AIP Conference Proceedings

Mathematical algorithms for artificial intelligence

Cite as: AIP Conference Proceedings **2172**, 110015 (2019); https://doi.org/10.1063/1.5133618
Published Online: 13 November 2019

Roumiana Ilieva, Kiril Anguelov, and Yoto Nikolov


View Online

Export Citation

Lock-in Amplifiers
... and more, from DC to 600 MHz

Watch

AIP Publishing

© 2019 Author(s).

Mathematical Algorithms for Artificial Intelligence

Roumiana Ilieva[1, a)], Kiril Anguelov[2, b)] and Yoto Nikolov[3, c)]

[1]*Faculty of Management, Technical University of Sofia, Bulgaria.*
[2]*Dean of the French Faculty of Electrical Engineering, Technical University of Sofia, Bulgaria.*
[3]*PhD School at French Faculty of Electrical Engineering, Technical University of Sofia, Bulgaria.*

[a)]Corresponding author: rilieva@tu-sofia. bg
[b)]ang@tu-sofia. bg
[c)]nikolov_7@yahoo.com

Abstract. Speaking for artificial intelligence and mathematics, some will say that they are two different fields. But we can assure you that mathematical models are its foundations, especially the main subset machine learning. Investigating these models and algorithms, we have the possibility to predict the abilities and limitations of artificial intelligence. Utilization of super-recursive algorithms for proving theorems, makes many such systems complete and decidable. Here we will discuss the main differences of recursive and super-recursive algorithm in particular Turing machine and inductive Turing machine. Our main statement is that artificial intelligent based on inductive Turing machine is much more powerful and will greatly enhance machine learning capabilities. As theorem proving and logic used commonly in artificial intelligence, these results will have a very important role in its implications.

INTRODUCTION

Artificial Intelligence(AI) can be seen as interactive intelligence realized by artificial means. Nowadays modern computers are the most powerful means for achieving this intelligence. This implies that AI is a computer program that allows computer to realize intellectual activity and express intelligent behavior, learn from its mistakes and make the needed calculation to be more precise and efficient in the next similar scenario. However, there are some severe controversy about definitions of intelligence and what is accepted to be an intelligent behavior, different people different perceptions. We are not going to discuss these differences. Our goal is to understand how underlying to mathematical technique, will find what computers can do and what cannot with respect to abilities of people. There are three main approaches to use in this article. The first one is empirical. The essence is formulated as this; Let us build more and more sophisticated computers, design more and more advanced software for them, and observe their functioning, analyzing results of our observations.[1] This approach is represented by the famous Turing's test for AI as well as by knowledge engineering aimed at the development of expert systems and systems like the Logic Theorist.[2] The second approach is philosophical and methodological. It is based on speculative reasoning about computers, human intelligence, and their interrelations. As an example of a problem considered in this approach we can take the problem whether mind or brain is some kind of computer or is something much more complex. Dreyfus in the late 1973 presented different aspects of this approach. The third approach is mathematical. Its target is the construction of mathematical models for computers and their software. This provides for explication of AI capabilities through a study of these models and evaluation of those aspects that can be considered intelligent.

Proceedings of the 45th International Conference on Application of Mathematics in Engineering and Economics (AMEE'19)
AIP Conf. Proc. 2172, 110015-1–110015-8; https://doi.org/10.1063/1.5133618
Published by AIP Publishing. 978-0-7354-1919-3/$30.00

In modern mathematics, different types of algorithms model computers and their software.[3] Consequently, the power of algorithms has been used as a measure for AI capabilities. In addition to this, because upper models of AI represent sufficiently broad classes of relevant structures, these models can comprise different types and kinds of intelligence of human beings. It gives a framework within which sophisticated systems of AI may be specified, designed, analyzed, and verified in a systematic rather than ad-hoc manner. A coherent application of this approach would result in the most promising payoffs. It is essential to remark that although this approach is based on methods of mathematics, its adherents explicitly speak and write, as a rule, what computers in themselves are while in reality they are dealing with mathematical models or even with informal images of actual computers. Consequently, they substitute computers for their models like Turing machines, RAM (Random Access Machine) and others and estimate power of computers by capabilities of one or another mathematical model. However, any model gives only some approximation to a real computer. That is why, before using a model as a measurement device, it is necessary to find adequacy of the model. Here we speak for a measure of precision of the estimations that are made by means of this model. Another remark serves to attract our attention to the fact that evaluation of computers has to take into account their hardware, software and usage. What concerns hardware and software, now these components are included in the evaluated mainframe, while usage is often neglected. As it is formal methods are used to describe only properties of hardware and software. At the same time, it is a great difference between using a computer as a calculator and simulating complex processes with the help of the same computer. As a consequence, if a new way of utilization of computers emerge, it can change drastically their capabilities and limitations. The three components that are basic for evaluation correspond to the three fundamental aspects of reality. Hardware relates to the physical actuality. Software, including its algorithmic constituents, represents the structural design. Usage reflects mental side of being. Complete measurement of computer capabilities has to encompass all these aspects.

MATHEMATICAL MODELS: RECURSIVE AND SUPER-RECURSIVE ALGORITHMS. DIFFERENCE BETWEEN UNIVERSAL AND INDUCTIVE TURING MACHINE.

People use algorithms all the time, without even knowing it. In many cases, people work, travel, cook, and do many other things according to algorithms. For example, we may speak about algorithms for counting, algorithms for going from Sofia to Plovdiv or to some other place, algorithms for chip production or for buying some goods, products or food. Famous Gödel incompleteness theorem [4] for formal systems is true only when conventional algorithms are used for proofs and this explains how people solve problems related to incomplete systems with undecidable properties

The first superrecursive algorithms is, limit recursive functions. They were shown to the public space in 1965 by Gold and Putnam, where the idea of “inductive inference” was born.[5] A form of limit recursive and partial limit recursive functions. In other words, they manage to create the concept of computability in a limit and a model for computations in the limit. At first these algorithms were not considered real, because no computing device was able to reproduce a functional compute model, like the one using conventional algorithms. Here we introduce a new and better way for computation. Supperrecurisve algorithms will radically change the functioning of computers, devices, and will open a new perspective for computation.

Limit Turing machines, constitute the most powerful class of the super-recursive algorithms known so far. Here we consider a special class of limit Turing machines. So called I TM. It is demonstrated that I TM with recursive memory are such super-recursive systems that are the closest to the recursive algorithms.

Universal Turing machine (UTM) was the study introduced by Hopcroft and Ullman in 1979 [6]. UTM is an interpreter for all Turing machines.

Here is shown how UTM works:

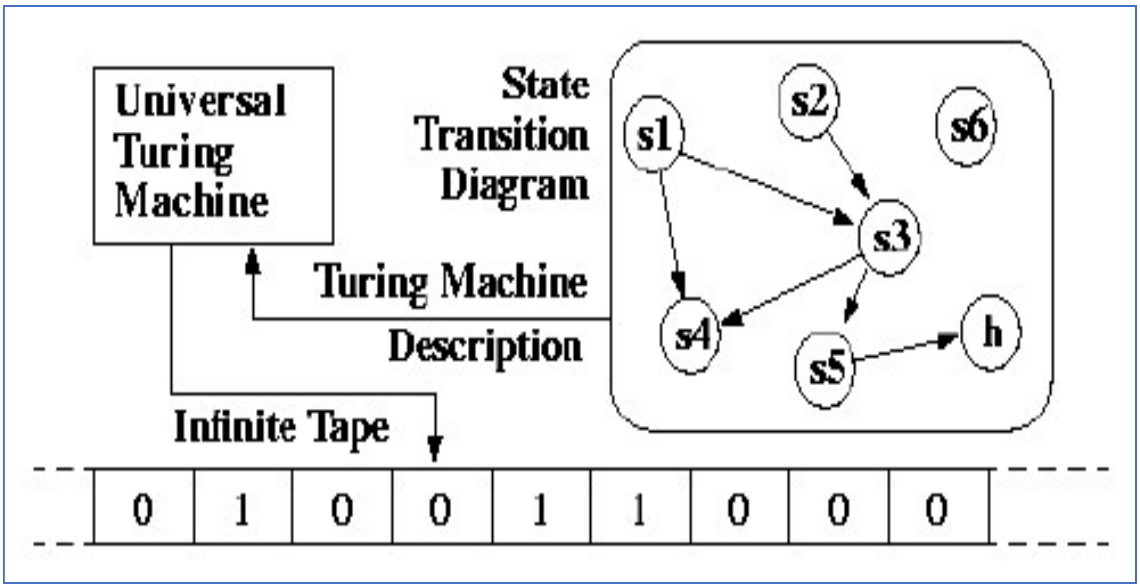

FIGURE 1. Universal Turing Machines State Transition [7]

TABLE 1. Universal Turing Machines State Transition Table

State table for 3 state, 2 symbol busy beaver									
Tape symbol	State S1			State S2			State S3		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	S5	**S2**	1	S4	**S1**	1	S4	**S2**
1	1	S4	**S3**	1	S5	S2	1	S5	**HALT**

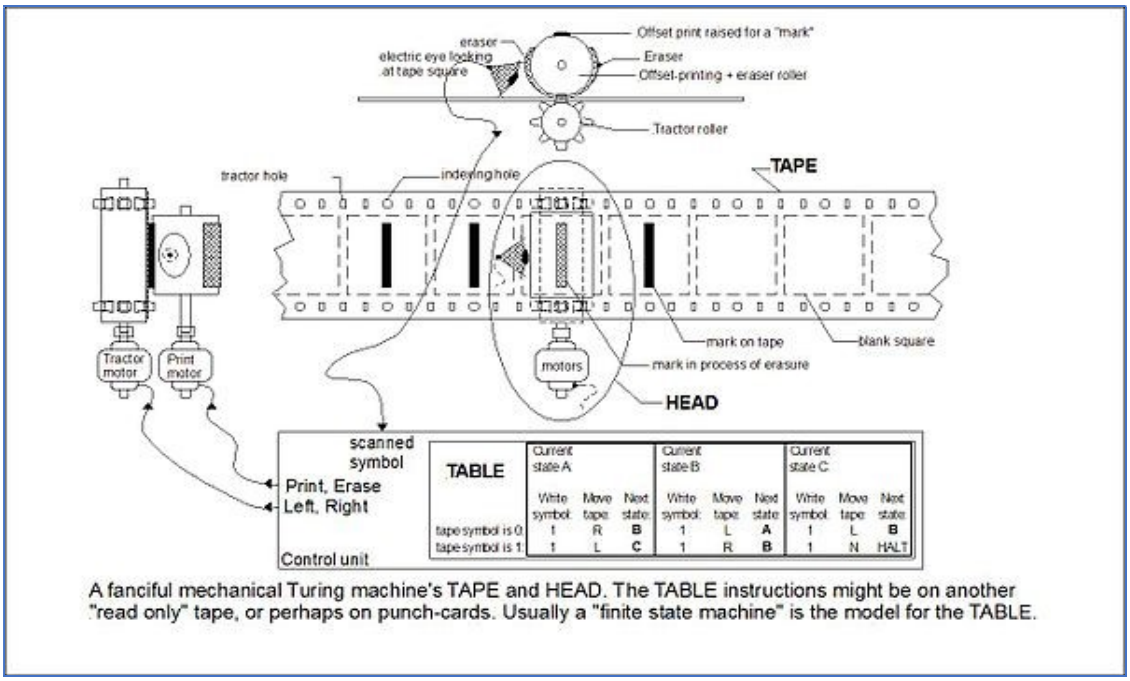

FIGURE 2. Universal Turing Machines Work Modification [8]

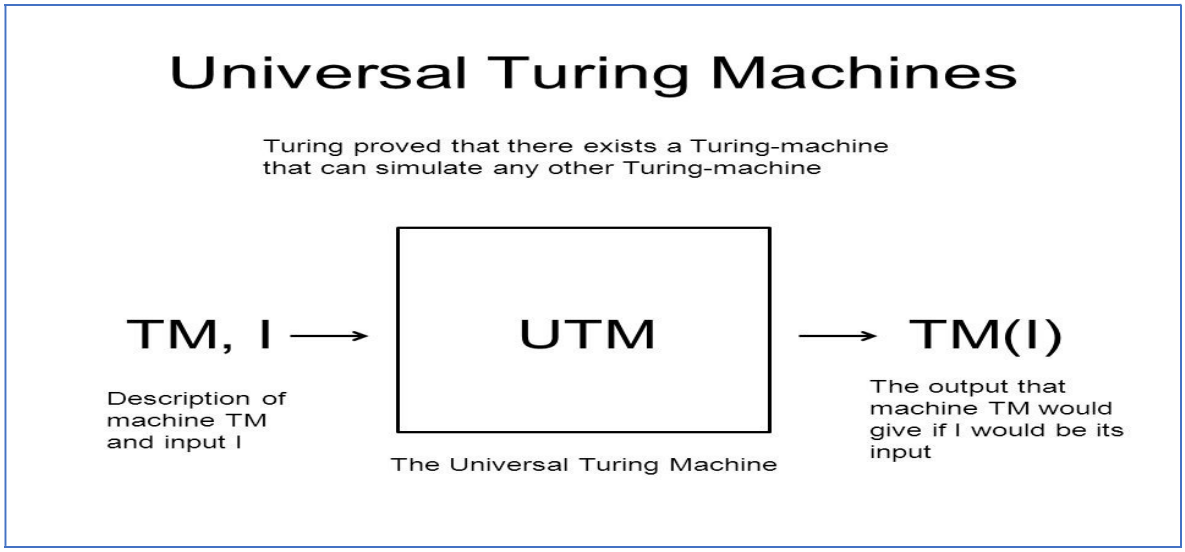

FIGURE 3. Universal Turing Machines In -> out [9]

A UTM can be described as Turing machine whose programming simulates other TMs. The input to the UTM is a description of a Turing machine TM and an input for TM, and the UTM simulates TM on that input.

It's universal in the sense that, for any problem that can be solved by Turing machines, you could either use a Turing machine that directly solves that problem, or you could use a UTM and give it the description of a TM that directly solves the problem

Implementation of Turing Machine

```
package body Turing is

   function List_To_String(L: List; Map: Symbol_Map) return String is
      LL: List := L;
      use type List;
   begin
      if L = Symbol_Lists.Empty List then
         return "";
      else
         LL.Delete First;
         return Map(L.First Element) & List_To_String(LL, Map);
      end if;
   end List_To_String;

   function To_String(Tape: Tape_Type; Map: Symbol_Map) return String is

   begin
      return List_To_String(Tape.Left, Map) & Map(Tape.Here) &
        List_To_String(Tape.Right, Map);
   end To_String;

   function Position_To_String(Tape: Tape_Type; Marker: Character := '^')
                               return String is
      Blank_Map: Symbol_Map := (others => ' ');
   begin
      return List_To_String(Tape.Left, Blank_Map) & Marker &
        List_To_String(Tape.Right, Blank_Map);
   end Position_To_String;

   function To_Tape(Str: String; Map: Symbol_Map) return Tape_Type is
      Char_Map: array(Character) of Symbol := (others => Blank);
      Tape: Tape_Type;
   begin
      if Str = "" then
         Tape.Here := Blank;
      else
         for S in Symbol loop
            Char_Map(Map(S)) := S;
         end loop;
         Tape.Here := Char_Map(Str(Str'First));
         for I in Str'First+1 .. Str'Last loop
            Tape.Right.Append(Char_Map(Str(I)));
         end loop;
      end if;
      return Tape;
      end To_Tape;

   procedure Single_Step(Current: in out State;
                         Tape: in out Tape_Type;
                         Rules: Rules_Type) is
      Act: Action := Rules(Current, Tape.Here);
      use type List; -- needed to compare Tape.Left/Right to the Empty List
   begin
      Current := Act.New State;      -- 1. update State
```

Implementation of Turing Machine … (continued)

```
      Tape.Here := Act.New Symbol;  -- 2. write Symbol to Tape
      case Act.Move To is           -- 3. move Tape to the Left/Right or Stay
         when Left =>
            Tape.Right.Prepend(Tape.Here);
            if Tape.Left /= Symbol_Lists.Empty List then
               Tape.Here := Tape.Left.Last Element;
               Tape.Left.Delete_Last;
            else
               Tape.Here := Blank;
            end if;
        when Stay =>
            null; -- Stay where you are!
         when Right =>
            Tape.Left.Append(Tape.Here);
            if Tape.Right /= Symbol_Lists.Empty List then
               Tape.Here := Tape.Right.First Element;
               Tape.Right.Delete First;
            else
               Tape.Here := Blank;
            end if;
      end case;
   end Single_Step;

   procedure Run(The_Tape: in out Tape_Type;
                 Rules: Rules Type;
                 Max Steps: Natural := Natural'Last;
                 Print: access procedure (Tape: Tape_Type; Current: State)) i
s
      The_State: State     := Start;
      Steps:     Natural   := 0;
   begin
      Steps := 0;
      while (Steps <= Max_Steps) and (The_State /= Halt) loop
         if Print /= null then
            Print(The_Tape, The_State);
         end if;
         Steps := Steps + 1;
         Single_Step(The_State, The_Tape, Rules);
      end loop;
      if The_State /= Halt then
         raise Constraint_Error;
      end if;
   end Run;

end Turing;
```

Approximate Outputs of Accelerated Inductive Turing Machines

```
Algorithm:
Set n:=1;
while time =< 1 min do:
  write "1" on the designated output square the symbol 1
(TRUE);
End while;
While time =< 1/2^n  and overallTime =< 1 min do:
  If sumOfTwoPrimes(2(n+1)) <> True then:
  write "0" in the output tap at the actual position of
the head;
  Exit;
  End if;
n = n+1;
End While;
```

One of the main advantages of an inductive Turing machine over an ordinary one is that it does not need to stop to produce the result of computation. This result also is produced after some computation of the one part of output memory and does not affect the rest of the memory. Another advantage of the inductive Turing machine is the ability to obtain results even in the case, when the operation of the device realizing the algorithm is not terminated. Final result will be a better performance and the abilities of this system of algorithms to be increased to a great extent, demonstrating that AI might be able to solve much more complex problems that it was assumed, when a model of AI is using only recursive algorithms.

The newly emerging field of the usage of super recursive algorithms belongs to both mathematics and computer science. It gives a glimpse into the future of computers and AI. In addition, super recursive algorithms provide more adequate models for modern computers, and embedded systems. Consequently, we hope that this theory of super recursive algorithms will, in the end, provide new insight and different perspectives on the utilization of computers, software, and artificial intelligence.

BENEFITS OF SUPER-RECURSIVE METHOD

It is important to understand that upper modeling of AI gives not only negative results demonstrating what AI cannot do. In addition to this, such modeling shows how better to utilize computers to achieve higher level of intelligence. Knowledge that computers work not according to the recursive pattern but employing super-recursive paradigm can help to write more efficient programs for AI also to organize functioning of computer systems in AI. This newly emerging field of the theory of super-recursive algorithms gives a glimpse into the future of computers and AI. In addition, super recursive algorithms provide more adequate models for modern computers, and embedded systems.

CONCLUSIONS

After some detailed research we come to the following conclusion. If realization of AI is based on inductive Turing machines, its abilities to solve problems are much greater than the capabilities of AI substantiated by recursive algorithms. Really, recursive algorithms can solve problems related only to the first two levels of the arithmetical hierarchy, while inductive Turing machines encompass the whole infinite hierarchy. Consequently, inductive Turing machines extend drastically the boundaries of AI. Many problems that are now considered non-solvable for computers are tractable for inductive Turing machines and consequently, for those computers that implement the super-recursive computability. Moreover, it was proved that inductive Turing machines are in many cases more efficient than an arbitrary type of recursive algorithms. It is important to understand that upper modeling of AI gives not only negative results demonstrating what AI cannot do. In addition to this, such modeling shows how better to utilize computers to achieve higher level of intelligence. Knowledge that computers work not according to the recursive pattern but employ super-recursive paradigm can help to write more efficient programs for AI as well as to organize functioning of computer systems in AI. We may compare this situation with the knowledge that the Earth is not flat but round like a ball. As we know from history, people believed that the Earth was flat for a long time. If they did not go far from the place where they lived, the difference between a flat and round Earth did not matter a bit in their everyday life. However, in a long distance trip, knowledge of the actual form of the planet helps to choose much better roots. It is especially important when people fly by planes. What concern space traveling, correct knowledge of the form of the Earth is absolutely necessary. In a similar way, knowledge of properties of super-recursive algorithms and programs, in general, and of inductive Turing machines, in particular, can essentially help to write better programs. For example, it is proved that in many cases programs for inductive Turing machines may be shorter than programs for conventional Turing machines. It means that programs utilizing inductive approach may be shorter than programs that implement recursive scheme. It is possible to ignore the differences between inductive and recursive patterns when we solve relatively simple problems, which demand not too much computations. However, when problems become very complex and computations very extensive, we need more powerful computational patterns and super-recursive approach provides such patterns.

ACKNOWLEDGMENTS

The research, described in this paper, was carried out within the framework of R&D Project in support of PhD student (session 2019), contract № 192ПД0025-15.

REFERENCES

1. IBM Systems Journal, v. 38, no. 4, pp. 504–507.
2. Turing, A. (1956) Can Machine Think? In The World of Mathematics (Ed. J.R. Newman)
3. Dreyfus, Hubert L. What Computers Can't Do: A Critique of Artificial Reason. New York: Harper and Row, 1972.
4. Gödel, K., 1995, Collected Works III. Unpublished Essays and Lectures, S. Feferman et al. (eds.), Oxford: Oxford University Press
5. SIAM Journal of Computing, v. 4, pp. 431–442.
6. Hopcroft J., J.D. Ullman: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Cambridge, 1979.
7. Gödel, K., 1995, Collected Works III. Unpublished Essays and Lectures, S. Feferman et al. (eds.), Oxford: Oxford University Press
8. Dr. Turing's Automatic Machine - Alan Richmond. “On computable numbers, with an application to the Entscheidungs problem”
9. Turing’s Legacy Minds & Machines Alan Turing presentation, MIT.