No-State Universal Turing Machine

Gheorghe Ştefan[*]

Abstract

The aim of this paper is to emphasize the possibility to build the *simplest* Universal Turing Machine (UTM). Results a *no-state* machine containing only structures having constant sized simple definitions. Any random structure can be avoided, the resulting structure of machine has only uniform circuits. **Removing the finite automaton from the definition of UTM** is the main structural effect of this approach. The state of the computational process is stored only on the "tape", in contrast with the current versions of UTM in which the state of the machine is given by the state of the finite automaton and by the content of the "tape". In the current UTMs, the automaton *interprets* the description of a certain Turing Machine (TM), but in the proposed, no-state UTM, the description of a certain TM is *executed* by some uniform, simple circuits. Thus, the **segregation** between the simple (physical structure of the machine) and the complex (symbolic description of the computation stored on the "tape") becomes maximal.

1 Introduction

The *0-state UTM* we propose in this paper is important mainly for its **simplicity**. The reduced number of states is only a side effect. But, what does it mean a simple physical or symbolic structure? It is or not important the *number of states* of the control automaton that interprets any TM's description? Claude Shannon reduced this number to two [Shannon '56]. Did he obtain a simple or a complex machine? Our first goal is now to obtain a simple machine and only the second to have a no-state machine. Therefore, we must state precisely the meaning of *complexity*.

Definition 1 *The apparent complexity of a structure is given by the size of its definition.* ◇

Definition 2 *The (actual) complexity of a structure is given by the size of its minimally expressed definition.* ◇

[*]Politehnica University of Bucharest, Faculty of Electronics & Telecommunications. Email: stefan@agni.arh.pub.ro

The previous definitions are suggested by Chaitin's *algorithmic complexity* [Chaitin '77]. (In this approach we will ignore the "user" who must "understand" the definition. Many times the "user" is a machine, having its own complexity to be taken into account. But this is another story.) According with this definition we will define what does it mean a simple or a complex structure.

Definition 3 *Let be a (physical or symbolic) structure, M, its size, $S_M(n)$, and its complexity, $C_M(n)$. We say that M is* **simple** *if*

$$S_M(n) >> C_M(n),$$

and we say that M is **complex** *if*

$$S_M(n) \sim C_M(n).$$

$\diamond$

According to the previous definitions we will prove that the complexity of a TM is given only by the complexity of the combinational circuit connected on the loop of the control automaton, that computes the output and the next state. The rest of the machine, the "head" and the "tape", are both simple. In order to minimize the complexity of the UTM we will start with an actualized representation for the TM and we continue presenting a 0-state UTM.

2 A New "Image" for Turing Machine

The standard definition of TM, using the oldest suggestions of the storage "tape" and of the access "head", hides some essential feature of the concept. In order to emphasize aspects related to the complexity of TM we give an equivalent up to date definition. Instead of "tape" accessed through a "head" will be used a *random access memory* (RAM) addressed with an *up-down counter* (U/DC).

Definition 4 *Turing Machine (TM) is a finite automaton (FA) loop connected with an infinite RAM (see Figure 1) addressed by an infinite U/DC commanded by FA. The automaton performs in each clock cycle the following operations:*

- *receives from the output DOUT of RAM the content of the current accessed cell*
- *according to its own state and to the received symbol:*
  1. *computes a new symbol to be stored in RAM at the same address and applies it on the input DIN of RAM*
  2. *determines the accessing mode of the next cell selecting one of the following actions:*
     - *incrementing the counter (U)*
     - *decrementing the counter (D)*

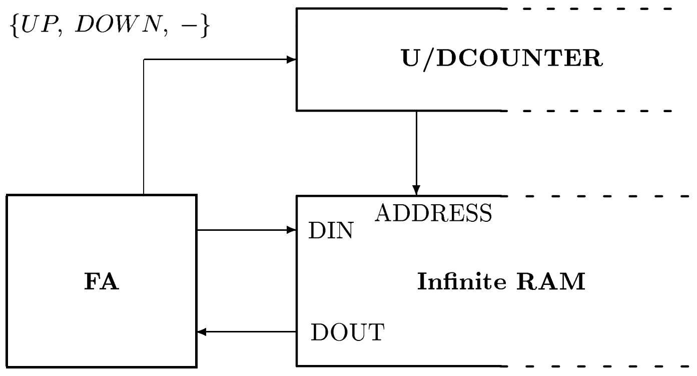

Figure 1: The structure of a Turing Machine

– *maintaining the counter to the same value (-)*

3. *computes the next state of the automaton.*

All changes are triggered with the following active clock transition. More formal:

$$TM = (I,\ Q, f; q_0, \#)$$

where: I is the finite alphabet of the machine, Q is the finite state set, $q_0 \in Q$ is the initial state of the automaton, $\# \in I$ is a symbol stored in the non active cell of memory and f is the transition function of the entire machine:

$$f = I \times Q \to I \times Q \times \{U,\ D,\ -\}.$$

In the initial state the automaton is in q_0, the infinite memory contains a finite string to be processed, ended on both sides by $\# \in I$, the accessed symbol from the string to be processed is the first. $\diamond$

3 The Complexity of the Turing Machine

A detailed structure of TM is presented in Figure 2 in order to emphasize its main components with the associated size and complexity:

1. a **finite automaton** containing:
   - a finite REGISTER: a **simple** structure
   - a combinational logic circuit (CLC): a **complex** structure
2. an **infinite automaton** that is the reversible counter, a simple recursive defined device containing:

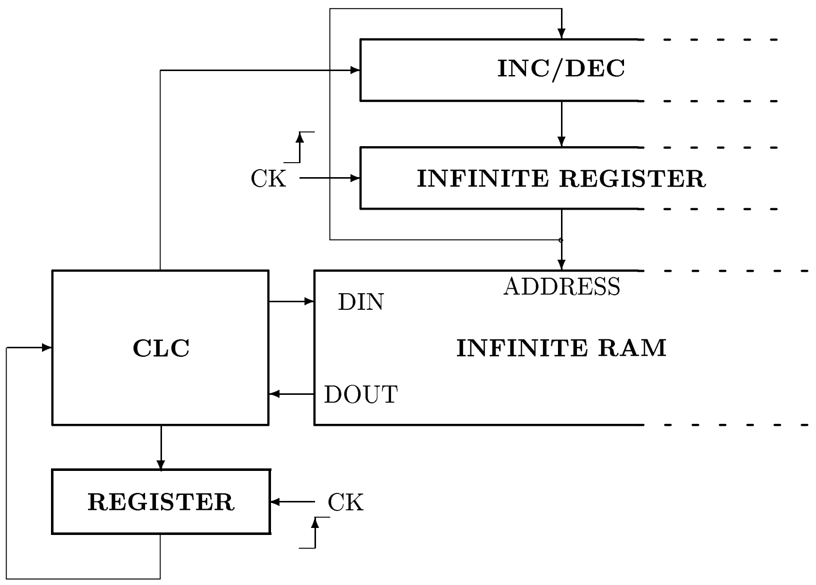

Figure 2: The detailed structure of a Turing Machine

- INFINITE REGISTER: a big sized but **simple** structure
- an infinite incrementer/decrementer (INC/DEC): a big sized but **simple** structure

3. an INFINITE RAM, also a big sized but simple structure.

The single complex, random in Chaitin's sense, structure is CLC that "contains" the algorithm ran by the machine. All the others components of TM have the complexity in $O(1)$. Therefore, we can say that C_{TM} is in the same order with C_{CLC}.

Proposition 1 *Let be* $TM = (I, Q, f; q_0, \#)$. *Then:*

$$C_{TM} \in O((dim(I) + dim(Q)) \times log\ (dim(I) + dim(Q))).$$

◇

Proof: Our TM is characterized by:

$$n = |log_2(dim(I))| + 1$$

$$m = |log_2(dim(Q))| + 1.$$

The complexity of TM, C_{TM}, is given by the sum of the complexity associated of each component added with the complexity of their interconnections. Excepting CLC, all the others components and the interconnections have constant descriptions that not depends by n (the number of bits used to encode the alphabet I) and m (the number of bits used to encode the state set Q). The CLC has $n+m$ inputs and $n+m+2$ outputs (see Figure 2). Therefore, its definition in the worst case is a random (uncompressible) table of binary symbols having 2^{n+m} rows and $n+m+2$ columns. Therefore:

$$C_{TM}(n,m) \sim C_{CLC}(n+m, n+m+2) \in O((n+m)2^{n+m}).$$

Substituting the value of n and m results the proposition. $\diamond$

4 0-State Universal Turing Machine: the Simplest Structure

We will prove that the *structural complexity* of TM can be reduced only implementing it as an Universal Turing Machine (UTM).

Early theoretical studies where devoted to reduce the number of states of the finite automaton, that control UTM, with a minimal increasing of the number of symbols in the alphabet I [Shannon '56]. But we believe that the more important thing is to reduce the structural complexity of UTM. In this respect we will present the simplest UTM built only with simple, recursive defined constant sized circuits.

The problem leading to UTM is to define a machine whose structure can remain unchanged when the executed function changes. In this case we need a machine with:

- an abstract representation for the needed TM, as a string of symbols stored in the memory
- an automaton, useful for all computable functions, that "understands" and "executes" by *interpretation* the abstract representation of any automaton associated to a TM stored on the tape.

Interpretation is a process that acts on a string encoded representation of an abstract machine, to emulate the behavior of that machine. It allows us to deal with *representations* of machines rather than with the machine themselves.

Let be a machine M with the initial content of the tape T: $M(T)$. An interpreter of $M(T)$ will be the machine

$$U(< e(M), T >)$$

where $e(M)$ is the string that describes the machine M. On the tape of the machine U there is the description of M and the string, T, to be processed by the machine M.

Definition 5 *An UTM is a TM, $U(< e(M), T >)$, that has a finite automaton that interprets any TM's description, $e(M)$, stored in the same memory with T, the string to be processed.* $\diamond$

In order to implement an UTM we start from the fact that the transition function f from the state q_i **can be reduced** to a set of the pair of transitions having the next form:

$$f(q_i, x) = (q_j, y, c_l)$$

$$f(q_i, \neq x) = (q_k, z, c_m)$$

where: $q_i, q_j \in Q$, $x, y, z \in I$, and $c_l, c_m \in \{U,\ D,\ -\}$ having the following meaning:

```
if out of RAM=x
  then the next state is $q_j$
       the stored symbol is y
       the access head command is $c_l$
  else the next state is $q_k$
       the stored symbol is z
       the access head command is $c_m$
```

Each such a pair will be associated with a state of the automaton. Therefore, any state can be represented as a string of nine symbols having the form:

$$\& q_i x q_j y c_l q_k z c_m$$

where & is a symbol indicating the beginning of the string associated with the state q_i.

A TM can be completely described by specifying the function f, associated to the *random structure* of the machine, using the above defined strings to compose a "program" P.

Example 1 *Let be a TM that computes the parity of an 1-ary represented number. If the final state is q_3, then the number is even, else it is odd. The initial value of T (the content of "tape") is:*

$$\ldots \#0111\ldots 1\#\ldots,$$

the "head" points the first 0 and the unstructured description of the function f is:

$f(q_0, -) = (q_1, 0, U)$ */The head points the beginning of 1's/*

$f(q_1, 1) = (q_2, 1, U)$ */Passes over the 1's switching between q_1 and q_2/*
$f(q_1, \neq 1) = (q_3, 0, -)$ */The end of 1's is found and the number is even/*

$f(q_2, 1) = (q_1, 1, U)$ */Passes over the 1's switching between q_2 and q_1/*
$f(q_2, \neq 1) = (q_4, 0, -)$ */The end of 1's is found and the number is odd/*

$f(q_3, -) = (q_3, 0, -)$ */Final state for an even number/*

$f(q_4, -) = (q_4, 0, -)$ */Final state for an odd number/*

The correspondent representation as a string $e(M)$ is:
$P = \&q_0 - q_1 0 U q_1 0 U \&q_1 1 q_2 1 U q_3 0, -\&q_2 1 q_1 1 U q_4 0 - \&q_3 - q_3 0 - q_3 0 - \&q_4 - q_4 0 - q_4 0 - .$

$\diamond$

The tape of UTM will be divided in two sections, one for the string T to be processed by the machine M, and one containing the description P of the machine M. The content of the tape will be

$$\ldots \# P @ T \# \ldots$$

where:

- @ is a special symbol which delimits the "program" from the "data"
- the string $P \in (I \cup Q \cup \{D,\ U,\ -\} \cup \{\&\})^*$ is the "program" that describes the algorithm
- the string $T \in I^*$ represents the "data".

The automaton of UTM "knows" how to interpret the string P in order to process the string T. It is the only physical random structure in UTM. The question is: what are the possibilities to minimize this random structure in UTM? The answer is: performing a strong *functional segregation.*

For simplicity, we will use a TM having two "tapes", one for P and one for T. This machine has an actual implementation using a RAM with two ports for *read* and a port for *write.*

The previous form of P must be *translated* in P' that uses for each state, instead of the string $\&q_i x q_j y c_l q_k z c_m$ stored in 9 successive memory cells, the next form, as a single entity stored in one cell:

$$x \triangle q_j y c_l \triangle q_k z c_m$$

where: $\triangle q_j$ and $\triangle q_k$ represents the *finite* distance in memory between the current location and the locations that store the descriptions for the states q_j and q_k. Each program P has a P' form.

Example 2 *Looking back to the previous example, the string P, with the next form:*

$\&q_0 - q_1 0 U q_1 0 U \&q_1 1 q_2 1 U q_3 0, -\&q_2 1 q_1 1 U q_4 0 - \&q_3 - q_3 0 - q_3 0 - \&q_4 - q_4 0 - q_4 0 -$

stored in 45 successive cells can be translated in a string of P' type, stored in five larger successive memory cells:

$$-, +1, 0, U, +1, 0, U$$

$$1, +1, 1, U, +2, 0, -$$
$$1, -1, 1, U, +2, 0, -$$
$$-, 0, 0, -, 0, 0, -$$
$$-, 0, 0, -, 0, 0, -.$$

$\diamond$

The structure of UTM in the most segregated form is presented in Figure 3, where the counters are detailed and some simple combinatorial circuits are added. The program P' is stored in RAM starting with the address n where the description of the state q_0 is loaded. In the following cells are stored the descriptions for $q_1, \ldots, q_4$. The string to be processed, in our case the 1-ary represented number, is stored starting with the address m, greater than the address in which the symbol @ is located. The initial value of the first address "counter" (ADD & $R1$) is n, and for the second counter (Inc/Dec & $R2$) the initial value is m. The multiplexer MUX selects (see Figure 3), according to the output of $Comp$, the appropriate values for:

- the value (y or z) to be written in RAM to the current address generated by Inc/Dec & $R2$ (the value to be written on the tape in the current cycle of the simulated TM)
- the signed number to be added to the current value of "program counter" implemented by ADD & $R1$ (the relative address of the cell that stores the description of the next state: the next "instruction")
- the command applied to the counter (Inc/Dec & $R2$) that points in the data part of the tape

(The latch connected to DOUT2 has only an electrical role, avoiding the transparency on the loop closed through the RAM built by latches. If the RAM would have been built with master-slave flip-flops (a possible, but a very inefficient practical solution) the latch on DOUT2 output is not necessary.)

This strong functional segregation in UTM, between the simple and complex, implies a machine with *no random circuits.* The randomness of the machine is totally shifted into the content of the memory, where a "random" string describes a certain algorithm. Instead of random circuits we have random string of symbols. The *hard* random structure of the circuits is completely converted into the *soft* random structure of the string describing the function executed by the machine.

In this last UTM version the *interpretation* of $e(M)$ is substituted with the *execution* of $e(M)$. The interpretation is a controlled process that involves a finite automaton. The execution is made by simple circuits (in this case, combinational). *Comp*, MUX, ADD, *Inc/Dec* are simple circuits that execute. The size of their definition do not depend on the dimension of I and Q sets. **Removing the finite automaton** from the structure of UTM the machine substitutes the *interpretation* of P with the *execution* of P'.

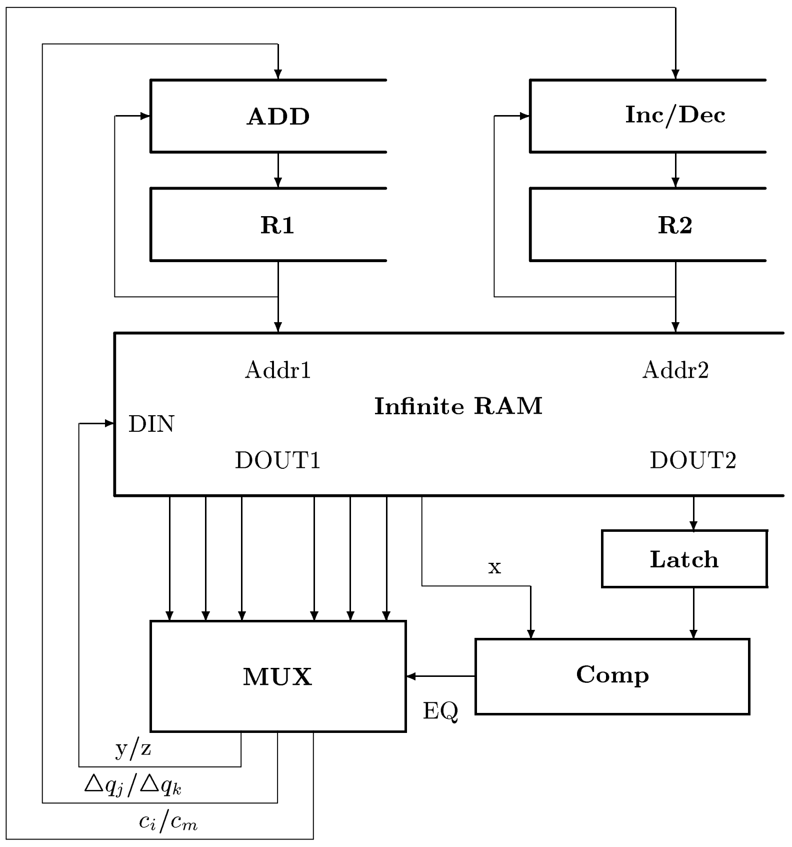

Figure 3: The structure of a recursive defined Universal Turing Machine

In order to use only the simplest structure for implementing the machine associated with any formal language it is evident that the best solution is UTM. The random part of its structure can be null.

Proposition 2 The 0-state UTM is a simple machine. $\diamond$

Proof There is no random part in UTM. The combinational circuit on the loop of a finite automaton is always random and all the structures previously associated to the UTM contain at least a finite automaton. Therefore, only the 0-state UTM is completely built with simple (see Definition 3) circuits. **The finite automaton is avoided** and the interpretation is substituted with the direct execution using simple circuits. $\diamond$

5 Conclusions

The reason of reducing the number of states of an UTM now, in *1Gtransistor/chip era*, is completely different from the reason to make the same thing in the 50s (in *no-chip era*).

1. The complexity of a computation performed by a 0-state UTM depends only on the algorithmic complexity of the string $e(M)$. The structural complexity of the TM is *completely converted* in the complexity of the symbolic description of the computation that will be executed (not interpreted) in 0-state UTM. A *hard* complexity is converted into a *soft* complexity even for the problem having solutions with a less powerful machine than TM.

2. The present day technological evolutions offer the possibility to design machines having bigger and bigger sizes but the complexity of this structure can not follow this tremendous growing process. **When the size of machines start to grow quickly, the complexity must grow slowly**. Else, because of their complexity machines become uncontrolable, more, they become unutterable. Imagine us, a random, complex, circuit containing 10^9 gates! We have no any solution to "express" it in order to be "understood" by an automatic design process. The simplicity of the 0-state UTM supports our steps toward using the new technologies for building powerful machines maintaining their complexity at a low level. Thus, we have a theoretical support to declutch the complexity of the computation by the complexity of machine. Is this a good or a bad way? This is another problem that surpass the aim of this paper.

3. The **segregation** between the *simple machine* and the *complex symbolic description* of computation is the main process in order to avoid the apparent complexity of computation. In this way we reach the actual complexity preserving the *competence*. In order to reach the *performance* we must add the *architectural approach*, as a process that offers a good balance between the physical structures and the symbolic structures involved in computation.

References

[Chaitin '77] Gregory Chaitin: "Algorithmic Information Theory", in *IBM J. Res. Develop.*, Iulie, 1977.

[Shannon '56] C. E. Shannon: "A Universal Turing Machine with Two Internal States", in *Annals of Mathematics Studies, No. 34: Automata Studies*, Princeton Univ. Press, pp 157-165, 1956.