AI that thinks in its mind

Dimiter Dobrev

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

If we aim to create AGI, our first job is to enable it understand the world. The key to understanding has a name and that name is *world model*. This is what AGI must look for. In fact, rather than looking for a model, we will aim to find a description of the world. For this purpose, we need a language for description of worlds. We will use the game of chess to create the language we need. We have already done this in a previous paper, but then the agent was able to see the chessboard, while now it will play blind. Playing without seeing the chessboard makes the problem more complex and requires the addition of abstract ED models. The result will be a world model which will enable AGI think in its mind and plan its actions.

Keywords: Artificial General Intelligence, World Model, Event-Driven Model.

1. Introduction

Why do we choose to go for a world model, while everybody bets on Large Language Models (LLMs)? Because the understanding approach delivers better results than statistical approaches.

In medicine, there are doctors who use drugs and who try to figure out why the drug works. There are also healers who treat their patients with herbs and rely entirely on statistics without understanding what is happening and why herbs help. While we do not deny the statistical approach, understanding does deliver better results. One can use statistics only if vast experience has been accumulated before. Thus, statistics will help you act by some analogy based on things that have already happened, but cannot offer you a fundamentally new solution.

For example, LLMs cannot play chess because they cannot understand the position on the chessboard. This should not surprise us. LLMs are unable to even do something as simple as addition of two arbitrary numbers. In other words, thinking without understanding is not proper thinking but only imitation of thinking. (This assertion is supported by many. Read e.g. the opinion of Gary Marcus [1]. Very recently Yann LeCun also endorsed this opinion [2].)

To play chess, AGI must understand the world. It must understand the state of the world (the position on the chessboard) and the rules of the game. In the world of chess there is also an opponent who plays against the agent. The world model consists of the set of internal states and a function that describes the rules of the game and the opponent.

We already created a model of the chess game in a previous paper, see [3]. That model was created on the basis of Event-Driven models (ED models). These models provide a partial description of the world through directed graphs. Let us assume that the world has some property which partitions the set of its states in disjoint subsets. These subsets will be the vertices of the directed graph, and the arrows will correspond to events. We will call these ED models *real models* because at any given time the agent will be at one of the graph vertices. In this paper, we will also introduce the concept of *abstract ED models,* which will differ from the real ones in that the agent will not be able to “enter” the vertices of the model.

An example of a real ED model are the days of the week. This is a model with seven states, and the event that triggers the transition from one state to the next one is “midnight”. This is a real model because the states are real and at any given time the world is in one of these states (one of the days of the week).

An example of an abstract ED model are the remainders when dividing by 7 (i.e., the numbers from 0 to 6). Again, we have a model with seven states. The event that switches between them will be “next”. The two models are very similar, but the latter is an abstract one because the states and the event that switches between them are abstract.

In [3] we described the chessboard by using real ED models. Here we are going to play blind, so we will have to use abstract models.

We will consider three versions of the chess game. The first version will be simpler because the agent will play alone by turning the chessboard around and playing alternately the white pieces and the black pieces. This version will be simpler because in the world there will not be another agent (opponent).

In the second version, we will have two players who will take turns to play with the white and respectively black pieces after each game. In the third version, the agent will always play with the white pieces. The third version will be the most difficult to understand because the agent will have to realize that there are black pieces as well, even though it will never touch them.

In addition to being able to understand, we also need to think in mind. The addition of two arbitrary numbers requires us to execute an algorithm. When executing an algorithm, we do not know how long this execution will take. It may take long or even to run forever. Therefore, thinking in mind must be asynchronous so that it does not block our AGI.

2. World model

We can imagine that the set of internal states is ℕ (because we can encode them in ℕ). We can imagine that the function of the model is from ℕ to ℕ (here again we use encoding to add the actions and observations).

It makes perfect sense to limit the search for the world model only to the set of computable functions (as is the case in papers [4–6]). However, we will extend our search to a larger set, and use semi-decidable predicates, randomness and additional agents.

For example, in the real world we often use the rule: “If there is proof, then it is true.” Although the predicate “there is proof” is semi-decidable, we do not hesitate to use it in the description of the real world.

We will assume that in the world there is randomness as well as other agents whose behavior is not fully predictable. In other words, we will assume that the world is not deterministic.

If we were to look for a deterministic world model we would have to understand the world completely and thereby say exactly what is going to happen. To illustrate this with the game of chess, we can assume that our opponent is a computer program and we found that program. In this case we will be able to predict with absolute accuracy the next move of our opponent. If we are faced with randomness, we can assume that it is not randomness, but pseudo-randomness and we found the function that generates this pseudo-randomness, and thereby become able to predict exactly what the next pseudo-random number will be.

Even if the world is deterministic, it is still better to simplify its description by presenting the agents as black boxes. That is, we will make some assumptions about their behavior without trying to understand exactly how these agents operate. We will represent these black boxes as oracles, and the world model will be a computable function that uses oracles.

3. Actions and observations

We will describe the world of chess when the agent plays blind. The agent’s action will consist of the coordinates of two chess squares:

$$\langle x_1, y_1\rangle,\ \langle x_2, y_2\rangle$$

These two squares describe the agent's move. (The piece in $\langle x_1, y_1\rangle$ is moved to $\langle x_2, y_2\rangle$.)

The observation will have the following form:

$$\langle x_3, y_3\rangle,\ \langle x_4, y_4\rangle,\ Result$$

This is the opponent's move and the result of the two moves. *Result* will have 8 possible values *{win, loss, draw, correct move, bad 1, bad 2, bad 3, bad 4}*. Here, *win* means victory for the white and *loss* means victory for black. Why did we provide four options for an incorrect move? Because we want to simplify the agent's task and give it a clue why its move is incorrect.

bad 1 – there is no white piece in column x_1.

bad 2 – there is no white piece at coordinates $\langle x_1, y_1\rangle$.

bad 3 – $\forall y$ the move $\langle x_1, y_1\rangle$, $\langle x_2, y\rangle$ is incorrect.

bad 4 – the move $\langle x_1, y_1\rangle$, $\langle x_2, y_2\rangle$ is incorrect.

(If agent plays with the black pieces, we replace *white* with *black*. If two or more of the above statements are true, we will stop to the first one. That is, we will choose the smallest possible value for *bad*.)

To give meaning to this world, we will introduce a certain reward function that will depend only on the result.

$$reward(R) = \begin{cases} 10, & R = win \\ -10, & R = loss \\ 0, & R = draw \\ 0, & R = correct\ move \\ -400, & R = bad\ 1 \\ -300, & R = bad\ 2 \\ -200, & R = bad\ 3 \\ -100, & R = bad\ 4 \end{cases}$$

With this definition the agent will be able to check whether there is a white piece at $\langle x_1, y_1\rangle$. He will execute the action $\langle x_1, y_1\rangle$, $\langle x_2, y_2\rangle$, where x_2 and y_2 are arbitrary. He will get some *Result* and if *reward(Result)> -300* then there is a white piece at $\langle x_1, y_1\rangle$.

This will be a testable event. That is, an event that is not directly observed, but one which can be confirmed by a certain test. The test in this case is to try to move this white piece.

The event has parameters, so we will call it a predicate. This event depends on the pieces the agent is playing with – white or black. When the agent plays with the black pieces, the same predicate will tell whether there is a black piece in the square. We will present the event with the predicate

my_piece(x, y)

Now we want to define another predicate:

move(Color, $\langle x_1, y_1\rangle$*,* $\langle x_2, y_2\rangle$ *)*

This predicate will be true when we can move the piece with color *Color* from square $\langle x_1, y_1\rangle$ to square $\langle x_2, y_2\rangle$ in accordance with the chess rules. However, simply being able to move the piece does not mean that this will be a correct move since there is one more rule which says that we cannot be in check after the move.

Therefore, the *move* event will not be testable. Let the agent play with the color *Color* and let the move be possible. Then *reward(Result)> -100* will be true for most, but not for all cases, because after a certain move we may find ourselves in check. Therefore, the test result will not be definitive.

An event which is not always observed, but is observed frequently, will be called a probabilistic event. Therefore, the *move* event will be probabilistically testable.

We could help the agent a bit more and add *bad 5* to make the *move* event testable. But we have already helped him enough. Let the agent be able to detect probabilistic events as well.

4. Playing alone

In a blind game we will need abstract ED models in order to describe the world. Let us begin with a simpler version of the chess game the description of which will include some abstract ED models and one real ED model, which we will call **B&W** (Figure 1). In this world, the agent will play alone against himself. It will play the first move with the white pieces, then turn the chessboard around and play the next move with the black pieces, and so on.

In this world, when the agent plays with the white pieces, it must wish victory to the whites and vice versa. This implies some bifurcation of personality. The agent must have two sets of consciousness because he must keep something in mind, and that something must change when he sits at the other side of the chessboard. The agent cannot start deciding at each step which way to go. He must have already chosen certain intermediate goals and started executing some algorithms. An example of an intermediate goal is “exchange queens.” An example of execution of an algorithm is moving towards the intermediate goal. The consciousness must remember what intermediate goals the agent has chosen and where he is headed.

You know the tale of Buridan’s donkey, which was paralyzed in hesitation between two haystacks as it could not decide which one to graze from. If the donkey was void of consciousness, it might shuttle between the two haystacks because at every step it would reconsider its choice and change direction. We assume that we have a consciousness donkey that chooses one of the two haystacks and starts the algorithm to approach the chosen stack. The consciousness donkey can also reconsider its choice, but not at every step.

For simplification purposes here will assume that *reward* returns zero for *win* and *loss*. That is, we will assume that the agent does not pursue victory, but only strives to play correct moves. In this world, there will not be a second agent (opponent), so we will assume that the observation consists only of *Result*.

We will start our description of this world with the **B&W** model which tells us which pieces the agent is playing with (Figure 1).

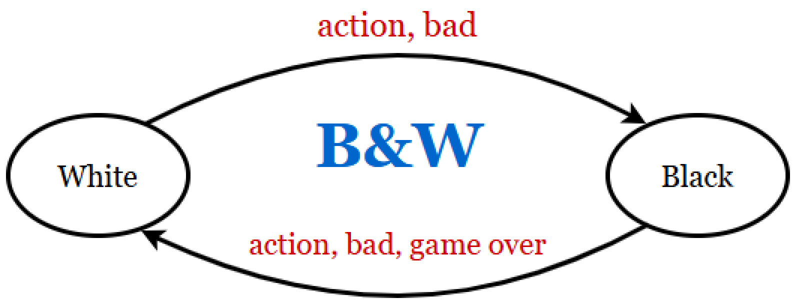

Figure 1

Here, *action* means any action of the agent, *bad* means any of the four types of incorrect moves, and *game over* means *win, loss,* or *draw*.

After each action of the agent, the model will switch to the next state, and if the move is incorrect, it will switch back so that the state remains the same. That is, for one step, the state can switch twice – once after the action and once more after the observation. Wouldn't it be better if the state switched only if the action is a correct move? We have a rule which tells that a move is incorrect if it puts us in check. Because of this rule, it is appropriate to be able to imagine that we have played the incorrect move and, if we are in check, to switch back.

We have put another arrow at *game over*. It is there because once a game is over, the next game must start with a move by the white pieces.

Our objective is not simply to describe the game of chess, but also provide a description which can be found automatically. How can the **B&W** model be found? ED models are described by certain events that switch the states of the model. In addition to these events, we have something called *trace*. Traces are events that allow us to distinguish between the various states of the model. A model whose states are identical would be meaningless. It is the trace that makes a model meaningful. These are events which occur in some states and do not occur in others. The trace can be permanent or transient. In other words, an existing specificity can appear and disappear or move from place to place. Whether the trace is permanent or transient is not so important. The important thing is to ensure that there is some trace.

The trace that will enable us discover the **B&W** model is the predicate *my_piece(1, 1)*. This predicate will be true until we make a correct move. Then it will be false until we make another correct move, and so on. This trace will be transient because when we move the white rook, it will disappear. The important thing about this trace – even though it is transient – is that it will enable us find the **B&W** model.

While it is definitely difficult to find a model with so many arrows, we can notice that *my_piece(1, 1)* periodically changes its value. Once we make this observation, we can start looking for the events that switch it. We do not need to find all the arrows at once. We can add them gradually. In other words, we can start with a simple model and improve it until we find what we are looking for.

5. Abstract ED models

Now that we already have the predicate *my_piece* and the model **B&W**, we can use them to create the predicates *white_piece* and *black_piece*. We can check for example the predicate *white_piece* when the model **B&W** is in *White* state. Therefore, this is a testable predicate. (The test does not need to be possible at any time. It would be sufficient when it can be executed in certain circumstances.)

We already have an idea of the white and black pieces. Now we need to somehow figure out the chessboard on which these figures are arranged. This will be an abstract ED model with 64 states. Why an abstract ED model and not a real one? Because the states of the world will not be partitioned in 64 disjoint subsets. There will not be a special square for the agent, i.e. an active square in which the agent is located or which the agent observes.

The abstract model will have abstract states, but a real trace. I.e., the square *⟨1, 1⟩* will be something abstract, but there will be a real piece there. This piece can be moved therefore we mean a transient trace.

The most natural way to obtain this abstract ED model is to take the domain of the predicate *my_piece*(x_1, y_1). This is the Cartesian product of two sets with 8 elements each. Here we can benefit from the fact that our world is defined in a very simple and natural way (the action is the Cartesian product of 4 sets with 8 elements each). We could assume that the input and output are encoded in a way which makes them very difficult to understand (decode), but the world is sufficiently complicated and we do not need to complicate it further. Therefore, we will assume that the input and output are presented in the simplest and most natural way possible.

We have presented the chessboard as a Cartesian product of two sets with 8 states each. This can tell us where the pieces are, but in order to move them, we need to introduce abstract events that will enable us move on the board. We will assume that x_1 and y_1 are described by the numbers from 1 to 8. The natural operation here is *next*. For x_1 we will call this operation *right*, and its inverse will be *left*. For y_1 these operations will be *up* and *down*, respectively.

Again, we assume that the world is not needlessly complicated. We could have scrambled the numbers from 1 to 8 by some arbitrary permutation, but then it would be difficult to define the event *right*.

To describe the movement of the chess pieces, we should notice that $\langle x_1, y_1\rangle$ и $\langle x_2, y_2\rangle$ describe the states of the same abstract ED model, and that the match is the same (i.e., the same coordinates in $\langle x_1, y_1\rangle$ and in $\langle x_2, y_2\rangle$ point to the same square). To this end, a helpful observation will be that if we successfully move a white piece from $\langle x_1, y_1\rangle$ *to* $\langle x_2, y_2\rangle$ then there will be a white piece at $\langle x_2, y_2\rangle$ (not always, but almost always).

6. Algorithms

To describe the movement of the pieces, we will use algorithms. These are dependencies described by ED models. The execution of an algorithm is the period during which the dependency is maintained. In [3], we used algorithms over real ED models, but now the ED models will be abstract. While in [3] we actually changed the world by doing certain actions through the algorithm, now we will only be interested in whether the algorithm can be executed.

Thus, we will define the predicate *move*, which will show us the possible moves. This predicate will connect the states of the abstract ED model (the squares). We will describe it with algorithms that tell us how to move from one square to another. We will start with the algorithm of the knight, which moves in an L-like pattern (Figure 2). We can present this algorithm as $up^2.right$. The algorithm $right.up^2$ would do the same job. The algorithm $up^3.right.down$ would do almost the same job, but this algorithm would be weaker because sometimes it would go outside the board and then it will not work.

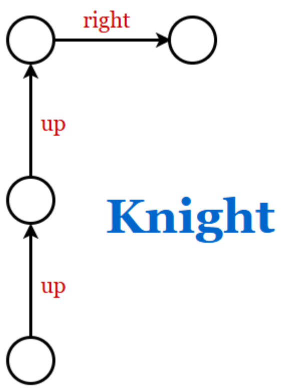

Figure 2

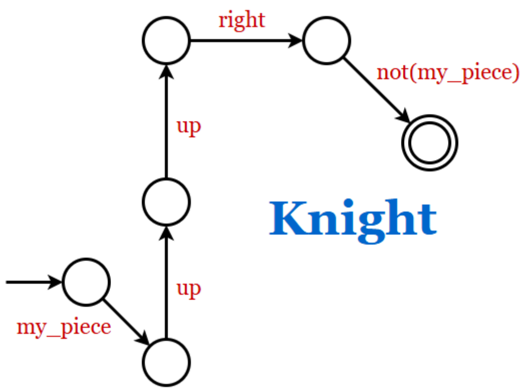

Figure 3

We are describing an algorithm of the *going home* type (see [3]). This means that if we execute the algorithm, we will reach the goal, but nothing tells us to follow exactly this path. In [3] we considered *railway-type* algorithms. The rail track there was defined by the world, which did not allow us to go outside the track defined by the algorithm. In [3] it was even possible to move back and forth (without leaving the rail track). Here we are not bound by rails and can move as we wish, but the algorithm describes a certain path, and it is the simplest possible.

To the knight algorithm we must add two rules: we can only move our own pieces and we cannot capture our own pieces. The result is an algorithm which uses the trace (Figure 3). Here we have an abstract ED model with a real trace. An abstract ED model could have many real traces. For example, we can play blind on 10 chessboards at the same time. In this case the chessboard (which is an abstract ED model) will have 10 traces that switch sequentially.

Certainly, the movement of the knight is described by 8 algorithms similar to the one in Figure 3. We can say that the movement of the knight is the disjunction of these 8 algorithms. A disjunction of algorithms is a non-deterministic algorithm. Let's see the algorithm of the rook, which is also non-deterministic (Figure 4).

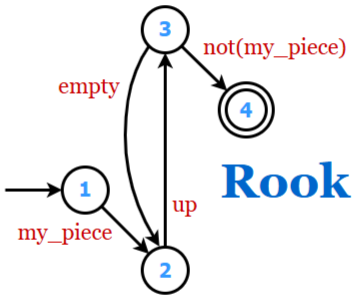

Figure 4

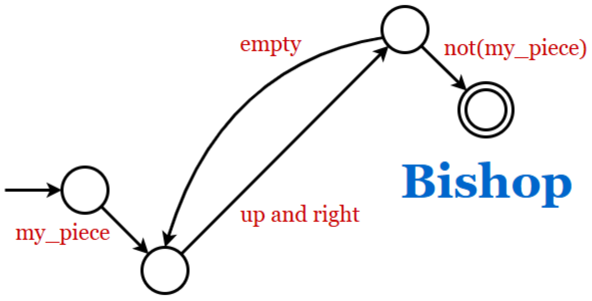

Figure 5

Here, *empty* means *not(white_piece) and not(black_piece)*. This algorithm describes how the rook moves *up*. It is non-deterministic because from state 3, when we have an empty square, we can non-deterministically move to state 2 or state 4.

Figure 5 shows the bishop algorithm for *up and right*. The interesting thing here is that we can move in diagonal direction, i.e., the abstract events *up* and *right* can be performed simultaneously. In [3], these were real events and the world did not allow them to occur simultaneously. Therefore, the bishop algorithm here is simpler than the one in [3].

7. Objects

The algorithms that can be applied to a chess piece are the properties of that piece. These algorithms determine what kind of piece it is. For example, the moves of the rook are determined by four algorithms, and the moves of the bishop by another set of four algorithms. The moves of the queen are determined by the algorithms of the rook and of the bishop. The important features of a chess piece are the properties it has as well as the properties it does not have. Thus, the queen cannot be mistaken for a bishop. It has all of the properties of a bishop, but it has also other properties which the bishop does not have.

We will introduce the *object* abstraction. Let us assume that by his action the agent moves an object from square $\langle x_1, y_1\rangle$ to square $\langle x_2, y_2\rangle$. We might imagine that only the properties are moved from one square to another, but the world will be easier to understand if we assume that there are objects and that the properties are packaged in objects.

Our perception of the world consists of the chessboard (an abstract ED model with 64 states) and real objects which are the trace of this model (the pieces on the chessboard). We will figure out the specific position on the board and see how this position evolves over time.

When we have a transient trace, the natural operations are to add an object to a state (square), remove an object, and move an object. What happens when we execute the action

$\langle x_1, y_1\rangle$, $\langle x_2, y_2\rangle$ can be explained in a natural way like this: “The object is moved from $\langle x_1, y_1\rangle$ to $\langle x_2, y_2\rangle$.”

Another relatively natural operation is to reverse (undo the last operation). We will assume that after a *bad* observation the last operation is undone and the chessboard returns to its previous position.

The only operation that is more complex and unnatural is the arrangement of the initial position of the chessboard after the observation *game over*.

8. We cannot be in check

If the agent is to avoid making incorrect moves, he must be able to predict when a move will be incorrect. The basic rule says that if a move cannot be made according to the chess rules, then it is incorrect.

$$\begin{array}{l} not(\ move(My\_Color,\ \langle x_1, y_1\rangle, \langle x_2, y_2\rangle)\ ), \\ action(\ \langle x_1, y_1\rangle, \langle x_2, y_2\rangle\ ) \\ \Rightarrow bad \end{array}$$

(*My_Color* is the state of the **B&W** model.)

The predicate we use here is semi-decidable, and we even use the negation of a semi-decidable predicate. The assertion is that there exists some execution of the algorithm *move*. In our specific case, this predicate is decidable because the number of pieces in the chess game is finite and the question of whether or not a move exists is decidable. In the general case, however, whether an execution exists will be a semi-decidable question, but we said that in describing the world we will use semi-decidable predicates and even their negations because this is a correct description of the world.

There is another case of an incorrect move. This is when our king can be captured on the next move (when we are in check after the move). This can be described as follows:

$$\begin{array}{l} action(\ \langle x_1, y_1\rangle, \langle x_2, y_2\rangle\ ), \\ possible(\ move(My\_Color,\ \langle x_3, y_3\rangle, \langle x_4, y_4\rangle\ ), king(\ \langle x_4, y_4\rangle\ )\ ) \\ \Rightarrow bad \end{array}$$

Here, *possible* means that an execution exist. More precisely, there exist some squares $\langle x_3, y_3\rangle$, $\langle x_4, y_4\rangle$, for which the algorithm can be executed. In this execution, *My_Color* is reversed because while we were thinking in mind we made *action*, which means that the **B&W** model had switched its state.

When thinking in mind, we must assume that the action in our mind has the same consequences as if that action had actually been performed. When we open an umbrella in our mind, we must assume that in our thoughts the rain will no longer fall on us and make us wet. We must distinguish between mental and real actions. We may have opened the umbrella in our mind, but in reality the umbrella may still be folded and we are still drenched by the rain.

When performing an action, we change our perception of the state of the world and thus have a real idea of what the current state is. When thinking in mind, we change the state of the world mentally, but this does not affect our perception of the actual state of the world.

9. Two players

We described the game of chess when the agent plays against itself. Now we will assume that in the world there is a second agent and that agent plays against us. The assumption that a second agent exists is an important abstraction which will help us understand the world. Trying to describe the world together with the second agent would have been a very difficult task. The second agent may be a human or a complex computer program. In both cases, it is difficult to understand (describe accurately) that agent. Therefore, we will imagine the second agent as a black box. We will not know how it thinks (or how it works). We will only make some very basic assumptions. For example, we will assume that the second agent is our foe and plays against us.

In other words, the idea of a multi-agent world is reduced to the task of presenting the other agents as black boxes and thereby conceal their internal complexity, and ultimately achieve a simplified description of the world.

Let us assume that after each game the two players turn the chessboard around and play with the opponent's previous pieces. In the single-player version, the agent was turning the chessboard around after each move and it was easier for him to get an idea of the white and black pieces. Now the agent will change the pieces less often, but he will still be able to figure out what the black pieces do, albeit with more difficulty.

In this version, there will again be one real ED model, and this will be model **B&W 2** (Figure 6), which will replace **B&W**.

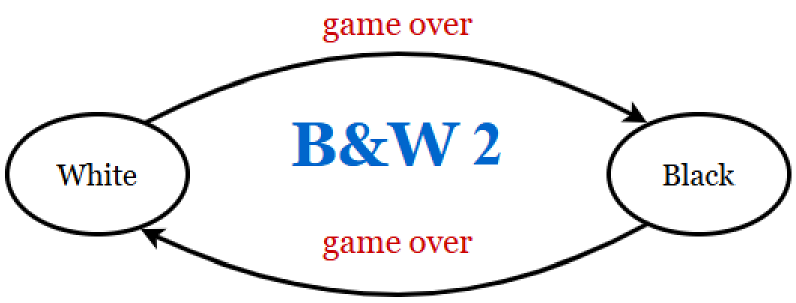

Figure 6

In the previous version, we assumed that the state of the world (the position on the chessboard) changes after an action (by moving a piece). Now we will assume that after the observation the state of the world also changes (then the opponent is moving a piece). Let us play with the white pieces, and the opponent with the black ones.

We will aim to find an explanation for everything. We will assume that the black pieces do not move on their own, but are moved by someone else. Furthermore, we will assume that this someone is hostile and wants us to lose. On the other hand, we will assume that this someone is not omnipotent and cannot move the pieces as he pleases. In other words, we will assume that the opponent is bound to follow the same rules that we are bound to follow. We will find this out when we notice that the opponent only makes correct moves. We will assume that the opponent has some freedom, but his freedom is limited within a certain framework.

Here, in order to use the same algorithms for both players, we will assume that the argument *Color* depends both on the state of **B&W 2** and on who is making the move (we or the opponent).

We will assume that when we play with the black pieces, *reward* reverses the values for *win* and *loss*. Thus, when we play with black pieces, we want them to win. This will not be a bifurcation of personality because we will have a single mind that knows what it wants, even though the intermediate goal after each game will alternate from *win* to *loss* and vice versa.

And now we will again need a rule that says we cannot allow them to capture our king. The form of this rule will be:

$action(\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle),$
$possible(observation(\langle x_3, y_3 \rangle, \langle x_4, y_4 \rangle, Result), king(\langle x_4, y_4 \rangle))$
$\Rightarrow bad$

Here the value of *Result* is irrelevant. What matters is that the opponent is able to capture our king. Now it is important to distinguish between what is *possible* and what is *allowed*. In this case, it is impossible the opponent to capture our king because we are not allowed to make such a stupid move. In other words, this will never happen, but in principle, if we played such a move, the opponent could capture our king.

We need the above rule in order to know which moves are allowed. As for the opponent, he is also disallowed to make such a stupid mistake. Things here depend on what kind of opponent we have. The opponent may think *n* moves ahead and avoid mistakes that are visible *n* moves ahead. We cannot know why he does not make these mistakes: because he is disallowed to do so or because he chooses not to make them. Whether the opponent is forbidden to make stupid mistakes or simply never makes them does not make a difference. In both cases, we have the same world.

10. Only with white

Now let us have two players who will not take turns to play with the opponent's previous pieces. Let the agent always play with the white pieces, and the opponent play always with the black. In this version of the game, we will not have the real ED model **B&W**. If we decide to have such a model, it would be a model with a single state, and an ED model with a single state is not a model at all.

This world will be more difficult to understand because it will be very difficult for the agent to arrive at the idea of black pieces. We have a test for detecting a white piece, but with black pieces it is more complicated.

We can detect black pieces by analogy. We have white pieces. By analogy, we will also have black pieces, and they will be the same as the white ones, but black. This is how physicists discovered antimatter. The principle is: "If matter exists, why should not antimatter exist, too? It will be the same as matter, but anti."

It is more natural to discover the black pieces by the fact that they stand in the way of the white ones. For example, both the white and the black pieces hinder the movement of the white rook, but they hinder it in different ways. The white piece stops it immediately, and the black piece stops it at the next step. When something stands in the way, we can assume that this thing is on the square where the black piece actually is, or we can assume that it is on the next square. In other words, finding the black pieces will be a difficult task.

11. Versions

Our goal is to make AGI, rather than simply a program that plays chess. Perhaps the abstract ED model we used is too easy to find. It is true that the squares on the chessboard are something abstract that cannot be seen directly, but we refer to these squares by their coordinates. In addition, each square has a name (coordinates). We can imagine an abstract ED model in which we know the names of only part of the states. For example, let us take the set of all humans. We know the names of only some of them and can refer only to those we know by name. Let us take the relation *parent*. Through this relation, we can define *brother* and *cousin*.

Another example of an abstract ED model is if in the game of chess we change the way we describe the moves: instead of using the coordinates of the two squares, we can describe the move as *(Type, Which One, X, Y)*. Here the first two arguments describe the piece, and the numbers *X* and *Y* describes the displacement relative to the current position. An example would be *(rook, 2, 0, -7)*. The number 2 here indicates which rook we are going to move, because we have two white rooks. Numbers 0 and -7 indicates how many steps and in which direction we will take.

In the previous world, the pieces were properties of the squares. Now it will be the opposite. In this version of the world, there is not any reference to squares. The references are to the pieces, and their coordinates are their property, which will not be easy to discern. Now the chessboard will be an abstract ED model whose states are not referenced at all.

It would be very difficult to understand a world described in this way, but nobody learns chess by playing blind. We learn with a chessboard with pieces and a teacher who explains how the pieces move. We only start playing blind once we know the rules. For humans, the difficulty of playing blind comes from the need to memorize the position on the chessboard. The problem we set here is much more difficult, that is, to understand the rules of the game by trial and error. By solving this more difficult problem, we learn how we can find abstract ED models and understand the world.

12. Conclusion

Understanding (the world model) is rapidly displacing the statistical approach (LLM). This means that we are very close to creating the real AI (AGI). Perhaps some reader of this paper will be the one to make the decisive leap and create AGI. However, please act responsibly and do not rush. AI is not just another step on the way; it is the final step. Before you take that final step and leap into the abyss of the unknown, think carefully about what kind of AI you want to create. Read papers such as [7] in which this issue is discussed.

References

[1] Marcus, G. (2025) Game over for pure LLMs. Even Turing Award Winner Rich Sutton has gotten off the bus. https://garymarcus.substack.com/p/game-over-for-pure-llms-even-turing.

[2] Snyder, G. (2025). Yann LeCun, Pioneer of AI, Thinks Today's LLM's Are Nearly Obsolete. *Newsweek.AI.* https://www.newsweek.com/nw-ai/ai-impact-interview-yann-lecun-artificial-intelligence-2054237

[3] Dobrev, D. (2023) Language for Description of Worlds. Part 2: The Sample World. *Serdica Journal of Computing 17(1), 2023, pp. 17-54.*

[4] Hutter, M. (2000). A Theory of Universal Artificial Intelligence based on Algorithmic Complexity. *arXiv:cs.AI/0004001 [cs.AI]*

[5] Hutter, M. (2007) UNIVERSAL ALGORITHMIC INTELLIGENCE A mathematical top→down approach. *In Artificial General Intelligence, 2007.*

[6] Hernández-Orallo, J., Minaya-Collado, N. (1998). A formal definition of intelligence based on an intensional scenario of Kolmogorov complexity. *Proc. intl symposium of engineering of intelligent systems (EIS'98), February 1998, La Laguna, Spain (pp. 146–163). : ICSC Press.*

[7] Dobrev, D., Ivanov, L., Popov, G., Tzanov, V. (2024) How Can We Make AI with a Nice Character? https://www.researchgate.net/publication/384159921_How_Can_We_Make_AI_with_a_Nice_Character.