SECOND ATTEMPT TO BUILD A MODEL OF THE TIC-TAC-TOE GAME [1]

Dimiter Dobrev

Abstract: We want to make a program which can play any game or in other words we want to make AI. It is impossible to include in this program the rules of all games and that is why our program should be able to find these rules by itself. We cannot solve this problem in the general case. So, our first task will be to make a program which is able to find the rules of the Tic-Tac-Toe game. Even this task is too complicated. So, first we will try to find these rules manually and this will help us make a program which is able to find these rules automatically.

Keywords: Artificial Intelligence, Games, Automated reasoning.

ACM Classification Keywords: I.2.0 Artificial Intelligence - Philosophical foundations

Conference: The paper is selected from XIVth International Conference "Knowledge-Dialogue-Solution" KDS 2008, Varna, Bulgaria, June-July 2008

Introduction

We are trying to build a formal model of one particular game. We need such a model in order to predict the future. For example, look at the third position shown on figure 1. At this position you see that if you play in the centre then you will win but if you do not play in the centre then probably you will lose. This means that you have in your head a model of the Tic-Tac-Toe game and you can predict the future and say what will be the consequences of your next move.

We try manually to find a model of the Tic-Tac-Toe game which will give us the possibility to play this game successfully. If we write a simple program which can recognise which move is wining and which one is losing then this program will be a model of the game. Anyway, we do not like this model because it is not easily discoverable. First reason for that is that the set of all programs is too huge. The second reason is that the program models are not easily checkable. This means that if you have one program which is a model of the game it is not easy to check this fact because for this you should play some time following the recommendations which this model gives and after that judge how good this model is on the basis of the results achieved in the testing period.

So, we are looking for an easily discoverable model. Do not forget that we need a model which can be found automatically.

There is one more reason why programs are not good candidates for such models. The programs are not easily modifiable. If you make a small random modification in one program then as a result you will receive a program which will work in totally different way and in most cases it will not work at all. In most optimisation tasks we try to find the best solution by making small modifications. Let's take for example the simplex algorithm which is for solving the linear programming problem, or the "go up" algorithm for finding the highest place, or the process of the evolution in nature where the child is a small modification of its parents.

How will our easily modifiable model look like? It will be a logic theory which consists of set of assumptions (axioms). A modification will be to add or to remove one assumption.

[1] This publication is partially supported by the Bulgarian Ministry of Education (contract БОЕ 4-02/2004 г.)

Formalisation of the game

Before finding a formal model of the game we have to formalise it. This means to represent the real game as a mathematical object. In this case this mathematical object will be the set of all possible sequences of inputs and outputs. Of course, the best representation of this set is the tree of all possible moves. Here when we say moves we mean our moves and the moves of our opponent.

You see that for the formalisation of the game we need formalisation of our opponent. Really, when you play a game you try to build a model both of the game and of your opponent. So, when you play you try to understand your opponent and to predict his behaviour.

From formal point of view, playing with different opponents is playing a different game. This is because different opponents play different moves and that is why the tree of all possible moves is different.

In order to formalise the game we need to fix the opponent. Let us assume that our opponent makes line and wins if he can do this on the next move. Otherwise he plays a random move choosing randomly from all possible correct moves with equal possibility for each of them to be chosen.

Note. If we have a game and a fixed opponent and if we try to make a model of both of them do we need to separate the model of the game from the model of the opponent? For example, if our opponent never starts with a move in the centre then should this fact be in the model of the game or should it be in the model of the opponent. The answer is that we do not care why the opponent has certain behaviour. Maybe this is part of the rules of the game or maybe not. In any case, if certain behaviour is fact then we can use this fact no matter what is the reason behind it.

There is one case when it is good to separate the model of the game from the model of the opponent. This is the case when we try to see the world through the eyes of our opponent. We will not try to do this because it is a difficult task. They say that children younger than three years cannot do this, so our program will not be able to do this either.

One more reason for this is that we will assume that each time we play the first move. So, the world is not symmetric for us and for our opponent and this makes it more difficult to look at the world through the eyes of our opponent.

In order to finish the formalisation of the game we have to say what is the information which we input and output on every move. Let us assume that on every move we input the game status (i.e. what we have on the board) and on every move we output the coordinates of the cell where we put a cross.

So, we want to represent the game as sequence of inputs and outputs like this:

$$a_0, b_0, a_1, b_1, a_2, b_2 \ldots$$

On figure 1 you can see an example of a game.

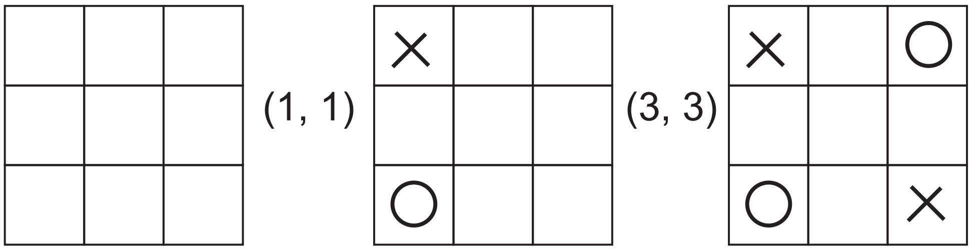

Fig. 1: Part of a game

The first attempt

On the basis of this formalisation we made our first attempt to build a model of the Tic-Tac-Toe game. Let's start with the size of the input and the size of the output according to this formalisation. The input includes the situation on the board. These are nine cells and to describe the situation we will use 18 bits (two bits per cell). Also we need three additional bits which we will call "victory", "loss" and "bad move". These three bits give us information about the result of our activity. For example, if we win the bit "victory" will be on and if the game is draw then both "victory" and "loss" bits will be on. The bit "bad move" will be on when we try to do something forbidden (like putting a cross in a cell which is not empty). These three additional bits give us the purpose of our program. It is not sufficient to have a model of the game because if you do not have a purpose you cannot distinguish good from bad and you cannot choose your next move even if you know perfectly well what will happen if you choose this move.

> **Note.** In the terms used in our definition of AI [1, 2, 3] the purpose of our program is called "the meaning of life". Here this purpose is clear. It is to achieve more victories and fewer losses. In this paper it is almost useless to explain what the purpose of our program is because it is obvious that when we play the Tic-Tac-Toe game our purpose is to win. Anyway, these explanations are not useless because in other games the purpose may be difficult for defining. Let's take for example the real life of a human being. In this case if we have a clear purpose (or a clear meaning of life) we can look at the real life as a game.

So, our input is 21 bits (18 for the state of the board plus three additional bits). Our output is 4 bits (two for the "x" and two for the "y" coordinate). Now we will try to make simple implications of the type: "If you see this and do that then on the next step you will see this." Here is an example of such simple implication:

$$p_{11} \,\&\, \neg\, out_{x1} \,\&\, \neg\, out_{x2} \,\&\, \neg\, out_{y1} \,\&\, \neg\, out_{y2} \Rightarrow bad\_move$$

The meaning of this simple implication is that if you have a cross at position (1, 1) and if you play at this position then on the next move you will see the bit "bad move" on.

How many are these simple implications? The maximum length of them is 46 (two times 21 plus 4). So, their number is 3 to the power of 46. Of course, we do not need all of them but only these which are true and even only small fraction of them which are essential. The number of these essential implications is millions and the first program which we have made in order to decide the problem cannot manage to proceed with so many implications. Anyway, we have made a more sophisticated program which keeps these implications in a tree structure, which allows it to proceed with sufficiently many implications in a reasonable time. You can find this program in [10].

The idea of the first attempt is that in the set of the simple implications which are true there is coded the information about the experience (about the first moves which will be used for our education). Of course, this is not all the information from the experience but this part of it which is essential. So, on the basis of this set of simple implications we can say which move is bad. For example, the implication which is described above, says us that if we have a cross at position (1, 1) we cannot play at this position.

Unfortunately, this first attempt was a complete disaster. Really, the essential implications were millions but this was only a technical problem, which was solved in [10]. The real problem was the number of steps which we have to make in order to collect enough experience. In [10] you will see that after 20,000 steps the program almost stops to make bad moves. This means that the time (the number of steps) for education is extremely big. Let's take an example connected with human beings. With people boys study slower than girls. Anyway, this is not a problem because they study just a little bit slower. In our case we have a serious problem because the time for education is so huge that it is practically infinite.

Where is the problem? This model is too stupid and here you cannot apply neither analogy nor something similar to analogy. Here the problem comes from the formalisation of the game. On one hand, the input is too big (21 bits) and on the other hand, we have made the wrong assumption that we see the full status of the game. For example, in real world this assumption is not true because we do not see the full status of the world (Actually, we cannot see behind our back.) The assumption that we see (receive as an input) the full status of the game is possible only with very simple games.

The conclusion is that the next attempt to build a model of the Tic-Tac-Toe game will start with a new formalisation of the game.

First published attempt

The first attempt was a complete disaster and that is why it is not published. The next attempt was much better and that is why it is published in [4, 5].

As we've said, this attempt starts with a new formalisation of the game. Here we reject the assumption that we see the full status of the game. Now we will assume that we see only one cell (the current cell). In figure 2 you will see the eye which pinpoints the current cell. In this case the input is only one cell which is two bits. Of course, we have also three additional bits as before. The output is also different. Before we had nine moves (to put cross in one of the cells). Now we have six moves and they are to move the eye in the four possible directions, to put a cross at the current cell and to start a new game.

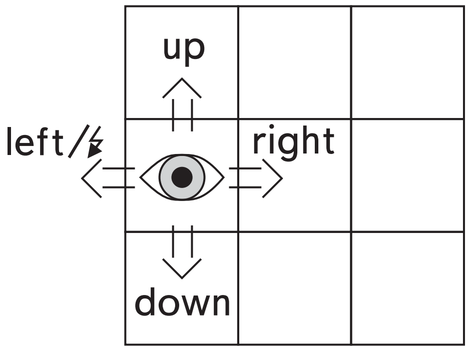

Fig. 2: The new formalisation

In this new formalisation the concept of the move is changed. Before the move was to put a cross somewhere but now it is to move through the board or to put a cross. The main change is in the concept of what we see. Before the assumption was that we see everything but now we see only a small part of the game status and we have to imagine what the status of the game is. So, now building of the model is a much more interesting task.

On the basis of this new formalisation in [4, 5] there was made an attempt to build a model of the Tic-Tac-Toe game. This model included three main parts: simple implications, FSMs (finite state machines) and first-order formulas. The connection between these three parts was not clear and the attempt from [4, 5] did not give us a working model of the game.

The second attempt

That is the reason why in this paper we will make a new attempt to build a model of the Tic-Tac-Toe game. The basis of this new attempt will be the first-order logic with types. From the theoretical point of view, there is no difference between this logic and the common first-order logic but types give us much bigger expressiveness.

What is the difference between the first-order logic and first-order logic with types? In the first case the universe is one not empty set but in the second case the universe is a union of several non-intersected sets. The second difference is that in the first case the relation and the function symbols have only valence but in the case with types every argument has a type.

In our first-order logic with types we will have one countable set *T* which will correspond to the time and several finite sets which will correspond to the states of some FSMs. In this paper we will mention only the sets *X* and *Y* which have three elements each and which correspond to the coordinates of the eye.

What will be the structure of the set *T* or what will be the structure of the time. We will have two function symbols "next" and "previous". These symbols will have one argument of type *T* and will return object of type *T*. We have to decide whether to make *T* isomorphic to the natural numbers or to make it isomorphic to the integer numbers. In other words, to introduce one constant for the first moment or not. The better choice is to make the time isomorphic to integers because this model is simpler. Really, we have a first moment but we cannot use this moment in order to make conclusions. You cannot conclude something like "When I am born they give me milk" because you have not enough statistical information for such conclusion. Even if you have such rule you cannot use it because you will not be born again.

Let's see what one simple implication will look like. For example, "If you see a cross you cannot put a cross":

$$I_x(T)\ \&\ put\_cross(T) \Rightarrow bad\_move(next(T))$$

Now, let us take M_x. This is the FSM which was described in [4].

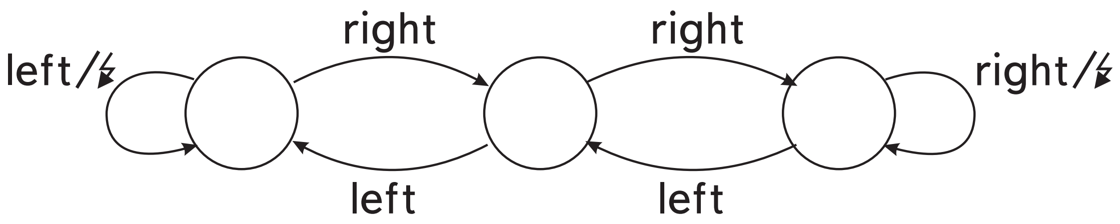

Fig. 3: The FSM which gives the "x" coordinate

M_x is essential part of our model because it corresponds to the "x" coordinate of the eye. M_x can be included in our model. Here is the description of one of its arcs:

$$x(T)=x_1\ \&\ right(T) \Rightarrow x(next(T))=x_2$$

Here *x* is a function symbol which has one argument of type time and which returns an object of type *X*. The meaning is that it returns the current "x" coordinate of the eye or the current state of M_x. The constants x_1 and x_2 will correspond to two of the states of M_x. Of course, these constants will be of the type *X*.

We will notice that the simple implications and these FSMs are easily discoverable. (This is important because we are looking for a model which can be found automatically.) In [10] you can see that the simple implications can be generated automatically and if the size of the input and output is not too big this can be done without combinatorial explosion. In [8, 9] you can see that M_x can be found through the method of suns. The idea of this method is that FSMs are too many but every FSM can be decomposed in simpler objects, which we call suns. You will receive such object if you observe only one letter in your FSM or observe only the arcs labelled with this letter. The result is one or several simple directed graphs with one cycle and several paths which flow in this cycle. Such graph looks like the picture of the sun which children use to draw. For example, if we take M_x and observe it only on the letter "right" then we will receive one loop (which is a cycle with length one) and one path with length two which flows in this cycle. In this way we will receive one "sun" which includes three arcs. This sun is easily discoverable because one of its arcs gives bad move each time and the other two give correct move each time.

As we said, for us it is important to make a model which is able to give us a mental picture of what we cannot see. The FSMs M_x and M_y give us the idea where we are at the moment (actually we cannot see directly the coordinates of the eye). The next information which we have to include in the model is what the situation on the board is. For this we need first-order formula like this:

$$p(x(T), y(T), T) \Leftrightarrow I_x(T)$$

Here we did not say anything essential. In our input we have five bits and two of them are nameless (we do not know nothing about them). It is normal to try to say something about these two bits. One of them is $I_x(T)$. It is normal to assume that $I_x(T)$ depends on something other than T or to assume that $I_x(T)$ is a projection of some relation which has more arguments. Actually, p is this relation and it depends on the time and on the current coordinates. Existence of such relation would be not interesting at all if there was not the following formula:

$$\neg\, put\_cross(T)\ \&\ \neg\, new\_game(T) \Rightarrow (\forall X\ \forall Y\ (p(X, Y, T) \Leftrightarrow p(X, Y, next(T))\,))$$

This formula gives us the stability of the relation p. This formula is not easily discoverable but we are looking for stable relations and this means that we are looking for formulas which describe stability.

At the end we will show what the formulas which describe the victory look like.

$$(\exists X\, \forall Y\ p(X, Y, T)) \Rightarrow victory(T)$$
$$(\forall X\, \forall Y (d_1(X, Y) \Rightarrow p(X, Y, T)\,)) \Rightarrow victory(T)$$

The first of these formulas says that if we made a vertical line we won. The second formula says that if we made the first diagonal we won. The first diagonal is a relation between X and Y but we have 512 relations between X and Y. Is this relation an easily discoverable one? Yes, because FSMs M_x and M_y are isomorphic and there are only two isomorphisms between them and these isomorphisms are the diagonals. So, this relation is easily discoverable because it is special.

Modification of the method of resolution

Even if we have a model of the game, we need a method for proving formulas in the first-order logic in order to make a program which can play successfully on the basis of this model. Of course, we have such a method and this is the method of resolution.

This method has one serious disadvantage, which we have to fix in order to make AI which is capable to work in acceptable time. In [6, 7] you can see that if we do not worry about the combinatorial explosion then it is easy to make AI (which is useless because it cannot work in acceptable time). If we want to make useful AI then we have to worry about its efficiency.

The problem, which is to be mainly blamed for the bad efficiency, is that the method of resolution starts every time from the beginning. In this way it is practically impossible to prove anything complicated by this method.

We would like to make such modification that allows constructing a database of proven disjuncts. Of course, only tautological disjuncts are true without any propositions but these disjuncts are not interesting. That is why we would like to build a semilattice of different logic theories which are to contain interesting implications. For example, if we want to prove $\varphi \Rightarrow \psi$ then we can find the logic theory where φ is a proposition and to check if ψ is an already proven implication in this theory.

Unfortunately, this works only for formulas which we can prove without skolemization. The main reason that the resolution starts every time from the beginning is that first we make the skolemization and on the next step we make the resolution. In order to improve the efficiency of the resolution we have to allow the skolemization and the resolution to work in parallel. This will be impossible if skolemization gives random names of the objects because if it gives a name to one object twice then it will give two different names to it but we would like the name on the second time to be the same.

So, what type of systematic names should our new skolemization use? For us the best choice is the system proposed by David Hilbert. For every formula $\varphi(x)$ he defined an object $\tau_x\varphi(x)$ which satisfies $\varphi(x)$ if there exists an object which satisfies $\varphi(x)$ (description of these Hilbert's terms you can find in [12]). An important fact is that if $\varphi(x) \Leftrightarrow \psi(x)$ then $\tau_x\varphi(x) = \tau_x\psi(x)$. This means that for one and the same object we give one and the same name.

For example, we want to prove $\forall x(\varphi(x) \Rightarrow \psi(x))$. If we do this in the old way the skolemization will give a random name to the object which satisfies $\varphi(x) \,\&\, \neg\, \psi(x)$. After that, the resolution will prove that the existence of such object leads to contradiction. It will be much faster if we have already developed the theory of $\exists x\ \varphi(x)$ and if in this theory $\psi(x)$ where x is $\tau_x\varphi(x)$ is already proven and this will be sufficient. Really, we want to prove that in this theory is true $\psi(x)$ where x is $\tau_x(\varphi(x) \,\&\, \neg\, \psi(x))$ but there is a connection between $\tau_x\varphi(x)$ and $\tau_x(\varphi(x) \,\&\, \neg\, \psi(x))$. For the second object we know more, so everything which we can say for the first object we can say also for the second one.

This modification of the method of resolution is only an idea and we need a lot of work in order to make a real system which is based on this idea.

Bibliography

[1] Dobrev D. D. AI - What is this. In: PC Magazine - Bulgaria, November'2000, pp.12-13 (in Bulgarian, also in [11] in English).

[2] Dobrev D. D. AI - How does it cope in an arbitrary world. In: PC Magazine - Bulgaria, February'2001, pp.12-13 (in Bulgarian, also in [11] in English).

[3] Dobrev D. D. A Definition of Artificial Intelligence. In: Mathematica Balkanica, New Series, Vol. 19, 2005, Fasc. 1-2, pp.67-74.

[4] Dobrev D. D. Testing AI in One Artificial World. In: Proceedings of XI-th International Conference KDS 2005, June, 2005 Varna, Bulgaria, pp.461-464.

[5] Dobrev D. D. AI in Arbitrary World. In: Proceedings of 5th Panhellenic Logic Symposium, July 2005, University of Athens, Athens, Greece, pp. 62-67.

[6] Dobrev D. D. Formal Definition of Artificial Intelligence. In: International Journal "Information Theories & Applications", vol.12, Number 3, 2005, pp.277-285.

[7] Dobrev D. D. Formal Definition of AI and an Algorithm which Satisfies this Definition. In: Proceedings of XII-th International Conference KDS 2006, June, 2006 Varna, Bulgaria, pp.230-237.

[8] Dobrev D. D. Two fundamental problems connected with AI, XII International Conference "Knowledge-Dialogue-Solution", June 2007, Varna, Bulgaria.

[9] Dobrev D. D. The "sunshine" Method for Finding Finite Automata, Trends in Mathematics and Informatics, July 2007, Sofia, Bulgaria.

[10] Dobrev D. D. Generator of simple implications, http://www.dobrev.com/AI/app4.html

[11] Dobrev D. D. AI Project, http://www.dobrev.com/AI/

[12] Bourbaki N. Theory of sets, Chapter 1.

Authors' Information

Dimiter Dobrev *– Institute of Mathematics and Informatics, BAS, Acad.G.Bonthev St., bl.8, Sofia-1113, Bulgaria; P.O.Box: 1274, Sofia-1000, Bulgaria; e-mail: d@dobrev.com*