A RELIABLE TURING MACHINE – A DIGEST

ILIR CAPUNI

Barleti University, Tirana, Albania

e-mail: ilir@bu.edu

Përmbledhja

Në këtë punim ne analizojmë elementët kryesorë të konstrukcionit të makinës së Turingut, e cila, me një rritje modeste të resurseve (kohë dhe memorie), mund të ekzekutojë algoritme të çfarëdoshme edhe nëse kjo makinë është nënshtruar zhurmës dhe efekteve anësore që shkaktojnë gabime në funksionimin e saj. Një gabim mund të ndodhë pavarsisht nga gabimet e mëparshme me probabilitet të vogël ϵ. Meqenëse ndërtimi është çuditërisht kompleks, skica e paraqitur këtu është e përshtatshme si një hyrje për ata që duan t'i mësojnë të gjitha detajet e këtij konstruksioni, por është edhe një zëvendësim për ata që duan të marrin një pamje me vija të trasha të këtij konstruksioni me detaje të mjaftueshme për të kuptuar parimet thelbësore të tij. Çështja e ekzistencës së kësaj makine ka qenë problem i hapur që nga viti 1987.

Fjalë kyçe: Makina e Turingut, besueshmëria, zhurma, vetësimulimi.

Abstract

Reliability of computation concerns itself with the computation using a computing machine that is subjected to some noise. In this paper we survey the key elements of the construction of a reliable Turing machine which, with some moderate overhead, can perform arbitrarily large computations even when its operation is subjected to noise causing faults to occur indepdently of each other with some small probability ϵ. Since the construction is surprisingly complex, the outline presented here is suitable as an introduction for those who want to go to the depths of the construction and a substitute for those who want to get a low resolution picture of the construction with enough details to understand the core principles of it. The existence of this machine was an open problem since 1987.

Key words: Turing machine, reliability, noise, self-simulation.

Introduction

This is an exposition paper for a complex hierarchical construction of the Turing machine that, with some modest overhead can simulate any other Turing machine G, even when it is subjected to some noise which causes errors of the head that occur independently of each other with small probability. Existence of such a machine was an open problem since 1987. This construction was first published by the author in (Capuni, 2013) and then, with some shorter proofs in (Capuni & Gacs, A reliable Turing machine, 2021).

Historically the first such construction of a one-dimensional array of cellular automata that can perform arbitrarily large computations even though at each step

each automaton makes an error independently of each other with some small probability is given in (Gacs, 1983). Clearly, in the case of the Turing machine faults do occur only where the head and information far from it does not decay spontaneously. However, even if only with small probability, occasionally a group of faults can put the head into the middle of a large segment of the tape rewritten before in an arbitrarily "malicious" way.

These constructions produce an infinite hierarchy of systems in which each layer simulates the next layer, which in turn has the same code as the previous layer.

An intuitive recipe of such constructions is as follows: We first construct a Turing machine M_1 that can withstand isolated bursts of errors of size β that are followed by an error-free time period of at least V beta steps. Clearly, for this construction, we need to add some redundancy in space and computation, and organize the simulation in a way that even after the burst, the state of the simulation can be easily restored. For the general probabilistic noise, where errors occur independently of each other with some probability ϵ, this program needs the core modification which consists on "forcing" that the machine, instead of simulating the given machine G it actually simulates itself (by writing its code on the tape and simulating it). This will create an array of Turing machines M_1, M_2, ... M_k, ... in which M_1 simulates machine M_2 and can withstand bursts of size β_1 separated by at least V_1 error-free steps (aka level 1 noise). Machine M_2 simulates machine M_3 and can withstand larger bursts of size β_2 and are separated by at least V_2 error-free steps (aka level 2 noise), and so on, machine M_k simulates machine M_{k+1} and can withstand large burst of size β_k separated by V_k error-free steps (aka level k noise). Each of these machines, writes its own code on the tape, and the program of M_1 is hardwired and cannot be corrupted by noise.

This is rather a simplistic description of the skeleton of the construction, and unlike in one-dimensional array of cellular automata where each cell is active, the simulation would work "right away". In the case of Turing machine model of computation, there are many problems that need to be addressed.

Recall, a Turing machine consists of an infinite array of constant sized cells that we call the tape and the head which is positioned over such a cell. One computational steps of a Turing machine consists of the following actions:

1. Read the content a of the current cell and the content s of the registers of the head.
2. Using the transition function (aka the program), infer the next state s', the new content a' of the current cell, and the direction $d \in \{-1, +1\}$ where the head should move.
3. Write a' on the current cell, update the state to s', and move the head in

the direction d.

The head of the machine M_1 is a part of the hardware. However, the heads of all other machines M_2, M_3, ... are actually "virtual" and not in hardware. They are defined in the information stored on the tape of the machine it is simulated by and henceforth, a large burst of errors can wipe out the information that defines the head of the simulated machine. Clearly, loosing a head can never occur at the first level (because the head is the part of the hardware). But this actually calls for modifications of the program of the machine M_1 to deal with this possibility and even for generalizing the definition of the Turing machine to allow for the head to vanish and appear. We will spell out other difficulties later in the sequel.

We will outline all the details of the construction and will also show the standard way of "switching" from bursts to probabilistic noise, and give a general structure of the proofs.

Intuitive explanations of the terminology

A standard definition of the Turing machine i

$$M = \langle \Gamma, \Sigma, \delta, q_0, F \rangle$$

where

1. Γ is the finite, non-empty set of states;
2. Σ is a finite, non-empty set of tape alphabet symbols;
3. q_0 is the initial state;
4. F is the set of final states;
5. $\delta_M: (Q \backslash F) \times \Sigma \to Q \times \Sigma \times \{L, R\}$ is the transition function.

The set F of final states has the property that whenever M enters in a state in F, it can only continue from there to another state in F without changing the tape.

A *configuration* is a tuple

$$(q, h, x),$$

where $q \in \Gamma$ is the state, $h \in \mathbb{Z}$ is the position of the head, and $x \in \Sigma^{\mathbb{Z}}$ is the array of all the cells, that is, it is the tape. The content of the cell at position p is $x[p]$.

The work of the machine can be described as a sequence of configurations $C_0, C_1, ...,$ where C_t is the configuration of the machine at time t. We are interested in a particular sequence of configurations that have the following properties:

1. $1.q(0) = q_0$

2. Nothing changes on the tape except possible where the head is: $x(t+1)[n] = x(t)[n]$ for all $n \neq h(t)$.
3. On each step, the head does not jump for more than two positions: $h(t+1) - h(t) \in \{-1,0,1\}$.

We will refer to this sequence as a *history of* machine M.

If the transition from one configuration to the next one in a history is not obtained by applying the transition function, we say that *a fault* occurred at that time.

Codes

Let Σ_1 be the alphabet of M_1 and Σ_2 the alphabet of M_2. A *block code* is given by a positive integer Q – called the block size and a pair of two functions

$$\phi_*: \Sigma_2 \to \Sigma_1^Q, \phi^*: \Sigma_1^Q \to \Sigma_2$$

with the property

$$\phi^*(\phi_*(x)) = x.$$

Function ϕ_* is the *encoding function* and we use it to encode one letter of M_2 to a Q letter word of M_1.

Function ϕ^* is the *decoding function* and we use it to decode a Q letter word of M_1 to the corresponding letter of M_2.

As an example, let us consider the *repetition code*. Suppose that $Q \geq 3\beta$ is divisible by 3, $\Sigma_2 = \Sigma_1^{Q/3}$, $\psi_*(x) = xxx$. If $y = y(1) \dots y(Q)$, then $x = \psi^*(y)$ is defined by $x(i) = \textbf{majority}(y(i), y(i+Q/3), y(i+2Q/3))$. For all $\beta \leq Q/3$ Clearly, this code can correct effects of a single burst of length β. If we repeat it 5 times, then we can correct 2 such bursts.

Statement of the theorem

Let us consider any machine G which at time t writes a value $y \neq *$ into the cell at position 0.

For any given Turing machine G, there exists some constants $\alpha_1, \alpha_2 > 0$ such that for any input size n there is a block code of size $O((\log n)^{\alpha_1})$ a fault bound ϵ and a Turing machine M_1 such that the following holds. Suppose that the machine M_1 starts working from the initial configuration (obtained by using the mentioned block encoding) with the head in the position 0. Suppose that M_1 runs and faults occur independently of the previous one with probability ϵ.

Then at any time greater than $t(\log t)^{\alpha_2 \log\log\log t}$, the tape symbol a of machine M_1 at position 0 will have the designated output field equal to y with probability at least $1 - O(\epsilon)$.

Remark 3.1 *As we can see, the machine uses very moderate space: the block code is of size* $O((logn)^{\alpha_1})$.

The slowdown also is very moderate: if G halts in t steps, then M_1 will stop in $t \cdot g(t)$, where

$$g(t) = (\log t)^{\alpha_2 \log\log\log t},$$

which is a very slowly growing function.

Later, when defining the noise, we will see that this tower of machines M_1, M_2, ... will not be high.

Machine M_1

Let us consider two machines: M_2 and M_1. We want to define M_1 that simulates M_2 even when it is subjected to noise that cause burst of no more than β_1 consecutive faults to occur that are separated with at least V_1 error-free steps.

We will first give the bare bones construction of M_1 and then, in the subsections below, we will introduce difficulties and problems and upgrade our initial construction accordingly.

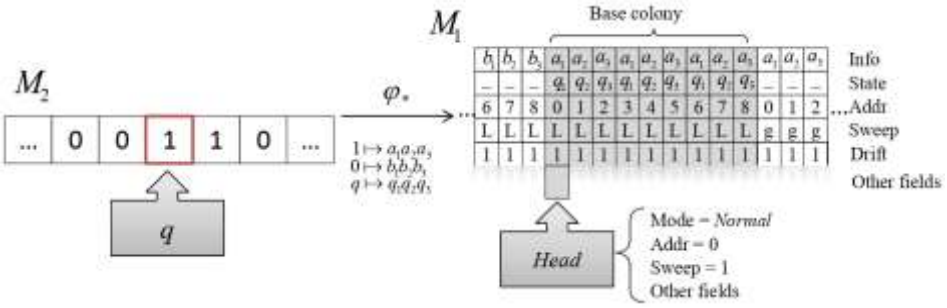

Figure 1: Encoding one cell of M_2 onto a colony of M_1. Each cell of M_1 has many fields

As depicted in Figure 1, we encode the content of a cell of machine M_2 using a repetition code onto Q consecutive cells of M_1. This "group" of consecutive cells that correspond to one cell of M_2 is called a *colony*. Each cell of a colony has a unique address ranging from 0 to $Q-1$. The address is stored in the Addr field. The encoded information from M_2 is stored in the Info field.

The colony corresponding to the active cell is called the *base colony*.

To simulate one step of the machine M_2, machine M_1 first needs to sweep the base colony and compute the majority of the Info to infer the content of the cell of M_2 and its state. Then, it consults the transition function of M_2 and writes the results in a Hold_1 track. It does this three times, storing the results in Hold_2 and

Hold_3 tracks. Then, it updates the Info field and the Drift field (where the direction of M_2 is stored) of each cell by computing the majority of the Hold fields of the cell. It repeats this two times.

To track the progress of the simulations, the M_1 contains the Sweep counter in its head. Similarly, to distinguish exactly the position of the head of M_1 it also needs to store the Addr of the cell where the head of M_1 is positioned. While doing these steps of the simulation, a single fault that changes these two registers will disturb the simulation. In order to restore the state of the simulation, we will write the Sweep and the Addr at each cell of the colony. Now, if a fault occurs, a specific healing procedure can reinstate the state of M_1 and move on with the simulation of machine M_2.

The last stage of the simulation of one step of M_2 is to transfer the head of M_2 in the colony determined by the Drift field: -1 for the left and 1 for the right colony. Recall that the state of M_2 is stored in the Info track of the base colony. That information needs to be "planted" in the state portion of Info track of the neighboring colony determined by Drift. This needs to be done again in a way that guarantees reliability. For this, the head will sweep Info track of the base colony and compute majority of the State and plant this information in the Hold_1 track of the designated neighboring colony. It repeats this another two times storing the result in Hold_2 and Hold_3 tracks. Then, state information at each cell will be computed by computing the majority of the Hold fields of each cell. This will be repeated two times. This completes the *transferring* stage and one work period of steps.

$$U = O(Q)$$

At every single step of M_1 before the machine executes the above program, it first checks if the Sweep value on the current cell is one less than the value of the Sweep register in the head of the machine and that the Addr field on the head and on the cell match. If they match, after the corresponding part of the above program is executed, the M_1 updates the Sweep value on the current cell and moves the head. If they do not match, Alarm is called. This will initiate the healing procedure which we will explain later.

The time diagram of the head movement in time during one work period assuming no faults occur is depicted in Figure 2.

What a single burst can do?

A single fault can change the content of the head (aka the state) of M_1. It can also change the content of the active cell. Therefore, after a burst, the state and the cells that were visited by the head during the burst have arbitrary content.

Intuitively, the entire simulation is devised in a way that it can be restored by doing local repairs of the parts of the colony and retrieving of the normal state of the machine before the burst occurred or before an island is encountered.

To see whether consistency – that is the basic tape pattern supporting simulation– is broken somewhere, a very local precaution will be taken in each step: each step

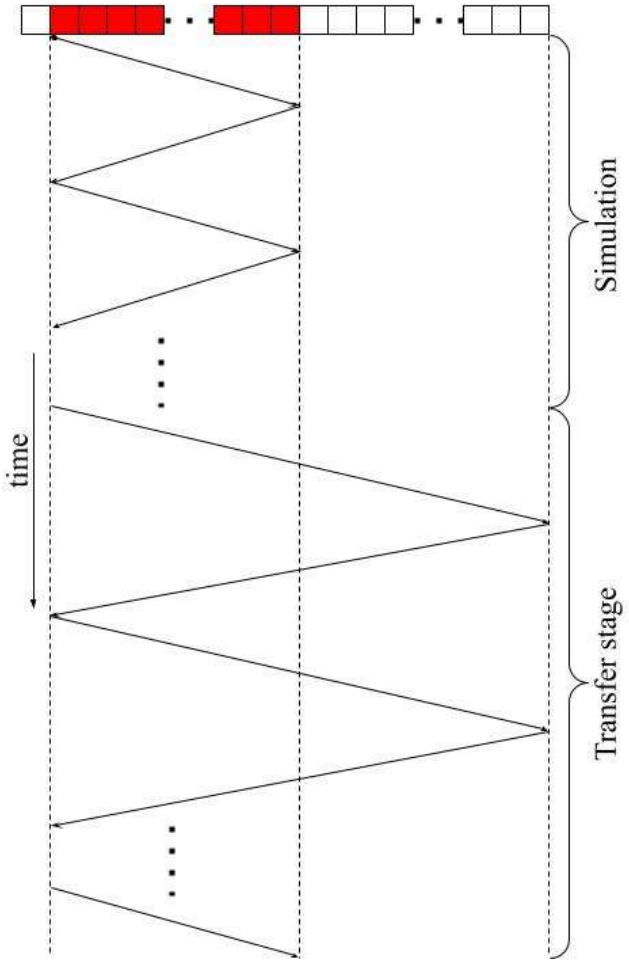

Figure 2: Illustration of the head movement in one work period

will check whether the current cell-pair is allowed in a healthy configuration, which we will elaborate later.

If not then a *healing* procedure will be called; we will also say that *alarm* will be called. On the other hand, the *rebuilding* procedure will be called on some indications that healing fails.

Zigging

Assume that the head is in the middle of the designated right neighbor colony moving to the right during the transfer stage. A single fault occurs and it only increments the Sweep number in the head of M_1. The head now turns left because it thinks that it is performing the next sweep. From the cell where the fault occurred until the right end of the designated colony, we have wrong Sweep number on the tape. Alarm will be called only when the head returns to this area, possibly never.

Our goal is to put enough checks to prevent the machine to cause disproportionate damage from a short burst of faults or a single fault. We want our machine to return to the simulation as soon as possible.

To prevent this extensive damage to occur, we will introduce *zigging*: every Z steps forward are accompanied by Z steps backward and forward, where Z is a constant defined in the paper. It checks that Addr and Sweep registers of the head correspond to those on the tape. If not, Alarm is called.

Healing

The healing procedures Heal , RebuildHeal and the rebuilding procedure Rebuild look as if we assumed no noise or disorder.

Healing performs only local repairs of the structure: for a given (locally) admissible configuration, it will attempt to compute a satisfying (locally) healthy configuration. If it fails—having encountered an inadmissible configuration—then the rebuilding procedure is called, which is designed to repair a larger interval.

Every healing operation starts with a survey zig around its starting point and marks the surveyed area appropriately. If the survey finds some possible healing to do then it performs one step of it, and returns. Otherwise the "attempt" *fails* and in this case, it will build a rebuilding an interval which is larger and is defined with the help of a new field special field.

Using majority, it heals the fields defining the state of the simulation (say Addr, Sweep, Drift, etc.), and not of the simulated machine.

Problems caused by big bursts

In the next two subsections we will consider some challenges that we need to deal with and are caused by big bursts.

Conceptual issues

Consider a big burst which have dislocated two neighboring colonies that correspond to two cells of M_2. The colonies are removed from each other and the space in between is filled with $< Q$ empty cells. Actually this means that two cells have some gaps in between. Clearly, this cannot happen according to our definition of the Turing machine and therefore we need to generalize the definition to allow for this possibility.

Consider a large burst of faults that spans multiple colonies.

By changing the content of the Info tracks, it can create more than one heads of M_2 in each colony, or erase the head completely. This is also something which we will need to address on a conceptual level.

A big burst can also erase completely the colony structure. From the point of view of the machine M_2, this means that a cell of M_2 can be completely removed. Again, the plain definition of the Turing machine does not allow this.

Finally, when two cells of M_2 are misaligned and have a gap in between, the need to remove a cell is accompanied with the need to create one.

We will take into account all these by adopting the notion of a *generalized Turing machine*.

Entrapment

Suppose that a large burst (spanning multiple colonies) occurs creating some intervals I_1, $I_2, \dots, I_n$ consisting of several colonies of machine M_1, and each interval containing a simulated head whereas the neighboring intervals have no relations to each other with respect to simulation. A burst can send the head from one interval to the next one, and from the next one to the previous one forever causing head entrapment.

What heals the effects of big bursts?

We reiterate that a big burst essentially may ruin intervals of cells spanning multiple colonies. Our healing procedure is very local and essentially spans a tiny fraction of a colony. The goal is to repair the simulation structure and carry on the simulation. Why is this so? Also, what heals these big intervals?

The answers to these two questions are related: A big island (of size β_2 – spanning multiple colonies of M_1) is essentially a small island of M_2. But M_2 is a simulated machine and it exists only on the tape as a part of the information

saved in the colonies. For this reason we need to establish the simulation of M_2 by M_1 as soon as possible. If it is not possible, this will be achieved by the Rebuild procedure. Colonies corresponding to (possibly misaligned) cells of M_2 may be created. What part of M_2 is it simulating now over this new cell? Depending on the content of the colony, most probably M_2 is executing some healing procedure.

Self-simulation and universality

A tricky issue is "forced self-simulation". Each machine M_k can be implemented on a universal machine using as inputs the pair (p, k) where p is the common program and k is the level. Eventually, p will just be hard-wired into the definition of M_1, and therefore faults cannot corrupt it. While creating p for machine M_1, we want to make it simulate a machine M_2 that has the same program p. The method to achieve this has been applied already in some of the cellular automata and tiling papers cited, and is related to the proof of Kleene's fixed-point theorem (also called the recursion theorem).

Forced self-simulation can give rise to an infinite sequence of simulations, achieving the needed robustness. Let us point out that fixing the program of self-simulation does not prevent universality. A special track (which in the paper is called Payload) is set aside for simulating the given arbitrary machine G. If this simulation of G does not finish in a certain number of steps, a built-in mechanism will *lift* its tape content to the Payload field of the simulated cell-pair, allowing it to be continued in a colony-pair of the next level (with the corresponding higher reliability).

Noise

The main theorem of the paper talks about faults that occur independently of each other with probability ϵ whereas our treatise and construction are talks about big and small bursts. How can we "switch" from combinatorial noise (bursts) to probabilistic noise?

The set of faults in the noise model of the theorem is a set of points in time. It turns out more convenient to use an equivalent model: an ϵ-bounded *space-time* set of points. Let us make this statement more formal.

We want to deal with *bursts* (rectangles of space-time containing Noise) that are bigger in size and sparser in frequency. To derive such combinatorial constraints from the our probabilistic model we stratify Noise as follows.

We will have two series of parameters: $B_1 < B_2 < \cdots$ and $S_1 < S_2 < \cdots$, where B_k is the size of cells of M_k as represented on the tape of M_1, and S_k is a (somewhat increased) bound on the time needed to simulate one step of M_k.

For some constants $\beta, \gamma > 1$, a *burst* of noise of type (a, b) is a space-time set that is coverable by a rectangle of size $a \times b$. For an integer $k > 0$ it is of *level* k when it is of type $\beta(B_k, S_k)$. It is *isolated* if it is essentially alone in a rectangle of size $\gamma(B_{k+1} \times S_{k+1})$ First we remove such isolated bursts of level 1, then of level 2 from the remaining set, and so on. It is shown in the paper that with not too fast increasing sequences B_k, S_k, with probability 1, this infinite sequence of operations completely erases Noise: thus each fault belongs to a burst of "level" k for some k.

As stated before, machine M_k will concentrate only on correcting isolated bursts of level k and on restoring the framework allowing M_{k+1} to do its job. It can ignore the lower-level bursts and will need to work correctly only in the absence of higher-level bursts.

The rest of treatise on this matter is simple and is well treated in the paper and even some earlier publications on these topics. Arguments and statements given there allow a doubly exponentially increasing sequence U_k, resulting in relatively few simulation levels as a function of the computation time, which is important for establishing the time estimate of the time overhead given in the theorem: if the given machine G halts in t steps, then we can read its result from the designated cell (by construction) after steps.

$$t(\log t)^{\alpha_2 \log\log\log t}$$

A roadmap of the proof

Recall that a big burst may cause cells of (simulated) machine M_2 to have space in between, or cells be dislocated, or a head to disappear or many of them to be created. For this reason and for the sake of proof, we need to define the generalized Turing machine. Then, we need to define the history of such a machine.

Healthy configuration

The definition of a healthy configuration is given by describing the conditions that a configuration must have. These conditions are given in axioms (H1) through (H5). To begin with, as we have explained before, we cannot count that the tape consists of a single interval of contiguous cells. No cell in these intervals should be marked by Rebuild marks and if removed from each other, they can contain a bridge of cells in between. In a healthy interval, the Drift marks on the cell always point to the front, that is a cell from where simulation can be carried out. Also, it is natural to require that all the Drift marks in the colonies in a healthy interval containing the head should point to the base colony. Other conditions establish the so called D-zone, an interval of $Z/2 \pm 1$ cells at the front or ahead of it which contains the head or is adjacent to it and contain zigging marks.

Simulation. Healing

A trajectory is a history in which transitions are done according to the transition function.

To validate the construction it is necessary to prove that the decoding map which is defined and used in the construction takes a trajectory to a trajectory.

Healthy configuration is an ultimate goal. We relax this definition with the notion of "almost healthy" or *admissible configuration*.

Informally, an admissible configuration may differ from a healthy one in a small number of intervals we will call "islands". Even a healthy configuration may contain some intervals called "stains": places in which the Info track differs from a codeword. These pose no obstacle to the simulation, and if they are small and few then will be eliminated by it, via the error-correcting code.

We introduce also the notion of *annotation* which interprets parts of a history, "covering up" small segments that are not quite healthy and leaves other parts uninterpreted.

A central and novel part of the argument is the *annotation game*. There are two players: the annotator and the range extender and they are not adversaries, but rather collaborators. A *range of the annotation* is a subset of $\mathbb{Z} \times [0, t)$. Essentially, the range extender is challenging the annotator to extend its reach. It is actually proven in the paper that the annotator can always respond to the challenge.

References

Capuni, I. (2013): A fault-tolerant Turing machine. Boston: Boston University

Capuni, I., & Gacs, P. (2021): A reliable Turing machine. arXiv preprint arXiv:2112.02152

Gacs, P. (1983): Gacs, P. (1983, December). Reliable computation with cellular automata. Proceedings of the fifteenth annual ACM symposium on Theory of computing (fv. 32-41). ACM