Transfinite Turing machines

Sebastián Bravo López

1 Introduction

With the rise of computers with high computational power the idea of developing more powerful models of computation has appeared. Suppose that we have a task whose solution involves an infinite number of steps. With the computational power of *transfinite Turing machines* [6] it is possible to compute the task.

The study of more powerful models of computation have implications in mathematics, physics, computer science and even philosophy [1]. The theoretical models of transfinite Turing machines can be applied to the real world to study physics. Quantum mechanics give us the opportunity to apply the new models to the real world.

Transfinite Turing machines [6] extend the Turing machine model. This approach extends the operation of ordinary Turing machines into transfinite ordinal time. This new model can complete algorithms that require an infinite amount of steps to be completed. The steps work following ordinal numbers. If the machine does not halt in finite amount of time it arrives to the next infinite stage and so on [5]. Transfinite ordinal numbers represent a sequence of increasing ordinals in which is possible to find an ordinal number greater than the rest. The construction of a new model of infinite computability is the key to understand and find algorithms that require an infinite number of steps to compute the solution.

The tasks solved by transfinite Turing machines are called *supertask* [6]. Supertasks are solved in a finite interval of time but that require a infinite number of steps to be finished. This paradox can help the reader to understand the supertask concept. Zeno of Elea (ca. 450 B.C.) had a theory that it is impossible to go from one place to another one. Before arriving, one must first get halfway there, and before that one must get halfway to the

halfway point, till infinite. Based on this argument all motion is impossible, but this kind of supertask could be solved with the use of the new model.

The main type of transfinite Turing machine is the *infinite time Turing machine* [5]. The main property of the infinite time Turing machine relies on the management of an infinite number of steps. Infinite time Turing machines are able to carry out and complete algorithms involving infinitely many computational steps. The main principle to do it is based on the division of the task into infinite stages based on ordinal numbers.

Accelerated Turing machines [5] can increase the computation speed in every step. Division of the infinite steps into several finite stages is one of the main properties to understand the method. Other examples are the *infinite state Turing machines* [6] or the *quantum Turing machines* [2][3][1].

The organization of this paper is the following: in Section 2 the transfinite ordinal numbers will be explained. This topic is very important to understand how transfinite ordinal time is represented. The main types of transfinite Turing machines, infinite time Turing machines, infinite state Turing machines, accelerated Turing machines and other approaches are explained in Section 3. Finally at section 4 there are some thoughts about the power of transfinite Turing machines and the problems that can be solved with the use of the new models. Final conclusions are drawn in the Section 5.

2 Transfinite ordinal numbers

Transfinite Turing machines extend the classical Turing machine into transfinite ordinal time. This implies that the machines need infinite number of computational steps. Unsolvable problems would need infinite computation. Some problems can be solved in countable infinite time and by using *transfinite numbers* [5] we can measure the complexity of infinite computations. The computational steps rely on ordinal numbers. An ordinal number is defined as the order type of a well ordered set.

Totally ordered set is defined by the following conditions:

1. Antisymmetry: $(a \leq b)$ and $(b \leq a)$ implies $a = b$.
2. Transitivity: $(a \leq b)$ and $(b \leq c)$ implies $(a \leq c)$.
3. Comparability (trichotomy law): For any a,b in S, either $(a \leq b)$ or $(b \leq a)$.

Two totally ordered sets $(A, \leq)$ and $(B, \leq)$ are order isomorphic if there is a bijection $f : A \to B$ such that all $\alpha_1, \alpha_2 \in A$, $\alpha_1 \leq \alpha_2$ iff $f(\alpha_1) \leq f(\alpha_2)$.

Finite ordinal numbers are denoted by arabic numerals and transfinite ordinals are denoted using lower case Greek letters. Every finite totally ordered set is well ordered. Any two totally ordered sets with k elements, where k is a non negative integer, are order isomorphic and they have the same order type. The ordinals for an infinite set are denoted by $1, 2, 3, ..., \omega$.

The number series is divided into finite stages $1, 2, 3, ..., \omega$. The number denoted by ω is the order type of a set of nonnegative integers. ω represents the first of the Cantor's transfinite numbers. ω is the smallest ordinal number but higher than the natural numbers. When we arrive to the stage ω, the series continues with the stages $(\omega + 1), (\omega + 2), ..., (\omega + \omega), ...$ New ωs are added an infinite number of times. By ω's it is possible to define all countable numbers [5], some examples of large ordinal numbers are: (ω^2) or (ω^ω).

One of the main properties of the transfinite numbers is the ability to interchange operations: $x + 1 = 1 + x$, $n \cdot x = x \cdot n$. We can define the successors adding numbers to the stage: $(1 + \omega)$ is the successor of ω. For transfinite numbers the order of operation determines the meaning:

1. $(1 + \omega) = (n + \omega) = 2 \cdot \omega = n \cdot \omega = \omega$, but
2. $(\omega+1) = \omega$'s successor, $(\omega+2) = (\omega+1)$'s successor, $(\omega \cdot 2) = (\omega+\omega)$, etc.

3 More powerful Turing machines

Transfinite Turing machines [6] are based on the classical Turing machines. It is possible to increase the power of standard Turing machines by allowing the machine to have infinite time to run, accelerated speed, infinite number of states or quantum states.

3.1 Classic Turing machines

The standard Turing machine consists on several parts [5]. There is a head moving back and forth reading and writing characters on a tape according to the rigid instructions. The characters depend on the alphabet used and generally Turing machines can use any set of characters. It is common the use of the alphabet $0, 1$. All other characters can be represented by strings

of 0's ans 1's. For example we could group 4 bits to create a character. For 4 bits we have 16 different combinations that could represent numbers $(0, ..., 9)$ and some operations $(-, +, *, /)$.

Classic Turing machines have also a control unit which defines the transitions between the different states based on the input. They have also tapes in which the information is stored. The input and output are read and written in those tapes. An algorithm for a Turing machine is defined by the state transitions. The heads can move along the tapes and read and write values according to the current state. When the algorithm finds the solution the current state is the final state and the Turing machine halts. The Transfinite Turing machines use the same principle but with the addition of some properties.

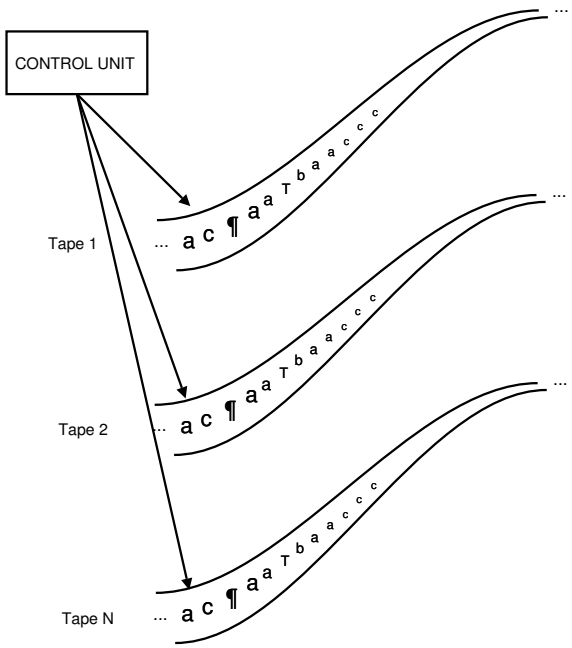

Figure 1: Turing Machine

3.2 Infinite time Turing machines

Infinite time Turing machines [5] can compute an infinite number of steps in a finite amount of time. The tapes on which the infinite time Turing

machines writes are infinite and there is no limit in the space to use. Since infinite time Turing machines can use the entirety of their tapes during their execution, it is possible to use a more wide range of algorithms with them.

The machine can perform several infinite sequences of steps and repeat the process infinitely. The number of steps can be seen as a sequence of ordinal numbers. For example the infinite sequence of infinite sequences of steps is denoted by ω^2. The infinite time Turing machine is an extension of the Turing machine into transfinite ordinal time. At a limit in the ordinal time the machine's actual state is based on the previous states. The machine can compute any recursive function in less than ω steps and evaluate the function in every ω steps [5] obtaining data already calculated. At every step ω its possible to consult the partial results. Even if the task is not finished, there is information already computed that can be obtained as a partial result. This allows to obtain results even if the machine does not halt. This results can be wrong or have some solution and they can be used to know the state of the machine.

3.3 Infinite state Turing machines

Infinite state Turing machines [6] are standard Turing machines with infinite number of states. Although this can be implemented with an infinite transition table, it is possible to do in practice with the use of states based on functions or some other properties. The tape is only used for input and output data and the main work is done by the use of infinite number of states. This leads to better results for questions of computational complexity. As we already have an infinite number of states, we have no need for a tape, we can simply incorporate this into the states.

The number of possible transitions is also infinite but only a finite number of states can be reached from a given one. One obvious way of specifying a state is by an infinite string of 0's and 1's. This gives an uncountable number of states. We simply require that every string at time t determines another string at time $t + 1$. It is possible even to have all the information in the states, and with the change of the states, the input data and the output of any function can be specified and different algorithms can be used. In this example the tape is only used for input and output but it is also possible to define inputs in the states and avoid completely the tape. If we are in the state 1 and we add 1 to our result we move to the state 2, and the state is the result itself.

For example, following the function that adds the number 1 to the current number the next step is current state+1, the previous step is current state -1. The state depends on a given function and the result can be obtained by the current state. If current state is n, the resulting number is n.

3.4 Accelerated Turing Machines

Accelerated Turing Machines [6] can increase the computation speed in every step. Since $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... < 2$, any task can be done in less than two time units. There is no difference between standard Turing machines and the accelerated ones other than the speed of operation which is the same in the first step and exponentially increased in the next steps.

Let us study the following example [6]: Let A to be an accelerated Turing machine that changes the value of a tape position from 0 to 1 when a given Turing machine B halts. If the Turing machine B does not halt, A leaves the original value 0, but after 2 time units the position on A holds the value of the halting function value for this Turing machine B and its input because after 2 time units the machine finish regard of if it halts or not.

This accelerated Turing machine only computes functions $N \rightarrow \{0, 1\}$ but it can be extended to $N \rightarrow N$ by designating the odd positions to begin with 0 and only being changed at most once. Natural numbers and even real numbers can be written in binary on the special squares after two time units.

3.5 Other Turing machines

The addition of the random factor is an interesting property for the Turing machines. The probabilistic Turing machines [7] can have two different and valid transitions from the current state. In each state the machine selects between the valid transitions giving them equal probabilities to be selected. Using this model the Turing machine can compute recursive functions on real numbers and natural numbers. Probabilistic Turing machines can give different results for different executions.

Quantum Turing machines [2][3][1] are becoming very popular. In a quantum Turing machine read, write, and shift operations are all accomplished by quantum interactions. The tape itself exists in a quantum state as does the head. In particular, in the place of the Turing tape position that could hold either 0 or 1, in quantum Turing machine there is a qubit [3][1], which can hold a value between 0 and 1.

Qbits can have different states 0, 1 and 0+1 (state between 0 and 1). Qbits are often represented by an sphere with an arrow inside. Arrow up means 1, arrow down means 0, any other arrow state is called arrow phase. Arrow phase is an intermediate state that provides the quantum Turing machines an interesting property, the capacity to select the transitions from non well defined value between 0 and 1.

The tape of the quantum Turing machine is made of qbits. The machine evolves in many different directions simultaneously. After some time t its state is a superposition of all states that can be reached from the initial condition in that time.

Although the quantum Turing machines can be emulated with standard Turing machines, the quantum Turing machines complexity appears in the layout of transitions of the quantum states. The quantum Turing machine can encode many inputs to a problem simultaneously, and then it can perform calculations on all the inputs at the same time. This is called quantum parallelism. Although the quantum Turing machines are faster than Turing machines, they can not solve more complex problems, they have the same power.

4 Power of transfinite Turing machines

The transfinite Turing machines can solve the same range of problems than the classical Turing machines and much more [5], but we must take into account the concepts of infinite number of steps. The main concept in the new model is the infinite computational power of those machines. The transfinite Turing machines have infinite time to compute an algorithm. Another property is the ability of the infinite time Turing machines to compute partial results at each step. When the transfinite Turing machine arrives to step ω its possible to obtain the state of the machine and results.

Since the conception of the Turing machines it has been known that there exist functions not computable with them. It is clear when we consider an uncountable set of functions from $N \to N$ and compare it with the countable set of the Turing machines.

Halting problem is unsolvable [8] on standard Turing machines. In the classical Turing machines, the halting problem represents the question of whether a given program p halts on a given input n in finitely many steps. Halting problem is one of the first problems which was proved to be unde-

cideable.

If a solution to a new problem is found it can be used to solve an undecidable problem. That is made by transform instances of the undecidable problem to instances of the new problem. If it is not possible to solve the old problem, then is not possible to solve the new one. One consequence of the halting problem is that is not possible to create an algorithm that finds if a statement about natural numbers is true or false.

The proposition that states that a certain algorithm will halt with some input can be converted to another statement about natural numbers. An algorithm that solves any statement about natural numbers can solve also the halting problem, but that determine when the original program halts, that is impossible, so halting problem is undecidable.

Halting problem of the classic Turing machines is solvable with the use transfinite Turing machines after ω (infinite ordinal stage) steps. But transfinite Turing machines have also their own halting problem unsolvable by using transfinite Turing machines.

5 Conclusions

In this paper I have covered the basics of the transfinite Turing machines. I have described different approaches of the transfinite Turing machines. Quantum Turing machines represents a new model capable of very powerful calculation like physical world simulation.

With the use of transfinite Turing machines its possible to extend the classic Turing machines into transfinite ordinal time. Transfinite Turing machines provide a natural new model of infinitely computability and a good point of view for setting the analysis of the power and limits of supertask algorithms (i.e. algorithms that involves infinitely many steps).

In this report I did not presented all the different extended Turing machines. For example asynchronous networks of Turing machines [6] or error prone Turing machines [6] are very interesting approaches, but there exist some similarities with the already mentioned Turing machines variants.

There exist several types of quantum Turing machines like the bulk quantum Turing machines [2][3][1]. Although bulk quantum Turing machines are not more powerful than quantum Turing machines, they provide a better way to define the problems. Both are related with spatial machines [4], an attempt to create more powerful models with the use of 3 dimensional space

as a limit for the develop of new Turing machines models.

References

[1] Paul Benioff. Models of quantum turing machines. *Fortschritte der Physik*, 46:423, 1998.

[2] Gilles Brassard. Quantum computing: the end of classical cryptography? *SIGACT News*, 25(4):15–21, 1994.

[3] Marco Carpentieri. On the simulation of quantum turing machines. *Theory of Computer Science*, 304(1-3):103–128, 2003.

[4] Yosee Feldman and Ehud Shapiro. Spatial machines: a more realistic approach to parallel computation. *Communications of ACM*, 35(10):60–73, 1992.

[5] Joel David Hamkins. Infinite time Turing machines. *Minds and Machines*, 12(4):521–539, 2002.

[6] Toby Ord. Hypercomputation: computing more than the Turing machine. *CoRR*, Department of Computer Science, University of Melbourne, 2002.

[7] Alon Orlitsky, Narayana P. Santhanam, and Junan Zhang. Always good Turing: Asymptotically optimal probability estimation. focs '03: Proceedings of the 44th annual ieee symposium on foundations of computer science, vol 302, no 5644. pages 427–431, 2003.

[8] Alan Turing. Halting problem of one binary horn clause is undecidable. pages Ser. 2, Vol. 42, 1937.