Demonstration of Event-Driven Models

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

The program Artificial Intelligence has to be able to understand the world. To do this, it has to build a model of the world. The language (format) in which this model will be described is very important. Could this format be Turing Machine or Markov decision process? Yes, it can, but only if we have an infinitely fast computer and an infinitely long training time. Can we offer a description format in which the world description is simple enough to be found automatically. Yes, Event-Driven models are such a format. We will demonstrate how the rules of the game of chess can be described by Event-Driven models and the resulting description will be simple enough.

Keywords: Artificial Intelligence, Event-Driven Model, Definition of Property, Definition of Algorithm.

What is the purpose of this report?

We want to demonstrate that with Event-Driven models [1] we can describe very complex worlds in a very simple way.

Why do we want to describe complex worlds in a simple way?

Our goal is to build a program that will cope well in an arbitrary world. That is, a program that would understand any world. The name of this program is Artificial Intelligence.

How the AI program works?

AI is trying to understand the world by looking for a description of the world. The form of this description is very important. If the description is written in some formal language, then it is very important which formal language we chose.

Can we describe the world through the Turing Machine?

Yes, we can. The question AI is asking is “What should I do?” The answer is policy. This policy is computable because non-computable policies are not the answer. That is, the policy can be presented through a computer program or through a Turing Machine (which is the same).

Can we find a Turing Machine that describes the world?

Yes, we can, if we have an infinitely fast computer, an infinitely long training time, and if there are no fatal errors in this world (i.e., if the agent cannot make a fatal error).

Is there a better way than looking for a Turing Machine?

Yes, AI may not ask itself the question “What should I do?” but start first with the question “What’s going on?”. When it understands what is going on, it will know what to do. That is, we suggest not to look directly for a policy, but first to find a model of the world.

What model will we look for?

We may look for a Markov Decision Process. Here we talk about “Partial Observability” because the case of “Full Observability” is not interesting.

Can we find a model in the form of MDP?

Yes, any world can be represented as an MDP. This model may be infinite, but we will assume that the model is finite and there are no fatal errors (i.e., that there is a path between every two states).

With these assumptions we can effectively find the MDP model of the world, but we still need an infinitely powerful computer and an infinitely long training time.

That we need an infinitely powerful computer doesn’t sound so scary. We can assume that one day there will be a powerful enough computer to find the model. Worse is the requirement of infinitely long training time. If AI is learning infinitely slowly, then our AI will be a slowly developing intellect.

Can we offer a model which can be found without infinitely powerful computer and infinitely long training time?

Yes these will be Event-Driven models. We will make a demonstration by showing how the rules of the game of chess can be described through these models.

Can the game of chess be described by Turing Machine?

Yes, there exist programs which play chess, but finding such a program automatically is virtually impossible.

Can the game of chess be described by MDP?

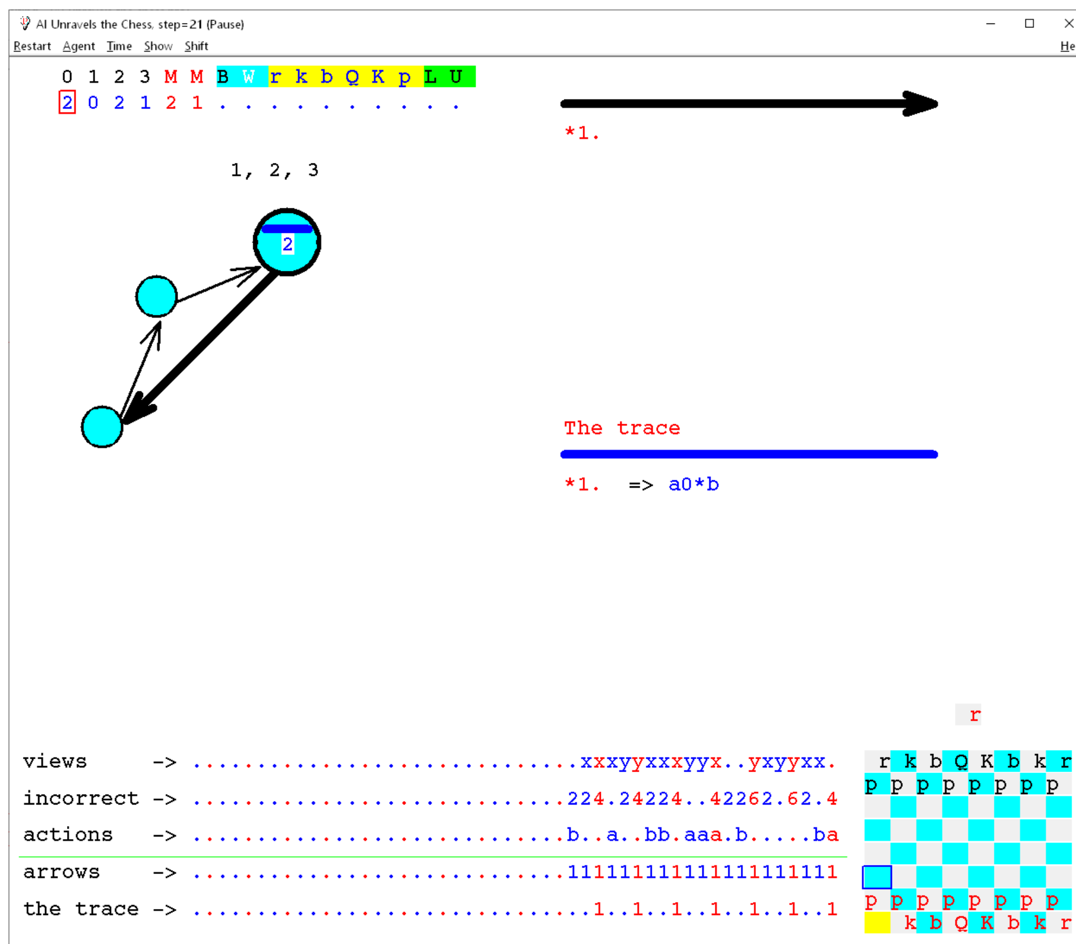
Yes, there exist a finite MDP that describes the game of chess, but its states are as many as the positions in the game. That is, this model cannot be found either.

How will the world of the game of chess look like?

First, it will be a world where the agent does not see everything (We have a “Partial Observability”, the case of “Full Observability” is not interesting).

In this world, the agent will see only one of the squares. This will not be a problem because the agent will be able to move the window (the visible square) and thus view the entire board.

We will demonstrate the following program [2], which is written on Strawberry Prolog [3]:



What the world is?

The agent input and output will consist of three letters. The input will be the letters 0, x, y, and the output will consist of 0, a, b. In addition to the input, the agent also sees which moves are incorrect:

1 – incorrect move is 0.
2 – incorrect move is a.
4 – incorrect move is b.

The agent will be able to move the visible window left and right, as well as up and down. It will also be able to raise the observed piece as well as place the raised piece on the observed square.

How so many actions can be expressed in three letters?

The move of the agent is divided into three. First, he says what the horizontal movement will be - left, right or in place. Second, there will be the vertical movement - up, down, or in place. Third, the agent will say whether he raises/lowers the observed/raised piece or does nothing.

This division of action into three is the first dependency in the world to be found. This first dependency is reflected in the first Event-Driven model. This is a three-state model that counts: 1, 2, 3.

This dependency has one event (True). This is the event that happens at every step. In one of the states, we have a trace that says the move b will be incorrect.

Is this model detectable?

Yes, if you look at the sequence incorrect you will see that every third member is 4 (or 6, which is 4 + 2). That is, something special is happening in one of the states and this will help us to find the model.

What will be the second model?

This will be the model that told us which column we are in. That is, the model that will give us the X coordinate of the observation window.

Here is the second model:

What are the events?

There are two events (going left and going right). These two events occur when the first model is in state 0 and the agent action is a or b, respectively. That is, this model is built hierarchically based on the first model.

What is the trace?

The trace is that it cannot be played to the left. Such is the trace of the first state (which corresponds to the leftmost column of the board). Accordingly, it cannot be played to the right in the last state.

Is this model detectable?

Yes, again thanks to the trace.

A trace with memory

The world can remember for each state of one Event-Driven model what was last happening in that state. What happens constantly is the constant trace, but what happened last is something that can be remembered in the memory of the trace. This memory is a one-dimensional array in which to each state of the model corresponds one cell in the array. We can make a Cartesian product of two models and use the memory of the trace of that Cartesian product. For example, the Cartesian product of the second and third models has memory and this is the board of the game. It can be seen in the following two-dimensional array:

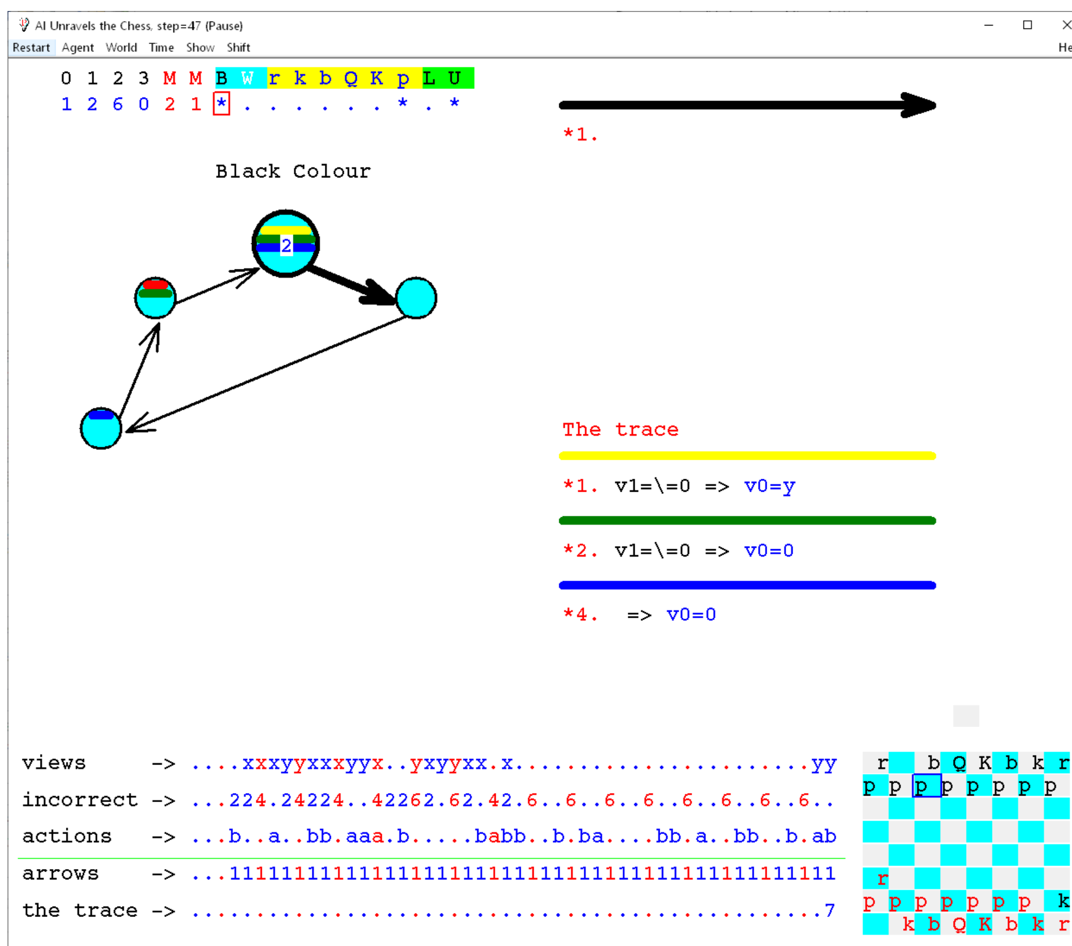
```
AI Unravels the Chess, step=21 (Pause)
Restart  Agent  World  Time  Show  Shift                          Help

0 1 2 3 M M B W r k b Q K p L U
2 0 2 1 2 1 . . . . . . . . . .

1.   up, here => copy(Lifted)
2. down, everywhere => del(Lifted)
3. down, here => copy(mem5(0))
4. move, here => del(Unmoved)

5. move, somewhere, Black is there =>
=> Temp := mem4(there), del(all)

6. move, somewhere, !(Black is there) =>
=> mem4(there) := Temp, del(Unmoved)

7. => ob(now) := mem4(here)

up   = m0=2 & a0=a & m3=0
down = m0=2 & a0=a & m3=1
move = down, !ob(Lifted)

views     -> ...........................xxxyyxxxyyx..yxyyxx.
incorrect -> ...........................224.24224..42262.62.4
actions   -> ...........................b..a..bb.aaa.b.....ba
arrows    -> ...............................................
the trace -> ...............................................
```

What happens in the squares?

Different objects appear there. The objects are not directly visible, but their properties are observed. For example, when a black pawn appears, we observe the "black" property.

How does the property "black" look like?

Properties are some Event-Driven models. We have no idea what these models should look like, so we generated them randomly. For example, the "black" property happened to look like this:



The important thing is that when we observe a black object, this property is active and influences the input that the agent sees. For example, the white property is not active and does not affect the input:

How the world determines the input, given that there are many active Event-Driven models that give different, sometimes contradictory, directions on what the agent will see. The next agent input is determined by voting of the Event-Driven models currently active. This can be seen in the following table:

This program demonstrate us how to describe the game of chess through 4 dependencies, 2 arrays (i.e., the memory of the trace of a dependency) and 10 properties. Properties are also Event-Driven models, but they are not constantly active but only when the property is observed.

Something is missing here. Player moves are random and do not comply with the rules of the game chess. For example, an agent played incorrectly with a rook, and an imaginary player inside the world answers him with a knight, jumping so far that he even took a pawn.

How to prohibit incorrect moves?

To say which move is correct, we need something else. We need to add algorithms. For example, in order for a knight to start moving in the form of the letter “L” he must perform a certain algorithm of movement.

What is an algorithm?

Since dependencies and properties are Event-Driven models, aren’t the algorithms the same? That’s right, algorithms will also be Event-Driven models.

The classic notion of an algorithm is the Turing Machine. Can we represent the Turing Machine through an Event-Driven model? We can represent the head, but we cannot represent the tape. In fact, what is an algorithm? Is it just the head or the head along with the tape?

What was the idea of Turing himself?

He imagined the head of the machine as a person with finite memory, and the tape imagined as the endless amount of paper that that person had to make notes. That is, according to Turing, the algorithm is in the head, and the tape is something external that is used in the execution of the algorithm.

If you have an algorithm for stacking jars, are jars part of the algorithm? No, jars are part of the world and you can apply this algorithm in this world to these jars.

It is the same with the tape, it will not be part of the algorithm, but will be part of the world. In the case of a game of chess, the algorithms will execute on the board. That is, the board will play the role of the tape.

We can represent the head of the Turing Machine with a finite Event-Driven model. What if we allow the Event-Driven model to have infinitely many states? That is, if we allow a Turing Machine whose head has infinitely many states. We will call such an algorithm theoretical. A real algorithm will be one in which the states of the head (or of the Event-Driven model) are finitely many.

What is the conclusion?

The conclusion is that through Event-Driven models we can describe a complex world. In doing so, we can describe it simply enough for this description to be found automatically. That is, Event-Driven models are the language (format) for describing worlds that AI will use.

References

[1] Dimiter Dobrev (2019). Before we can find a model, we must forget about perfection. *viXra:1902.0245.*

[2] Dimiter Dobrev (2019). AI Unravels the Chess. *http://dobrev.com/software/AI_unravels_the_chess.pro and http://dobrev.com/software/AI_unravels_the_chess.h* .

[3] Dimiter Dobrev (2019). Strawberry Prolog, version 3.1. *http://dobrev.com/* .