we suppose that it is true. We also suppose that the statement cannot be proven (similar to the thesis of Church).

Although we assumed that the AI definition does not depend on the language for description of the world, we kept assuming that the program which satisfies this definition strongly depends on the choice of language. The comparison between the first two languages clearly demonstrated that the second language is far more expressive and produces a far more efficient AI.

Let us look at one more language for description of worlds – the language described in [2]. That language describes the world in a far more efficient way by defining the term ‘algorithm’. The term ‘algorithm’ enables us plan the future. For example, let us take the following: ‘I will wait for the bus until it comes. Then I will go to work and will stay there until the end of the working hours.’ These two sentences describe the future through the execution of algorithms. If we are to predict the future only by running  $h$  possible steps, then  $h$  will necessarily become unacceptably large.

The language described in [2] is far more expressive and lets us hope that it can be used to produce a program which satisfies the AI definition and which is efficient enough to work in real time.

## 15. Acknowledgements

I wish to thank my colleague Ivan Soskov [5], with whom we discussed the efficiency of AI several years ago. Let me also thank my colleagues Dimitar Dimitrov [6] and Joan Karadimov [7]. This paper was inspired in a conversation I had with them. Now I wish to apologize to Marcus Hutter and Shane Legg for the unfair allegation in my paper [1] that they had used some work of mine without citing it. Later on, when I carefully read Marcus Hutter’s paper [4] I realized that he had published there the main concepts well before I did.

## References

[1] Dobrev D. (2013). Comparison between the two definitions of AI. *arXiv:1302.0216 [cs.AI]*

[2] Dobrev D. (2020). Language for Description of Worlds. *arXiv:2010.16243 [cs.AI]*

[3] Hernández-Orallo, J., & Minaya-Collado, N. (1998). A formal definition of intelligence based on an intensional variant of Kolmogorov complexity.

*Proc. intl symposium of engineering of intelligent systems (EIS'98), February 1998, La Laguna, Spain (pp. 146–163). : ICSC Press.*

- [4] Hutter, M. (2000). A Theory of Universal Artificial Intelligence based on Algorithmic Complexity. *arXiv:cs.AI/0004001 [cs.AI]*
- [5] Ivan Soskov (1954-2013) <http://www.fmi.uni-sofia.bg/fmi/logic/soskov/>
- [6] Dimitar Dimitrov. <https://scholar.google.com/citations?hl=en&user=pxzF6o0AAAAJ>
- [7] Joan Karadimov. <https://github.com/joankaradimov/>