What can we learn from universal Turing machines?

Maurice Margenstern

Université de Lorraine, Metz, France

ABSTRACT
In the present paper, we construct what we call a pedagogical universal Turing machine. We try to understand which comparisons with biological phenomena can be deduced from its encoding and from its working.

ARTICLE HISTORY
Received 26 October 2021
Accepted 29 October 2021

KEYWORDS
Turing machines; universality; genome; viruses

1. Introduction

In one of my papers, sorry, I do not remember in which one, I wrote that if we encode one of the smallest universal Turing machines in a DNA way, we get something whose size is much smaller than the smallest virus. As far as a universal Turing machine has an unpredictable behaviour, the parallel we indicated suggested that the behaviour of a virus is more unpredictable. At that time, I did not argue more than those few words which, clearly, did not raise any noise. As far as at the present moment people are more concerned with what we know about viruses, it can be interesting to go back to that comparison and to discuss whether it is relevant or not.

In Section 2, we remember the results about small universal Turing machines and classical results about Turing machines. Then, we remind the reader results about the smallest universal Turing machines. In Section 3, we introduce the informal notion of a pedagogical universal Turing machine and we build such a machine. In Section 4, we discuss whether the comparison which is indicated in the first paragraph of the present section is relevant or not. In Section 5, we put a temporary end to the dispute.

2. Universal Turing machines and tiny ones

In the present paper, the reader is supposed to know the definition of a Turing machine. To make things as clear as possible, we indicate that here, we only consider deterministic Turing machines with one head and a single infinite tape in both directions. The instructions of a Turing machine will be represented in a table: the columns are labelled with the alphabet of the machine, the lines are labelled with its set of states. We remind the reader that a configuration of a Turing machine is the smallest finite segment of its tape containing the scanned square outside which all squares are empty together with the position and the state of the head. All machines we consider in the paper have a finite initial configuration from which it clearly follows that at any time during its computation, the configuration of a Turing machine is also finite.

As an example, we give the instructions of a Turing machine which performs an addition over two numbers written in unary representation: \( n \) is represented by \( n \) vertical strokes. The initial configuration...
looks like that one:

\[ _* |^n * |^m * _ \]  \hspace{1cm} (1)

In that configuration, \_ represents the symbol which means that the square of the tape containing it is empty. It means that there is no information in the square. A simple machine consists in replacing the \* in the middle by a | and then erase the rightmost \* and replacing the rightmost | by a *.

The machine of Table 1 is what I call a courteous Turing machine. It is defined by two conditions: the machine head never goes to the left-hand side of its initial position and in the final configuration, the result is concatenated to the initial data. For a courteous addition, the final configuration corresponding to (1) is:

\[ _* |^n * |^m * _ \]  \hspace{1cm} (2)

Let us explain the notation used for Table 1 which follows Minsky’s conventions as formulated in [1]. The format of an instruction is \textit{AMs} where A is the letter written by the machine in the scanned square in place of the letter it has seen, M is the move of the head: either to left, symbolised by L, to right, symbolised by R or no move at all symbolised by Z. When a symbol is missing, it means that it is the same as the label of the column if it is a letter, or the same as the number of the line if it is a state or Z if it is a move. Those conventions are illustrated by the table. The symbol ‘!’ means the halting state: it stops the computation. Note that the instruction represented by an empty entry may also be considered as halting the computation: the scanned symbol is unchanged, the state of the head remains the same and the head scans the same square and all this forever. Accordingly, when such an instruction is performed no change happens after that over the configuration which endlessly remains the same. We shall consider that situation as identical to a halting instruction as far is it can easily be detected.

The working of the machine is rather simple: it marks the leftmost and the rightmost * by X, states 1, 2 and 3, and then, it marks the current read | by a which triggers the copying of | on the leftmost empty square after the rightmost X: states 4, 5 and 6. The end of the computation is detected under state 4 when marking a | fails as far as the rightmost X is met. States 7 and 8 allow the machine to write extremal *’s while replacing all X by *. State 9 allows the machine to stop at its initial position.

We can see that many instructions have a single symbol: that of the move. I call such an instruction a glide: it is particularly clear under states 5 and 6: the machine head runs over the configuration while it meets some expected symbol in order to change the direction of its motion. Note that there are 13 glides in the table, 21 entries of the table are empty, meaning halting instructions as already mentioned.

\section*{2.1. Universal Turing machine}

It is now time to consider universal Turing machines. In what sense such a machine is universal? The meaning is the following: a Turing machine is called universal if it is able to simulate any other Turing machine. Let us make it more precise. Let \( M \) be a Turing machine and \( C \) be an initial configuration for
The computation of $M$ starting from $C$ is a sequence $\{C_k\}_{k \geq 0}$ of configurations for $M$ such that $C_0 = C$ and $C_{k+1}$ is obtained from $C_k$ by the application of the single instruction of $M$ which can be applied to $C_k$. The computation is a finite sequence if and only if the last configuration in the sequence contains the halting state. A machine $U$ is said universal if, for any Turing machine $M$ and for any initial configuration $C$ for $M$, there is a configuration $K$ of $U$ such that the computation $\{K_n\}_{n \geq 0}$ of $U$ starting from $K$ contains a subsequence $\{K_n\}_{n \geq 0}$ such that $K_n = \tau(C_k)$ where $\tau$ is a transformation of configurations for $M$ which does not depend neither from $C$ nor from $M$.

Accordingly, a universal Turing machine $U$ must be able to simulate any Turing machine $M$, even if the number of states of $M$ is bigger than that of $U$, even if the size of the alphabet of $M$ is bigger than the size of the alphabet of $U$. Turing proved that it is possible and the key point for that is the construction of $\tau$ and of $K$. A solution to that problem is to define the transformation $\tau$ as a translation of the instructions of $M$ as well as the squares of its tape. Although the tape of a Turing machine is infinite, its configuration at any time is finite which clearly follows from the above definition of a configuration. The number of instructions of $M$ also is finite so that those finitely many finite elements can be, in principle, translated on the tape of $U$.

The idea for that is to split $K$ into two parts: one contains the $\tau(l)$ for $l$ running over the elements of the table of $M$, the other contains the $\tau(x)$ for $x$ running over the configuration of $M$ at the considered time of its computation. The place of the head and its state must be marked in some way. If that point is satisfied, the working of $U$ is simple. It locates the $\tau(l)$ for the $l$ which applies to $C_k$ for the considered time $k$. It copies $\tau(y)$ where $y$ is the letter in $l$ onto $\tau(\xi)$ where $\xi$ is the square of $C_k$ scanned by the head of $M$. When it is performed, $U$ moves the position of the head of $M$ from where it is in $C_k$ onto its place in $C_{k+1}$ under construction. When the head is in its new place, $U$ changes the state of the head of $M$ to the state indicated by $\tau(l)$. In Section 3, we describe an explicit universal Turing machine performing the just described behaviour. We will bring in a tuning of a few points which will be explained in that section.

How big is such a universal Turing machine? Roughly speaking, less than 20 letters and less than 100 states are enough to make the table of a universal Turing machine. Now, if we compose a universal Turing machine $U$ with a machine which does nothing, we obtain a new universal Turing machine whose table strictly contains that of $U$. Accordingly, there are infinitely many universal Turing machines. Note that it is easy to make a bigger universal Turing machine compared with another given one. Is it possible to make a smaller universal Turing machine?

The answer is yes, if we start from a rather big universal Turing machine.

Small universal Turing machines happened to be a source of important works. It is not the place in this paper to report the history of that race. We refer to [2] for such a sketchy account and to [3] where a more recent state of the art is described. The key reduction for the table of a universal Turing machine is to use the delayed computation of a Turing machine. Instead of directly simulating a Turing machine, we allow the machine supposed to be universal to simulate another way of computation which, in its turn, is able to simulate any Turing machine. Such a system is used in Rogozhin’s paper [4] which gave a decisive impulse to the race to universal Turing machines as small as possible.

2.2. Tiny universal Turing machines

The results of [4] were a long time the best results. Initially appearing in a Soviet journal, Rogozhin [5] was an improved presentation of the results in Theoretical Computer Science, far much accessible. More than 20 years later, Neary and Woods obtained smaller machines, see [6]. As far as the present paper does not aim at giving an account of that race, we will present a small universal Turing machine contained in [6], the machine with four symbols and six states. That machine improves the machine with four symbols and seven states of [5]. That machine, as well as all machines of [4] and [6], is universal in the following meaning: it simulates another system of computation which is able to simulate any Turing machine. The program of the machine is reproduced by Table 2 under a translation of the alphabet to which we turn back in Section 4.
The table displays those conventions. As examples, we have the instruction when reading $A$ in state 1: $U \ L$, and also the instruction when the head reads $U$ in state 5: $R$. We call that latter move a *glide* as far as the head passes over the symbol without changing it and remaining in the same state. From the table, we can see that the machine has 23 instructions. The missing instruction for the entry for the letter $A$ and state 6 is in fact the instruction $A Z 1$. Accordingly, that instruction makes the head stay over the same square without changing its content. We already mentioned that we consider that it is a halting instruction. The writing of the table requires at least 54 letters: those used in the writing of the instructions of Table 2. Without Minsky’s conventions, we need 72 letters. In that counting, each letter of the alphabet of the machine, each symbol of move and each number for a state counts for one unit. From our discussion about a universal Turing machine, we can see that the number of letters depends on the encoding we use. As an example, if we used a binary encoding, the four letters of the alphabet require two bits with the convention that $A$, $C$, $G$, $U$ are encoded by 00, 01, 10, 11 respectively. We can encode $L$, $R$ and $Z$ by 01, 10, 00 respectively. We can encode the number of a state by its binary representation. If we wish to encode the table itself, we need a delimiter for the instructions and a delimiter for the states. We deal with that question in Section 3 to which we now turn.

### 3. Pedagogical universal Turing machines

As indicated in the name *pedagogical universal Turing machine*, the machines we consider under that terminology defines something which should be easily understood. That very condition makes it impossible to provide a formal definition for that notion: what is indeed easily understandable? What another person understands can be not understood by me and, sometimes, conversely. So that such a notion is clearly subjective. However, I think that if somebody actually knows and understands what a universal Turing machine is, that person can understand the working of the pedagogical machine we give in the present section.

#### 3.1. Working of the pedagogical universal machine

As mentioned in Section 2, we consider a deterministic Turing machine with a single tape and a single head. In Section 2.1, we stressed the feature that a universal Turing machine $U$ must be able to simulate any Turing machine $M$, whatever the size of the alphabet $A$ of $M$ and whatever the number $N$ of states of $M$. The solution to those constraints is to encode the letters of $A$ as well as the numbers in $\{1 \ldots N\}$. Usually, the encoding is conceived in order to facilitate the location of instructions and of the new state. The most convenient solution is to represent the letters by their rank in a fixed ordering of $A$ and to do the same for the states. Then, the location is easy: it is enough to mark the instructions by a delimiter, to gather in the same area the instructions depending on the same current state of $M$ and to delimit also such an area. Then the location is obtained by a one-to-one correspondence between the number of symbols in the unary representation of the number of a letter or of a state and the number of delimiters to cross in order to access the needed area. Basically, the working is the same as what the courteous addition performed in Section 2.
When the needed instruction is obtained, the replacement of the content of the scanned square by the letter given by the instruction is performed by a copying process. The unary representation makes that operation somewhat complex: if the length of the new letter is equal to that of the scanned one, there is a simple copying process to perform. Otherwise, the situation is more complex if the length of the new letter is different from that of the scanned one. If it is shorter, the square has to be shrunk which triggers to move the rest of the tape to left. If it is larger, the square has to be widened which entails to move the rest of the tape to right.

The move of the whole tape of the simulated machine is also needed if the head goes to the left-hand side of the leftmost square of the simulated configuration. Fortunately, we may assume that our universal Turing machine has to only simulate Turing machines which work on a half-tape only, i.e. the tape is infinite to the right only. We say that such a machine which observes one condition of courtesy is polite. Indeed such machines can simulate any Turing machine \( M \) on a bi-infinite tape: it is enough to imagine that the half tape is divided into two parts: one devoted to the right-hand half of the tape of \( M \) and the other part is devoted to its left-hand side part. That entails a larger alphabet and a bigger number of states only. It is easily feasible. Another constraint, in order to make the code lighter is to forbid the stationary state: the head has always to move, either to left or to right, otherwise the instruction is considered as a halting one. That constraint does not alter the generality of the result as far as a machine with stationary instructions can be simulated by a machine which rules out such instructions. The price to pay is to allow the machine to use more states.

The price to pay with unary representation of the numbers is a longer representation of each element in the code of the universal machine. In order to get a shorter code for our pedagogical universal machine, we decided to represent the encoded numbers in binary. The counterpart is a complexification of the location process. In a first step, \( U \), our universal machine, transforms the binary representation into a unary one. In a second step, the temporary unary representation is used to locate the expected element.

In order to evaluate the maximal size needed for such a transformation, \( U \) counts the delimiters for the state areas and also those for the instructions in such an area, keeping the largest one written in unary. In Section 3.2, we more precisely describe the process together with its implementation in the code of \( U \). Roughly speaking, we implement the function \( n \mapsto 2^n \) together with the reverse function.

The tape of \( U \) is divided into two parts: to the left-hand side \( P \), the set of instructions collected state by state, to the right-hand side, \( T \), an encoding of the squares of the current configuration of \( M \).

The working of \( U \) can be divided into cycles where each such cycle simulates the execution by \( M \) of one step of its computation on its tape. A cycle is divided into five steps. At the beginning of the cycle, \( U \) knows the current state of \( M \) and it knows which square of the tape of \( M \) is currently scanned. The area and the scanned square are both marked by the same symbol \( \text{w} \) which replaces the corresponding delimiter. The first step for \( U \) consists in transforming the binary representation of the scanned letter in the scanned square into its unary representation stored in an appropriate area \( B \) to the left-hand side of \( P \). The second step consists in locating the execution of \( M \) to be performed by \( U \): \( U \) counts the number of appropriate delimiters in \( P \) thanks to the unary representation stored in \( B \). In the third step, \( U \) copies the letter indicated by the instruction \( I \) onto the letter of the scanned square, in \( T \). As we use the binary representation of numbers and we know the size of \( A \), we decide that the size of binary representations is standardised into a fixed size possibly using an additional padding symbol: if \( A \) contains \( n \) letters, the maximal size is \(|n|\), the size of its binary representation, so that if \( k < n \), it is represented by \( k_2 \text{w}^{n-|k|+1} \), where \( \text{w} \) is the padding symbol, and \( k_2 \) is the binary representation of \( k \) and \(|k|\) is the size of \( k_2 \). The third step consists in moving the head of \( M \) which is dictated by the appropriate symbol in \( I \). The step consists in moving \( \text{w} \) on the previous or on the next delimiter. The fourth step consists in transforming the binary representation of the new state in \( I \) into its unary representation. The fifth step locates the delimiter of the area devoted to the new state thanks to that unary representation: accordingly, we are in the situation of the starting point of the next cycle.

We implement those indications in the next section, completing the just given description by details implied by the implementation.
3.2. Constructing a pedagogical universal machine

As mentioned in Section 2.1, the tape of $U$ contains an encoding of the table of $M$ together with an encoding of the tape of $M$. That latter representation is possible as far as $M$ starts from a finite configuration and as far as at each step of its computation, only finitely many squares of its tape are non-blank as already mentioned.

Figure 1 illustrates the basic structure observed by the configuration of the tape of the pedagogical universal Turing machine $U$. The part of the tape to the left-hand side of the leftmost $S$ is devoted to auxiliary computations we explain a bit later. In between both $S$ the tape contains an encoding of the table of $M$. The part of the tape to the right-hand side of the rightmost $S$ contains an encoding of the configuration of the tape of $M$ restricted to a segment outside which all squares of the tape of $M$ are empty such that the segment also contains the head of $M$.

The code of the program of $M$ is a concatenation of the codes of its instructions, provided that for each state, an instruction is present for each letter. The code of an instruction obeys the following format:

$$Ya_1 \ldots a_k h^l M a_0 \ldots a_m$$

where $a_i \in \{0, 1\}$, and $a_k = a_m = 1$. Indeed, $c_l = \sum_{i=1}^k a_j$ and $c_s = \sum_{i=0}^m a_i$ are numbers: $c_l \in \{1 \ldots L\}$ and $c_s \in \{1 \ldots N\}$ where $L$ is the number of letters in the alphabet of $M$ and $N$ is the number of its states. Accordingly, letters and states are designated by their rank in an ordered representation of the alphabet and in the list of states. Moreover, in order to facilitate the working of $U$, in (3), we assume that $k+l$ is a constant value with $l \geq 1$. The codes in (3) are binary representations of the codes written with the low powers to the left.

The area which lies to the left hand-side of the leftmost $S$ is devoted to the computation of the values of $n \mapsto 2^n$ for $2^n \leq p$, where $p$ is the maximum between $L$ and $N$.

Let us clarify that point. The initial configuration of the tape of $U$ is the following one:

$$\_S\_ \_S\_$$

The program and the tape of $M$ are delimited to left by $S$. Each square of $M$ is delimited to left by $U$. The instructions of $M$ are delimited to left by $Y$ and the area containing the instructions attached to a given state are delimited to left by $X$. The tape of $M$ is delimited to right by $\_$. The blank of $U$ is, by convention, the first letter of $A$. The area devoted to the computation of $n \mapsto 2^n$ is delimited to right by the leftmost $S$ and, to left by $\_$. That area contains two sub areas marked by $F$. In between $S$ and $F$, we have in unary a number $p = \max(L, N)$. To the left-hand side of $F$, we have the representation in unary of the powers of 2 which are not greater than $P$. We can see that area of powers of 2 as a pattern to construct a number in unary knowing its binary representation.

As an example, the following configurations show us how to compute 111111 from 011, as illustrated in (5):

In (a) we have the initial setting when it is installed by the first operation performed by $U$ before the simulation itself. In (b) we have the copy of 011 on the pattern: note that the writing of the digits is on the reverse order with respect to the source. In (c) $L$ marks the power of 2 to be copied over the 0’s

$$\_F\_ \_S\_ \_S\_$$

Figure 1. Basic structure of the configuration of the tape of the pedagogical universal Turing machine.
between \( F \) and \( S \). In (d) the copy is performed: four \( h \)'s replaced four 0's. Moreover, the next power of 2 to be copied is marked with \( L \). In (e), we can see that the copy of the last power of 2 to be copied is performed. The pattern is completely restored and we have six \( h \)'s which is the writing in unary of 011. Table 3 is extracted from the table of the pedagogical universal Turing machine. It contains the instructions which perform the transformations sketchily indicated in (5).

The reader is invited to look at the configurations of (5) in order to better see the transformations given in what follows. Under state 40, a \( d \) is transformed into \( h \) and the head crosses \( F \) which makes it to change the state to 41. Under that state, meeting the leftmost 0 in between \( F \) and \( S \), the head transforms 0 into \( h \) and change its state to 42. State 42 is a glide to left until \( L \) is met which triggers the copy of the corresponding power of 2. The meeting of \( L \) makes the head change its motion to right and to change its state to 43. That state is a glide over the symbols which are marked as copied. The glide occurs when a not yet copied symbol is met: 0, \( h \) or \( e \) which are changed to \( Y \), \( U \) or \( R \) respectively. When a symbol is met, the head goes again to right under state 40. The head glides over \( U \), \( h \) and \( e \), it changes \( d \) to \( h \) and changes its state to 41 going on to right, running over \( h \)'s until 0 is met. That writing triggers a new cycle of copying that power of 2. That copying is completed when \( F \) is met under state 43 which, looking after a symbol to copy, meets \( F \). Then the head turns to state 44 which unmarks the copied symbol and looks after \( L \) which is changed back to \( d \) and makes the head to turn to state 45 and to go to right. Under that state, the head looks after the next \( e \) which is changed to \( L \). While going to the right, the head restores the 0's to the left-hand side of the new \( L \) which were changed to \( h \). When \( e \) is met, the head is now under state 40 and a new cycle of copying the current power of 2 is triggered. Under state 45 too, if looking after \( e \) the head meets \( F \), it goes to right and, meeting \( h \), it knows that the transformation from binary to unary is performed. The head turns to state 46 which starts the process of detecting the needed instruction. The process is already started by state 45 which transforms the met \( h \) to 0.

That process is used twice in a cycle of computation of \( U \) devoted to the execution of one step of \( M \). In fact, the same states are used for the conversion of the binary representation copied on the pattern to the left-hand side of \( F \). We shall explain that point a bit later.

We give the full table of \( U \) in Appendix 1 to the paper, see Table A1. Before going on the explanation of the precise working of \( U \), it is time to indicate how the tape of \( M \) is encoded. The implementation of a square of \( M \) is given by (6):

\[
Ua_1 \ldots a_k h^\ell
\]

where \( a_i \) is in \( \{0, 1\} \), \( \ell \geq 1 \) and \( h + k \) is the same constant value as in (3) for the instructions. That makes easier the copying of the letter of the instruction onto the scanned square: it overwrites the content of the square without any comparison.

The first transformation from binary to unary is performed for the letter \( n \) written in the scanned square. We know that \( n \) is the binary representation of the rank of the considered letter in the alphabet of \( M \). From (6), we know that \( n = \sum_{k=1}^{\ell} a_i 2^{i-1} \), so that the binary representation of \( n \) on the square has the low powers of 2 to left. We note that (5) shows us that the binary representation over the pattern has the high powers to left. The display is chosen in order to take benefit of the direction of the motion of the head in order to facilitate the operations.

**Table 3.** Part of the table of the pedagogical universal Turing machine which computes \( n \) in unary from its binary representation in the pattern at the left-hand side of \( F \).

|   | 0 | 1 | L | R | Y | U | F | h | d | e |
|---|---|---|---|---|---|---|---|---|---|---|
| 40 | R |   |   | R | R41 | R |   | hR | R |
| 41 |   | hL42 | L |   |   | R |   | L42 |
| 42 | L |   | R43 | L | L | L | L | L | L |
| 43 |   | YR40 | R | R | R | L44 | UR40 | RR40 |
| 44 |   | dR45 | eL | 0L | dL | R |   | OR46 | R |
| 45 | R |   |   |   |   |   | R |   | LR40 |
The alphabet of $U$ consists of the following symbols:

$$\_ , 0, 1, L, R, X, Y, U, S, h, d, e, F, Z, T.$$ 

Note that \_ is the blank, i.e. the symbol indicating that the square which contains it is empty. Symbols $L$ and $R$ occur in the instructions as indication of the direction of the move to be performed by the head when it executes the corresponding instruction while $Z$ indicates a halting instruction with no mention of a direction. The symbols $X$, $Y$ and $U$ are delimiters. We already met $Y$ and $U$, delimiters of an instruction and of a square of the tape respectively. The symbol $X$ delimits the states: in between two consecutive $X$, we have, on the tape of $U$, all the instructions corresponding to the left-hand side $X$. It is the same for the last state whose instructions are displayed between the rightmost $X$ and the rightmost $S$. The symbol $h$ is basically a padding symbol in the binary representations used in instructions and squares of the tape. It is also used for marking other symbols during some operations. We already met the symbol $F$. For what are symbols $d$ and $e$, they are auxiliary symbols used during copying processes. Two $W$’s occur on the tape: one usually replaces the $Y$ which delimits the instruction currently executed. That $W$ replaces the leftmost $X$ in the initial configuration. The other $W$ replaces the $U$ which delimits the scanned square.

The first operation performed by $U$ is to compute the number of states and of letters of $M$: it is enough to compute from $S$ the number of $Y$’s from the leftmost $X$ to the next one and then, again starting from $S$, to compute the number of $X$’s until the next $S$ is met. That computation is performed by states $1$ up to $11$, see Sub-table $A1.a$: states from $1$ up to $6$ compute the indicated $Y$’s and states from $7$ up to $11$ compute the $X$’s. State $11$ puts the $F$ which closes the zone of $1$’s giving $P$ in unary. In that zone, the machine computes in unary the powers of $2$ which are not greater than $P$. That is performed by states $12$ up to $22$, see Sub-table $A1.b$. The first four states writes $\ddhd$ to the right-hand side of $F$. Then state $17$ looks after the leftmost $1$ which is transformed into $h$. The head turns to state $18$ and moves to left to mark the symbol corresponding to the transformed $1$: $d$ is turned to $e$ and $h$ in between $F$ and the rightmost $d$ or $e$ is changed to $R$. Under state $18$ the head goes back to left under state $16$ in order to look after a fresh symbol $h$ or $d$ to copy, marking it as $R$ or $e$ respectively. The cycle of the succession of states $17, 18$ and $16$ is repeated until $F$ is met under state $16$. It means that the number of $h$’s written by state $17$ is equal to the number of symbols $e$ and $R$ in between $F$ and the leftmost $h$. The head goes back to right under state $19$ in order to change the rightmost $h$ written by state $17$ to $d$, which is performed by state $20$ and state $16$ is called for a new cycle of copying. When $S$ is reached under state $17$, $U$ knows that the last marked power of $2$ not greater than $P$ is the highest one. Accordingly, that part of the computation is over. State $21$ is called in a motion to left in order to rewrite $0$’s as $1$’s and to return $e$’s and $R$’s to $d$’s and $h$’s respectively. The task is continued by state $22$. When $F$ is reached under that state, a new action of the pedagogical universal Turing machine starts beginning under state $23$.

In Sub-table $A1.c$, states $23$ up to $29$ construct the pattern which lies to the left-hand side of $F$ and which is later used to compute the unary representation of a number given in binary representation. The head goes from one side of $F$ to the other, looking after the next symbol to be copied when it is to the right-hand side of $F$ and copying it when it is to the left-hand side. States $23$ up to $27$ perform that copying. That part of Sub-table $A1.c$ is displayed on the left-hand side half of the sub-table. State $23$ performs the overwriting of the symbols to be copied, namely $0$, $h$ and $d$. State $24$ writes $d$ on the rightmost $\_ $-square, while state $26$ performs the same operation for $0$. As far as those states go from the marked symbol to the copying place, when it is under those states, the head goes to left, gliding over $0$, $1$, $d$ and $F$. State $25$ operates the opposite motion, leading back the head under state $23$ when it meets $F$. The end of the transformation happens when looking after a symbol to be marked the head under state $23$ reads $S$. It turns to state $27$ which transforms the initial $1$’s in between $F$ and $S$ into $0$’s and it turn to state $28$ when it meets $F$. Two states $28$ and $29$ are needed to go to the $W$ which locates the square scanned by $M$. States $28$ and $29$ are displayed on the right-hand side half of Sub-table $A1.c$. 
In Sub-table A1.d, states 30 to 40 copy $n$, the binary representation of the letter scanned by the head of $M$, onto the scale-pattern lying on the left-hand side of $F$. State 30 operates on the square scanned by $M$ which is located by the appropriate $W$. It glides over 0’s and 1’s which have been changed into $\bar{d}$’s and $e$’s respectively until it meet a free symbol 0 or 1 which it changes as already mentioned. Then, state 31 is called to start the copying of a 0 while state 35 is called to perform the same operation for a 1. Two states are needed to cross the configuration of $U$ from the square scanned by $M$ up to $F$. In the area delimited by $F$ and the rightmost $\_\_\_\_\_\_\_$ to the right-hand side of it, state 32 overwrites a $\bar{d}$ as $h$ as far as the corresponding power of two is marked by 0 in the binary representation of $n$. Then, by states 33 and 34, the head goes back to the scanned square and then state 30 is called for marking a new symbol. Similarly, state 35 looks after $F$ and then state 36 overwrites the fresh $\bar{d}$ as $e$ as far as the corresponding power of two is marked by 1 in the binary representation of $n$. That marking is performed by state 36 and the return to the scanned square is again initiated by state 33. That copying process is completed when under state 30 the head meets $h$. As mentioned in (6), there is at least one $h$ in the code of a square of the tape of $M$. Then state 37 is called and the head goes to left until it meets under state 38 the rightmost $\_\_\_\_\_\_\_\_$ Then the head goes to left under state 39 until it finds the leftmost $e$. Then state 40 is called to change that $e$ into $L$ in order to start the conversion of the binary representation to a unary one.

In Sub-table A1.e, its left-hand side half reproduces Table 3 in another display. That sub-table displays the states from 40 up to 50. The columns, the lines are labelled by the states, the letters respectively. In the sub-table, there is no line for $\_\_\_\_\_\_\_$ as far as it raises only a halting instruction under the considered states. We explained the process of converting the binary representation into the unary one. States 46 up to 50 are devoted to the location of the needed instruction. By assumption, the $W$ in between the two $S$’s of the configuration of $U$ replaces the $X$ delimiting the instructions associated to the current state of $M$. The counting is performed by erasing a unary symbol and then mark a $Y$ in between $W$ and the leftmost $X$ or $S$ to its right-hand side. When meeting $W$ under state 46, the head turns to state 47 looking after the closest $Y$ while gliding to right. That $Y$ is transformed into $F$ which triggers state 48 and a glide to left back to $S$ where state 49 is called. Under state 49, the head goes back to the closest $h$, gliding other 0’s. When that $h$ is met, it is marked by 0 and, turning to state 50, the head goes back to right to the closest $F$ which marks a $Y$. When that $F$ is met, the head restores the $Y$ and it goes to state 47 in order to find the next $Y$ which has to be marked. The location of the appropriate instruction is obtained when looking after a not yet erased $h$, the head finds $F$ under state 49. It then turns to the next stage of the simulation under state 51.

In Sub-table A1.f, states 51 up to 60, $U$ copies the letter encoded in the instruction to be executed by $M$ onto the square of $M$ tape squared by $M$. First, state 51 goes to the right-hand side $F$ which is replaced by $W$; it is the instruction to be executed. Then, the copying processes as usual: the first 0 or 1 met by state 52 is replaced by $\bar{d}$ or $e$ respectively. The meeting of a 0 triggers state 53 in order to overwrite the first not yet overwritten symbol by $\bar{d}$ under state 54: the crossing of $W$ by state 53 means that the square scanned by $M$ is reached. When the marking is performed, the return to the $W$ of the instruction is obtained by state 55. When it is done, state 52 is again called so that a new cycle of copying one symbol is performed. When 1 is overwritten under state 52, that triggers state 57 which overwrites with $e$ the leftmost not yet transformed symbol of the square scanned by $M$. States 57 and 58 are parallel to states 53 and 54. The return state is 59 which calls state 60 when the $W$ delimiting the instruction is met: state 52 is again called so that a new cycle of copying one symbol is performed. The process is stopped when $h$ is read under state 52: it means that the binary representation of the new letter is completely copied onto the square scanned by $M$. Note that during the glide to left or to right in between both $W$’s glides over $\bar{d}$ and $e$. When looking after a new symbol to be copied, state 52 also glides over the $\bar{d}$’s and $e$’s present in the letter of the instruction.

In Sub-table A1.g, a first part of the states erase the markings: $\bar{d}$’s and $e$’s are back turned to 0’s and 1’s respectively. That operation is performed by states 61 in the instruction, by state 62 which makes the head of $M$ goes back to the scanned square and by state 63 which clears the marking in the scanned square. When it is completed as far as state 63 reads $h$, the head of $U$ goes back to the
instruction, states 64 and 65, in order to look which move of the head of $M$ has to be performed. It is
given by the letter $R$ or $L$ of the instruction transformed into $F$ or $U$ respectively. States 67 up to 70
move $W$ to the right place. As far as $M$ is supposed to be a polite Turing machine, the move to left raises
no problem: the required $U$-delimiter will be found. A move to right is more complex: when going to
right in the scanned square, the head of $U$ under state 70 meets $U$, the encoding of a square follows to
the right-hand side of that $U$. But such an $U$ is missing if $W$ was put on the rightmost square of
the tape of $M$. In that case, the head of $U$ meets _. It is transformed into $U$ and an empty square must be
copied, which is performed by states in Sub-table A1.h.

In Sub-table A1.h, states 71 up to 79 perform the construction of an empty square at the right-hand
side end of the configuration of $M$ and they also control the copying process of the new state
of the head of $M$ onto the scale-pattern at the left-hand side of $F$ and its conversion in a unary represen-
tation. The construction of an empty square is performed by states 71 up to 75. The symbols of
the previously scanned square $Q$ are marked in a copying process. As far as the size of a square of $M$
is not known but as far as, by construction, all squares of $M$ have the same size, it is enough in that
copying process to replicate any symbol of $Q$ as a $h$. In that process, $h$'s, 0's and 1's of $Q$ are one by one
replaced by $Y$, $e$ and $R$ respectively, which is performed by state 74. States 71, 72 and 73 initialise
that process. State 72 copies $h$ on _ each time a symbol of $Q$ has been marked. The construction is
done when under state 74 the head reads $U$. Then under state 75, $U$ replaces by 1 the leftmost $h$ of
the just constructed square and still under state 75 it erases the marks in $Q$, returning $Y$'s, $e$'s and $R$'s
to $h$, 0 and 1 respectively.

State 77 crosses $S$ and then state 78 makes the head of $U$ go back to left to the other $S$ which is
changed to $T$: the head goes to $W$, still marking the state of the executed instruction and turns it back
to $X$. Then, under state 79, the head of $U$ looks after $U$ or $F$ which is restored into $L$ and $R$ respectively
and the new state is 30. It means that the binary representation of a number which is to the right-hand
side of that move symbol is copied onto the scale-pattern to the left-hand side of $F$, see Sub-table A1.d.
State 39 of that table calls states 40 of Sub-table A1.e which converts that binary representation into
unary. When the conversion is completed, state 46 reads $T$, which triggers state 80, the first state of
Sub-table A1.i.

In Sub-table A1.i, the last sub-table of Table A1, states 80 up to 86 perform the location of the new state
of $M$. The principle is the same as that performed in Sub-table A1.e. The leftmost $X$ is replaced
by $F$ by state 80, corresponding to the first marking of a $h$ in the unary representation of the number
of the new state. Then state 84 makes the head of $U$ go to left until it reaches the rightmost _ to the
left-hand side of the configuration of $U$. Then, under state 85, the head marks a new $h$. That marking
changes the state to 81 under which the head goes to right until reaching $T$. There the head changes
to state 82: when meeting $F$, it changes it back to $X$. Then under state 83, it still goes to right, looking
after the next $X$, which is marked and which triggers state 84 so that a new cycle of search starts.
When under state 85 the head meets no more $h$, there are only 0's between $F$ and $T$, so that the head
meets $T$. It changes $T$ to $S$ and it goes to right under state 86 which changes $W$ to the $Y$ it previously
marked and then replaces $F$ by $W$: the new state is located. The new scanned square is also located
so that meeting $S$ under state 86, the head goes to left under state 87 until it meets again the $W$ in
the program of $M$. Then state 29 is called so that a new cycle for simulating the next step of $M$ is
starting.

Accordingly, the pedagogical universal Turing machine we devised has 87 states and 16 letters
which means 1392 instructions. The encoding we described for $M$ can also be used for $U$. It requires
10,351 letters of that encoding. The execution of $U$ on the program of Table 1 as $M$ and on the encoding
of *———*——* as its data requires 106 steps of computation for $M$ and 1,143,717 steps for $U$, as
indicated by the execution of the computer program $P_1$ I devised to check the correctness of the
pedagogical universal Turing machine. The initial configuration of the tape of $M$ in that of $U$ is:

```
SW01hhU11hhU11hhU11hhU01hhU11hhU11hhU01hh,
```
and its final configuration is:

\[ SW01hhU11hhU11hhU11hhU01hhU11hhU11hhU01hhU11hhU01hhU11hhU11hhU11hhU01hh_ \]

as computed by \( P \) which is what was expected.

### 4. Comparison: relevant or not?

Let us now turn to what is raised in the introduction. What can be the basis of a comparison between a universal Turing machine and a virus? Of course, I do not have in mind computer viruses which are something different based on another behaviour of programs which aim at replicate themselves inside as most machines as possible and to invest each infected machine, preventing it to work. A computer virus has an intention, a natural virus has no will. What we say will when speaking of a natural virus, we speak of the result of natural selection on its evolution. As an example, if we say that a natural virus tries to adapt to its hosts, that ‘trying’ is indeed the result of selection: the variant of the virus which is the least malevolent to their hosts has the best chance to survive.

The basis of our comparison is, I think, more profound. It is grounded on the complexity of the things we consider and on their behaviour, mainly the working of a universal Turing machine and the behaviour of a virus at a molecular level.

The unpredictability of the behaviour of a universal Turing machine is a theorem: it is a corollary of both the existence of universal Turing machines and the algorithmic unsolvability of the halting problem for Turing machines from which a lot of corollaries are derived as, for instance, the same impossibility to say whether a Turing machine launched on a given configuration reaches another fixed in advance configuration. Before going on the dispute we postpone to Section 4.2, I give in Section 4.1 the description of how we can encode the code of \( U \) in RNA-terms and how it is possible to deal with that RNA-code.

#### 4.1. How to construct a universal RNA Turing machine

Let us look at the size of a code of a Turing machine, using the encoding defined for the pedagogical universal Turing machine \( U \) in (3) and (6), in Section 3.2. As given in Table 2, the code of that tiny universal Turing machine, say \( N \), requires 206 letters in the just mentioned encoding. The code of \( U \) itself in the same encoding requires 10,351 letters. That encoding is based on the following alphabet: \( _{,} 0, 1, L, R, X, Y, U, W, S, h, d, e, F, Z, T \), considering also the working of \( U \). The genome of a DNA-virus consists in a very long chain of thousands of nucleotides of four kinds: adenine, cytosine, guanine and thymine denoted by \( A, C, G \) and \( T \) respectively, given in alphabetic order. In the case of an RNA virus, the composition of the genome is similar: we have also four kinds of nucleotides, the same ones with the exception of thymine which is replaced by uracil, denoted by \( U \). We later refer to the alphabet \( \{A,C,G,U\} \) as the RNA-alphabet.

With those four letters, we may encode the alphabet of \( U \). For example we can define the following correspondence, encoding a letter of \( U \) by two letters of the RNA-alphabet:

\[
\begin{align*}
S & \rightarrow X \\
X & \rightarrow Y \\
Y & \rightarrow U \\
T & \rightarrow F \\
U & \rightarrow W \\
L & \rightarrow L \\
R & \rightarrow 0 \\
0 & \rightarrow 1 \\
h & \rightarrow d \\
d & \rightarrow e \\
 & \rightarrow Z
\end{align*}
\]

As an example, state 42 of Table 3 looks like that:

\[
\begin{align*}
XY1hhhhhZ010101Y01hhhhL010101Y11hhhhL010101Y001hhhhR110101 \\
Y101hhhhL010101Y011hhhhZ010101Y111hhhhL010101Y001hhhhL010101 \\
Y1001hhZ010101Y011hhhhZ010101Y1101hhL010101Y0011hhL010101 \\
Y1011hhL010101Y0111hhL010101Y1111hhZ010101Y00001hZ010101
\end{align*}
\]
The translation of (8) into the RNA-alphabet is given in (9).

\[
\begin{align*}
\text{AACCCUGAGAGAGAUCACUCUCACUCCACUGAGAGAACCACUCACUCACU} \\
\text{CCCUCUGAGAGAACCACUCACUCUCCACUGAGAGAGAUCACUCUCACUCU} \\
\text{CCCUCUCUGAGAGAACCACUCACUCUCUCUCUCUCUCUCUCUCUCACUCUCU} \\
\text{CCCUCUCUCUGAGAGAACCACUCACUCUCUCUCUCUCUCUCUCUCUCACUCUCU} \\
\text{CCCUCACUCACUGAGAGAACCACUCACUCUCUCUCUCUCUCUCUCUCUCACUCUCU} \\
\text{CCCUCUCUGAGAGAACCACUCACUCUCUCUCUCUCUCUCUCUCUCUCACUCUCU} \\
\text{CCCUCUCUCUGAGAGAACCACUCACUCUCUCUCUCUCUCUCUCUCUCACUCUCU} \\
\text{CCCUCUCUCUCUGAGAGAACCACUCACUCUCUCUCUCUCUCUCUCUCUCACUCUCU}
\end{align*}
\]

(9)

It is important to note that a code similar to (8) is dealt with by \( U \) but a code similar to (9) cannot be used by an RNA Turing machine. Let us see how such a universal RNA Turing machine works, say RNA universal Turing machine, denote it by \( rnaU \). It is assumed that the data used by \( rnaU \) is constituted in the same way as (9), starting from (7). Table A2, in Appendix 2, displays the program of \( rnaU \). In the table, as in Table A1, the halting instructions are represented by an empty entry. Moreover, as the program of Table A1, the program has been checked by a computer program I devised to simulate it, as already mentioned.

The principle of transforming \( U \) into \( rnaU \) is simple: each state of \( U \) contains 16 instructions. With a four-letter alphabet five states at most are needed to sort the instructions corresponding to those of \( U \). Indeed, the execution of an instruction requires to write down two letters in place of the two ones scanned by the machine. That may require one, two, exceptionally four more states as a backward step may be required to overwrite a two symbol pattern. Most often a state is called by just a single other one from a single instruction. The calling instruction \( I \) has a move, say \( \mu \in \{L, R\} \). Most often too, the majority of the instructions under the state \( s \) called by \( I \) have the same move \( \mu \). Most often too, in the calling state, the calling instructions also have the same move. Consequently we have the following two patterns for those sets of instructions:

\[
\begin{align*}
\text{A - C - G - U -} & \quad \text{A - C - G - U -} \\
001 \text{ R002} & - \text{ R003} - \text{ R004} - \text{ R005} - \\
002 \text{ R00X} & - \text{ R00L} - \text{ R00R} - \text{ R00Z} - \\
003 \text{ R00z} & - \text{ R00Y} - \text{ R00L} - \text{ R00Z} - \\
004 \text{ R00h} & - \text{ R00d} - \text{ R00U} - \text{ R00e} - \\
005 \text{ R00T} & - \text{ R00F} - \text{ R00W} - \text{ R00S} - \\
\end{align*}
\]

(10)

To left, in (10), we have the pattern for a move to right, to right of (10), the pattern for a move to left. Note that the ‘instructions’ are in fact pseudo-instructions which should be replaced by the right state corresponding to the letter of \( U \) represented by the position of the considered pseudo-instruction. In the case when several instructions with opposite moves call the same state, we need both sets together. In Table A2, it happens once for states 88 up to 95 which simulate state 21 of \( U \). But in several case, when the state only has two or three non-halting instructions, we need less states and also much less non halting instructions in the RNA version. In (11), (12) and (13) we give three examples of the translation required for \( rnaU \).

\[
\begin{align*}
\text{42} & - \text{ L} - \text{ L} - \text{ R43} - \text{ L} & - \text{ L} & - \text{ L} & - \text{ L} & - \text{ L} \\
\text{A} & - \text{ C} & - \text{ G} & - \text{ U} & - \\
\text{191} & \text{ L192} & - \text{ L193} & - \text{ L194} & - \text{ L195} & - \\
\text{192} & - & \text{ L191} & - \text{ L191} & - & - \\
\text{193} & \text{ R194} & - \text{ L191} & - \text{ L191} & - \text{ L191} & - \\
\text{194} & \text{ L191} & - \text{ R196} & - \text{ L191} & - & - \\
\text{195} & - \text{ L191} & - \text{ L191} & - & - & -
\end{align*}
\]

(11)

Those instructions belong to Table A2, but they are not the code of such a machine. We shall go back to that point in Section 4.2. The piece of a table given in (11) implements the instructions of the state 42.
of $U$ as described in Section 3. Note that (11) applies to 16 instructions of $U$ while Table 3 given in Section 3 displays 10 instructions only for states 40 up to 45. That can be checked in Table A1 of Appendix 1 to the paper. In (12), we can see, to left, the transcription of state 13, to right, that of state 14. In Table A1, both states have only two non halting instructions. In state 13, although the calling instruction and the instructions of state 13 themselves have $R$ as motion, as 1 is encoded $CU$ and $d$ is encoded $GC$, the rewriting of the scanned square requires a step to left in order to write down the appropriate letter followed by another step to right as the move is $R$. The same steps are required in state 14 to rewrite 1 by $h$. But the second letter in the RNA-code is the same for 1 and $h$, which allows us to use the same RNA-state for the step to right as there are halting instructions only for a first RNA-letter $A$. Hence two RNA-states are enough to translate state 14 instead of three ones required to translate state 13.

\begin{align*}
A & - C - G - U -  \\
057 & - R058 & - & - & 060 & - R061 & - & - & (12)
\end{align*}

In (13), with have the unique case where the translation of a state of $U$ requires 9 RNA-states. It concerns state 21 which is called by state 17 and by state 23. In state 17 the calling instruction is to left while it is to right under state 23.

\begin{align*}
A & - C - G - U -  \\
088 & L089 & - L090 & - L091 & - L092 & - 093 & R094 & - - R095 & - R096 & -  \\
089 & - R102 & - CL088 & - CL090 & - 094 & - - AL090 & - -  \\
090 & GR097 & - - L088 & - R102 & - 095 & L089 & - L090 & - L097 & - CL090 & - \\
091 & GR091 & - - AL095 & - R089 & - - R102 & - - - & - & -  \\
092 & - - R089 & - & - & - & (13)
\end{align*}

It can be seen that the instruction from state 17 arrives on $S$, while that from state 23 arrives on $F$.

The number of letters of the code of $U$ is 10,351 so that its translation in the RNA-alphabet yields 20,702 letters. Note that the genome of the SARS-Cov2 virus contains around 31,000 letters, each one being materialised by a nucleotides. The genome of that virus has 15 genes, we shall go back to that point in Section 4.2.

If we consider the RNA-universal machine, its application to the RNA-code of the machine whose program is given in Table 1 requires 2,303,033 steps corresponding to the 106 steps of the courteous addition of that table. It was checked by a computer program $P_2 I$ devised for that purpose. The initial configuration of $rnaU$ is:

\begin{align*}
UUUGCACUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGA  \\
GGCACUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGACG
\end{align*}

and its final configuration is:

\begin{align*}
UUUGCACUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGA  \\
GGCACUCUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGAGGCUCUGAGACG
\end{align*}

In that final configuration, we can note that the head, $UG$, is on the square containing the leftmost $*$ encoded as $CACUGAGA$. We can also check that there are five squares $GGCUCUGAGA$, where CUCU encodes $|$. 
Compared to the 1,143,717 steps of the pedagogical universal Turing machine, the RNA-universal machine requires a little more than twice that number of steps.

At this point, I have to indicate that it is easy to get a smaller universal Turing machine which can still be considered as pedagogic. There are two possible solutions which can also be both applied. The first solution is to note that the construction of the area to the left of $F$ with the unary pattern for the powers of 2 not greater than $P$ requires 29 states. The price to pay is to require that the code of the simulated Turing machine $M$ contains the area constructed by $U$ between the leftmost $S$ and the rightmost $-$ to the left-hand side of that $S$. The other solution consists in noticing that the majority of instructions in $U$ are glides. If we append a new letter, say $M$, we can replace all halting instructions by the code $MDM$ where $D$ is the move and $M$ says that the element at the place of the symbol is the same as in the scanned square for the letter and the new state is the current one.

We conclude that subsection by indicating the RNA-translation of the small universal Turing machine whose instructions are displayed by Table 2. We use (7) and (14) displays the RNA-code which requires 410 letters.

$$
\begin{align*}
\text{AACCCACACUGAACCUCUCUCUGAGAACC} & \\
\text{CCCACUGAGAACCUCUCUCUGAGAAGCAC} & \\
\text{AACCCUCUCUGAGAAGUCACUCUCUCUCUGAGAAGC} & \\
\text{CCUCUCUGAGAAGCCUCUCUCUCUGAGAAGCAC} & \\
\text{AACCCACACUGAACCUCUCUCUCUCUCUGAGAAGC} & \\
\text{CCCACUGAGAACCUCUCUCUCUCUCUGAGAAGCAC} & \\
\text{AACCCUCUCUGAGAAGUCACUCUCUCUCUGAGAAGC} & \\
\text{CCUCUCUGAGAAGCACUCUCUCUCUGAGAAGCAC} & \\
\text{AACCCACACUGAACCUCUCUCUCUCUCUGAGAAGC} & \\
\text{CCCACUGAGAACCUCUCUCUCUCUCUGAGAAGC} & \\
\text{AACCCUCUCUGAACCUCUCUCUCUCUCUGAGAAGC} & \\
\text{CCUCUCUGAGAAGCACUCUCUCUCUCUGAGAAGC} & \\
\text{AACCCACACUGAACCUCUCUCUCUCUCUGAGAAGC} & \\
\text{CCCACUGAGAACCUCUCUCUCUCUCUGAGAAGC} & \\
\end{align*}
$$

(14)

(14)

It is time to turn back to the dispute.

### 4.2. Continuation of the dispute

Clearly, the complexity of the RNA-code obtained from the code of $U$ is significantly less than the size of the SARS-Cov2. It is comparable to the size of an influenza virus. At first glance, as the behaviour of $U$ is unpredictable we could conclude that, a fortiori, the behaviour of the SARS-Cov2 is also unpredictable. Although the history of the pandemia confirms that point up to now, is that comparison of code sizes a relevant argument?

One can certainly raise the following objection: you speak of completely incomparable objects, what a Turing machine has to do with a virus?

At first glance, the objection seems to be sound. Moreover, the tape of a Turing machine is a linear thread like which is also the case of any code written within a segment of the tape even if the segment does not contain any empty square. Indeed, a virus has a complex 3D-structure which is not at all represented by the Turing tape. It can be objected to the objection that a Turing tape is an abstract object. What is important, in the tape, is that it has a linear order. Whether the tape is folded in whichever way is meaningless for the computation. For RNA or DNA strands, the way they are folded may be important as far as elements which are far away from each other in the linear order may become neighbours thanks to the folding. That means more complication on the side of
viruses. Consequently, that objection does not seem to me a serious one. Moreover, the 3D-folding which makes molecules close neighbours has a kind of counter-parts in the Turing technology: in the pedagogical universal Turing machine most instructions of a state refer to a rather neighbouring states according to their numbering, if not the immediate successor or the immediate predecessor. Other references to distant states which are less frequent may be looked as a jump which makes close states which are not according to their numbering. Accordingly, the jumps perform what the folding does.

Another objection could be the fact that in the concept of a Turing machine, at least those considered in the present paper, the tape is infinite in both directions which has no meaning in a biological context. I answered to that objection in underlying that the configuration of a Turing machine is finite at each step of its computation. Remember that namely that property allows us to construct universal Turing machines. I already stressed that point in the section devoted to the pedagogical universal Turing machine. Now, a finite segment which can be continued step by step as long as needed has its relevance in a biological context. Another point on the side of the relevance is the consideration of polite machines only as simulated machines. We mentioned that theoretically that restriction does not alter the generality of the result as far as a non-polite machine can be simulated by a polite one. But such a limitation on the tape means that it has a beginning which is also the case for RNA-strands of a virus.

The complexity is not the single argument in favour of the unpredictability of the behaviour or evolution of viruses. The other reason I have to consider that comparison as relevant lies in the behaviour of the pedagogical universal Turing machine. What are its actions if we only look at the behaviour of the head? We can mainly see two basic features. In many cases, the head runs over the configuration looking after some pattern which is expected to be somewhere on the tape. It is most probably something which also happens within a cell during a replication process. That latter word points at the other main activity of the Turing machine: it copies some portion of the tape over another portion of the tape. What does a virus do? It takes advantage of the copying possibilities contained in a cell to replicate itself as it cannot do that by itself only. Copying and searching something are the basic actions of a Turing machine and those phenomena are also at work at a bio molecular level. But there is a third point. A code by itself can do nothing. It must be treated by a Turing machine in the case of abstract computations, it must be treated by the cell machinery in case of the replication of a virus. The computation of a Turing machine is performed in the head of the human being which created it. The same person may create a computer program in order to perform that computation, in particular to check the correctness of the machine he/she built. It is the reason why at several places I clearly distinguished between a Turing machine and its encoding. As an example, (14) is a code which can be performed by the machine of which (11), (12) and (13) display a few states.

There are simple model of computations, more simple but more abstract than the Turing machine, which are based on that copying process like Post systems, for example, see [1] for a short description. I have no room to consider them here but it is significant that such models are used in order to make the machine of Table 2 able of universal computations. Universality lies in the data too. Which makes us return to the code.

Another point about Turing machines and viruses are the contrast with errors. When a Turing machine works on some data, it is assumed that not only the data can be read by the machine but also that there are no errors in the data. It is also assumed that there is no error in the program of the Turing machine. In the abstract world in which they are living, Turing machines are error free objects which work on error free data without error during the computation. A Turing machine exactly does what is written in its program. Clearly, such a perfect behaviour is far away from what happens in a biological context. However, things are not that different if we look at Turing machines as objects created by human brains. How does a human being who decides to create a Turing machine in order to solve a given problem? He/she writes down the program and submit it with the data to a simulator of Turing machines. Whether the simulator is borrowed from the web or it is created by the same person, there are several trials before the human being obtains a machine he/she can trust as error free. Those
trials are interesting as far as there are usually errors during that tuning process. There are errors in the
data which are not completely conformal to the format which they are supposed to obey. There are
also errors in the Turing machine which does not behave as expected. Sometimes, the Turing machine
halts because no instruction corresponds to what it reads under its current state. Sometimes too, the
Turing machine runs a so long time without halting that its creator thinks it never halts. Looking care-
fully, the human being detects an error in his/her program or in the data he/she wrote. Both cases may
also happen together. Accordingly, data and program mute until they arrive to a sufficiently stable
form. That real-life process of creating a Turing machine looks like some evolution process of real life.
Things occur as just described as far as nobody can prove the correctness of a program as far as the
controlling tools are guaranteed to work just by long enough trials. We reach here the same limit to
the power of universal Turing machine: there are (infinitely many) problems which they cannot solve.
That latter sentence is a theorem.

Another interesting point consists in the following statistics: among the 1652 instructions of the
RNA universal Turing machine (4 letters and 413 states), 958 of them are glides: the head does not
overwrite the scanned square. Also, among those 1652 instructions, 542 of them are halting instruc-
tions, so that we remain with 152 instructions which overwrite the scanned square with another letter:
it means less than 10% of the instructions make changes on the tape. It looks like the situation in the
genome where active coding elements are a small minority, which does not mean that the others are
useless as is the case in the RNA universal Turing machine: only 45 instructions among the non-halting
one does not change the state of the head. Moreover among those 45 instructions, 24 of them replace
the letter of the scanned square by another one. The big number of changes of states comes from the
fact that the pedagogical universal Turing machine $U$ has 16 letters in its alphabet while the RNA uni-
versal Turing machine has only 4 of them at its disposal. In Table A2, we dispatch the table of that latter
machine by grouping its states according to those of $U$ which they are simulating. In each such groups,
the first state dispatches the execution into at most four states in order to sort what is read according
to the letter it represents in terms of those of $U$. Note that the operations of $U$ can be split into nine
parts, so that in mean, from 30 up to 53 states of the RNA-universal Turing machine correspond to a
specific action. More precisely, those groups are the following:

\[
\begin{align*}
1 &- 53 :: 54 - 101 :: 102 - 132 :: 133 - 182 :: 183 - 231 :: \\
232 &- 281 :: 282 - 331 :: 332 - 373 :: 374 - 413
\end{align*}
\]

The number of genes in a virus is very small. Let us remember that, as an example, the SARS-Cov2 has
15 of them.

Assuming that the comparison I raised with the present paper is relevant, what can we infer from
that?

The first conclusion is the unpredictability of the evolution of a virus. All reasons which were raised
against the relevance of the comparison point at the same conclusion: a virus is much more complex
than a Turing machine so that if the behaviour of the latter is unpredictable, it is all the more the case
for the former. Does it mean that there is nothing to do to limit as strongly as possible the damage a
virus can do when it is virulent? Certainly not. The comparison I consider means that something which
looks like algorithmic has little chance to fight a virus. At that point, let us go back to the comparison.
Computer scientists introduced sometimes a topology on data from which it can be proved that Turing
machines viewed as operators on the data are continuous. The notion of continuity corresponding to
that topology can be stated in a more concrete way: for a given Turing machine, if it is given long
enough data and if there are no differences between the data within that length, the Turing machine
gives the same output for those data. The larger that length with no difference is, the finer, we say, the
continuity is. In other words, if different data entail the same behaviour of the machine, it means that
the considered data are big and that ‘close to the machine’ we could say, they are not different. Every
thing relies on the fineness of that continuity.
The consequences of that continuity theorem are that viruses try to cheat on the immune system of their hosts, it is the way they can survive. When I say that ‘the virus tries’ it is a short circuit for the sentence: among the ‘child’-virions produced by the ‘parent’-virions, those which are the more adapted to the present situation survive and only them. From the side of the immune system, its efficiency is a kind of measure of the fineness of its continuity. If it can be cheated only when data are very large and are little different, then it will be difficult for a virus to deceive it. Viruses and our immune systems have a long history of constant struggle one against the other.

5. Conclusion

Homo sapiens sapiens is very proud of the knowledge that could be accumulated thanks to science. Let us remember that the basis of today sciences stems from late 16th century, mathematics being a bit older. Certainly it is a little duration at the geological scale. Indeed, that means less than 500 years, around 3000 years for mathematics. Life on our planet exists from over than 3,000,000,000 years. For sure, life evolution requires much time to evolve from viruses and bacteria to plants, to animals, to man. Let us stress that human beings are newcomers at geological scale. The way life appeared make all its components interact, and that was apparently always the case. Science of which homo sapiens sapiens is so proud is the work of how many people? Certainly less than 1 million and a half persons at world scale at the present moment and we have to compare that with the more than 7,000,000,000 human beings at present. Moreover, the number of to day scientists is several times more than the number of dead scientists since the beginning of science, at some point 5000 years ago, roughly speaking. That large estimate shows us that scientific activity, whatever the appraisal of its results, is a very partial and very recent activity of mankind. Can it allow us to play with our environment as it were something given at our free use and free will? It seems to me that the answer is no. We should be careful when dealing with our environment. We should try to foresee as far as possible the consequences of our actions on the environment.

Let us be more modest and let us try to take benefit of what can we learn from the other fields of science in each field of science. Theoretical computer scientists observe their environment and that careful look brought them valuable results. Perhaps Turing machine may be of help to better understand phenomena studied by other sciences. When computation and information are involved we may have relevant points of view for other people. Theoretical computer science brings some light to philosophical problems too, but that point goes for beyond the goal of that paper.

I hope that the reader will find those comments of interest, whatever his/her point of view.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

[1] Minsky ML. Computation: finite and infinite machines. Englewood Cliffs: Prenice Hall; 1967.
[2] Margenstern M. Frontier between decidability and undecidability: a survey. Theor Comput Sci. 2000;231(2):217–251.
[3] Woods D, Neary T. The complexity of small universal turing machines: a survey. Theor Comput Sci. 2009;410(4–5):443–450.
[4] Rogozhin Yu. V. Seven universal turing machines. In: Systems and theoretical programming, matematicheskie issledovania, Vol. 69, Kishinev: Akademia Nauk Moldavskoi SSR; 1982. p. 76–90 (in Russian).
[5] Rogozhin Yu., Small universal turing machines. Theor Comput Sci. 1996;168:215–240.
[6] Neary T, Woods D. Four small universal turing machines. Fundam Inform. 2009;91(1):123–144.
Appendices

Appendix 1.

Table A1. Table of $U$: it is divided into several sub-tables where the columns are labelled by the states and the lines are labelled by the letters. That latter display is more convenient, taking into account the number of states.

| Sub-table A1.a: building the area between $S$ and $F$. $T$ is missing as not used at that stage. |
|-----------------------------------------------|
| 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   |
| _    | 0R05 | 1R08 | FR12 |
| 0    | L    | R    | L    | R    | L    | 1R08 | R    | R    | L    | L    |
| 1    | L    | R    | L    | R    | L    | R    | R    | L    | L    | L    |
| L    | L    | R    | L    | R    | L    | R    | R    | L    | L    | L    |
| R    | L    | R    | L    | R    | L    | R    | R    | L    | L    | L    |
| X    | L    | FR03 | L06  | R    | R    | FL10 | L    | L    |     |     |
| Y    | L    | WL04 | L    | R    | L    | R    | R    | L    | L    | L    |
| U    | W    | SL   | L    | YR03 |
| S    | R02  | L    | R    | L    | R    | L    | R    | L    | L    | L    |
| h    | L    | R    | L    | R    | L    | R    | R    | L    | L    | L    |
| d    | e    | F    | L    | R    | L    | R    | L    | L    | L    | L    |
| dR13 | dR14 | hR15 | dL16 | hL18 | L20  | dR16 |
| R    | L    | R    | L    | R    | hL22 | hL   |
| S    | L21  | L20  |
| h    | RR17 | R    | L    | R    | dL16 | 0L   | L    |
| d    | eR17 | R    | L    | R    | dL16 | 0L   | L    |
| e    | L    | R    | L    | R    | dR   | dL   |
| F    | R19  | R23  | R23  |

Sub-table A1.b: States for constructing the scale of powers of two not greater than $P$. The missing letters, namely _, 0, L, X, Y, U, W and T, involve halting instructions. It is the reason why the corresponding lines are missing in the table.

| 12   | 13   | 14   | 15   | 16   | 17   | 18   | 19   | 20   | 21   | 22   |
|------|------|------|------|------|------|------|------|------|------|------|
| _    | dR13 | dR14 | hR15 | dL16 | hL18 | L20  | dR16 |
| 1    | R    | L    | R    | L    | R    | hL22 | hL   |
| S    | L21  | L20  |
| h    | RR17 | R    | L    | R    | dL16 | 0L   | L    |
| d    | eR17 | R    | L    | R    | dL16 | 0L   | L    |
| e    | L    | R    | L    | R    | dR   | dL   |
| F    | R19  | R23  | R23  |

Sub-table A1.c: To left: states for preparing the pattern to the left-hand side of $F$ allowing to convert the binary representation of $n \leq P$ into $h^n$. The following symbols are not concerned by the states 23 up to 27: L, R, X, Y, U, W, e, Z and T. To right, on two sub-tables, the motion of the head to the $W$ indicating the scanned square on the tape of $M$. Here too a few symbols are not concerned: _, d, e, F and T.

| _    | dR25 | 0R25 | 0    | R    | R    | U    | R    |
| 0    | 1R   | L    | R    | L    | 1    | R    | R    | W    | R30  |
| 1    | R    | L    | L    | 0L   | L    | R    | R    | S    | R    | R    |
| S    | L27  | R    | R    | R    | h    | R    |
| h    | 1L26 | R    | X    | WR29 | R    | Z    | R    |
| d    | 1L24 | L    | R    | L    | Y    | R    |
| F    | R21  | L    | R23  | L    | R28  |

(continued)
Table A1. Continued.

Sub-table A1.d: States for copying the binary representation of $n$ in the scanned square of the tape of $M$ onto the pattern of the useful powers of $2$. Here there is no missing letter as far as each letter is read under at least one state among those of that part of the table.

| 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
|----|----|----|----|----|----|----|----|----|----|
| _  | dL31| L  | L  | R  | R  | L  | L  | L  | R  |
| 1  | eL35| L  | L  | R  | R  | L  | L  | L  |   |
| L  | L  | R  | R  | L  | L  |   |   |   |   |
| R  | L  | R  | R  | L  | L  |   |   |   |   |
| X  | L37 | L  | R  | R  | L  | L  |   |   |   |
| Y  | L37 | L  | R  | R  | L  | L  |   |   |   |
| U  | L  | R  | R  | L  | L  |   |   |   |   |
| W  | L  | R  | R  | L  | L  |   |   |   |   |
| S  | L  | R  | R  | L  | L  |   |   |   |   |
| h  | L37 | L  | R  | R  | L  | L  | L  | R  |
| d  | R  | L  | hR33 | 0R30 | eR33 | 0L | L  | R  |
| e  | R  | L  | L  | eL45 | 1R30 | L  | L  | 1L | L  | LR40 |
| F  | L32 | R  | L  | L  | L  | L36 | L  | 38 |
| Z  | L  | R  | R  | L  | L  |   |   |   |   |
| T  | L  | R  | R  | L  | L  |   |   |   |   |

Sub-table A1.e: Left-hand side half, states 40 up to 45: transformation of the binary representation of $n$ copied to the right-hand side of $F$ into $n$ copies of $h$ to the right-hand side of $F$. Those states are used twice during the simulation of one step of $M$ computation. Right-hand side half, states 45 up to 46: thanks to the unary representation of $n$, location of the instruction to be applied under the state marked by $W$.

| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
|----|----|----|----|----|----|----|----|----|----|----|
| 0  | R  | hL42 | L  | YR40 | R  | R  | R  | L  | L  | R  |
| 1  | L  | L  | R  | R  | L  | R  | L  |   |   |   |
| L  | L  | R  | eL45 | 0R40 | eR45 | 0L | L  | R  |
| R  | L  | R  | R  | L  | R  | L  | R  |   |   |   |
| X  | L  | R  | R  | L  | L  |   |   |   |   |   |
| Y  | L  | R  | 0L | R  | FL48 | L  | R  |   |   |   |
| U  | R  | L  | R  | dL |   |   |   |   |   |   |
| W  | R  | L  | R  | dL |   |   |   |   |   |   |
| S  | R  | L  | R  | L  | L  | L49 | R  |   |   |   |
| h  | R  | R  | L  | L  | UR40 | 0R46 | R  | R  | L  | OR50 |
| d  | hR | L  | L  | R  | R  |   |   |   |   |   |
| e  | R  | L  | RR40 | LR40 | R  |   |   |   |   |   |
| F  | R41 | L42 | L  | L44 | R  | R51 | YR47 |   |   |   |
| Z  | L  | 84 | R  | R  | L  | R  |   |   |   |   |
| T  | L  | 80 |   |   |   |   |   |   |   |   |

Sub-table A1.f: States 51 up to 60. Copying the letter of the instruction onto the square scanned by $M$.

| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
|----|----|----|----|----|----|----|----|----|----|
| 0  | R  | dR53 | R  | dL55 | L  | R  | eL59 | L  |  |
| 1  | R  | eR57 | R  | dL55 | L  | R  | eL59 | L  |   |
| L  | R  | R  | R  | L  | R  | L  | L  |   |   |
| R  | R  | R  | R  | L  | R  | L  | L  |   |   |
| X  | R  | R  | R  | L  | R  | L  | L  |   |   |
| Y  | R  | R  | R  | L  | R  | L  | L  |   |   |
| U  | R  | R  | R  | L  | R  | L  | L  |   |   |
| W  | R  | R  | R  | L  | R  | L  | L  |   |   |
| h  | R  | L61 | R  | dL55 | L  | R  | eL59 | L  |   |
| d  | R  | R  | L  | L52 | R  | R  | L  | L  |   |
| e  | R  | R  | L  | L52 | R  | R  | L  | L  |   |
| F  | W  | R52 |   |   |   |   |   |   |   |
| Z  | R  | R  | L  | R  | L  | L  |   |   |   |
| T  | R  | R  | L  | R  | L  | L  |   |   |   |

(continued)
Table A1. Continued.

Sub-table A1.g: States 61 up to 70. Return to the scanned square in order to erase the markings and then return to clear the instruction from the markings. Then, $U$ performs the move of the head of $M$ over its tape.

| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
|----|----|----|----|----|----|----|----|----|----|
| _  | _  | _  | _  | _  | _  | _  | _  | _  | _  |
| 0  | L  | R  | hR | L  | L  | R  | R  | L  | R  | R  |
| 1  | L  | R  | hR | L  | L  | R  | R  | L  | R  | R  |
| L  | R  | L  | L  | UR67| R  | R  | R  | R  | R  | R  |
| R  | R  | L  | L  | FR69| R  | R  | R  | R  | R  | R  |
| X  | R  | R  | L  | L  | R  | R  | R  | R  | R  | R  |
| Y  | R  | R  | L  | L  | R  | R  | R  | R  | R  | R  |
| U  | R  | L  | L  | R  | WL76| R  | WL76|WL76|WL76|WL76|
| W  | R62| R63| L65| R66| UL68| UR70|UL71|UL71|UL71|UL71|

Sub-table A1.h: States 71 up to 79. Creating a new empty square at the right-hand side end of the tape of $M$. The first states also clear the marking performed during the action indicated by sub-table A1.g. We also have a new call to states 30 up to 45 to copy the binary representation of the new state onto the scale-pattern to the left-hand side of $F$ and to compute to the right-hand side of it the unary representation of that number.

| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
|----|----|----|----|----|----|----|----|----|
| _  | _  | _  | _  | _  | _  | _  | _  | _  |
| 0  | hL | 73 | eR72| L  | L  | R  | R  | R  | R  |
| 1  | RR72| R  | L  | L  | R  | R  | R  | R  | R  |
| L  | R  | L  | 1R | L  | R  | R  | R  | R  | R  |
| R  | R  | L  | R  | L  | R  | R  | R  | R  | R  |
| Y  | R  | L  | hR | L  | L  | R  | R  | R  | R  |
| U  | R  | L74| R75| WR | L  | L  | LR30|LR30|LR30|LR30|
| W  | L  | L  | XR79| R  | S  | L77|TR78|TR78|TR78|TR78|
| h  | YR72|R  | L  | YR72| 1L76| L  | L  | R  | R  |
| e  | R  | L  | 0R | L  | L  | R  | R  | R  | R  |
| F  | L  | L  | RR30| L  | L  | L  | L  | L  | L  |
| Z  | L  | R  | R  | L  | L  | R  | R  | R  | R  |

Sub-table A1.i: States 70 up to 87. Locating the new state and, when it is obtained, start the simulation of the next step of $M$ in its computation.

| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 |
|----|----|----|----|----|----|----|----|
| _  | _  | _  | _  | _  | _  | _  | _  |
| 0  | R  | R  | R  | R  | L  | R  | R  |
| 1  | R  | R  | R  | R  | L  | R  | R  |
| L  | R  | R  | R  | R  | L  | R  | R  |
| R  | R  | R  | R  | R  | L  | R  | R  |
| X  | FL84| R  | FL84| L  | R  | L  | L  |
| Y  | R  | R  | R  | R  | L  | R  | L  |
| W  | YR | L  | YR | L  | YR | L  | R29|
| S  | L87| L87| L87| L87| L87| L87| L87|
| h  | R  | R  | R  | R  | L  | OR81| R  | L  |
| d  | L  | L  | L  | L  | L  | L  | L  |
| F  | XR83| L  | R  | WR | L  | R  | L  |
| Z  | R  | R  | R  | R  | L  | R  | L  |
| T  | R82| L  | SR86| R  | L  | L  | L  |
Appendix 2.

Table A2. The RNA-universal Turing machine: it works with the encoding defined both by that of the and by its translation in the RNA-alphabet. The label state n indicates the state of the which is simulated by the states of the RNA-universal Turing machine which follows it. Empty entries are halting instructions.

|   | A  | -  | C  | -  | G  | -  | U  | -  |
|---|----|----|----|----|----|----|----|----|
| state 1 |  |    |    |    |    |    |    |    |
| 1 | L2 | -  | L3 | -  | L4 | -  | L5 | -  |
| 2 | L1 | -  | L1 | -  | L1 | -  | R6 | -  |
| 3 | L1 | -  | L1 | -  | UL | -  | L1 | -  |
| 4 | L1 | -  | -  | -  | -  | -  | R3 | -  |
| 5 | -  | L1 | -  | -  | -  | R2 | -  |
| state 2 |  |    |    |    |    |    |    |    |
| 6 | R7 | -  | -  | -  | -  | -  | -  | -  |
| 7 | CL8| -  | R9 | -  | -  | -  | -  | -  |
| 8 | UR7| -  | -  | -  | -  | -  | -  | -  |
| state 3 |  |    |    |    |    |    |    |    |
| 9 | R10| -  | R11| -  | R12| -  | R13| -  |
| 10 | L13| -  | R9 | -  | R9 | -  | R9 | -  |
| 11 | R9 | -  | GL12| -  | -  | -  | R9 | -  |
| 12 | R9 | -  | UL14| -  | -  | -  | -  | -  |
| 13 | L24| -  | R9 | -  | -  | -  | -  | -  |
| state 4 |  |    |    |    |    |    |    |    |
| 14 | L15| -  | L16| -  | L17| -  | L18| -  |
| 15 | -  | L14| -  | L14| -  | -  | -  | -  |
| 16 | L14| -  | L14| -  | -  | -  | L14| -  |
| 17 | L14| -  | R  | -  | AR19| -  | L14| -  |
| 18 | L14| -  | L14| -  | -  | -  | L14| -  |
| state 5 |  |    |    |    |    |    |    |    |
| 19 | R20| -  | R21| -  | R22| -  | R23| -  |
| 20 | R19| -  | R19| -  | R19| -  | R19| -  |
| 21 | R19| -  | R19| -  | -  | -  | R19| -  |
| 22 | R19| -  | R9 | -  | -  | -  | CR | -  |
| 23 | -  | R19| -  | -  | -  | -  | R19| -  |
| state 6 |  |    |    |    |    |    |    |    |
| 24 | L25| -  | L26| -  | L27| -  | L28| -  |
| 25 | -  | L24| -  | L24| -  | -  | -  | -  |
| 26 | L24| -  | L24| -  | -  | -  | L29| -  |
| 27 | L24| -  | -  | -  | -  | -  | -  | -  |
| 28 | L24| -  | -  | -  | -  | -  | -  | -  |
| state 7 |  |    |    |    |    |    |    |    |
| 29 | L30| -  | -  | -  | -  | -  | L32| -  |
| 30 | UR33| -  | R  | -  | UR33| -  | -  | -  |
| 31 | -  | R30| -  | -  | -  | -  | -  | -  |
| 32 | -  | L29| -  | -  | -  | -  | L29| -  |
| state 8 |  |    |    |    |    |    |    |    |
| 33 | R34| -  | R35| -  | R36| -  | R37| -  |
| 34 | R33| -  | R33| -  | R33| -  | R33| -  |
| 35 | R33| -  | R33| -  | -  | -  | R33| -  |
| 36 | R33| -  | -  | -  | -  | -  | AR37| -  |
| 37 | R38| -  | AL36| -  | -  | -  | R33| -  |
| state 9 |  |    |    |    |    |    |    |    |
| 38 | R39| -  | R40| -  | R41| -  | R42| -  |
| 39 | CL42| -  | R38| -  | R38| -  | R38| -  |
| 40 | R38| -  | R38| -  | -  | -  | R38| -  |
| 41 | R38| -  | -  | -  | -  | -  | L49| -  |
| 42 | UL44| -  | -  | -  | -  | -  | L41| -  |
| 43 | -  | -  | -  | -  | -  | -  | L44| -  |

(continued)
Table A2. Continued.

| State 10 | A  | C  | G  | U  |
|----------|----|----|----|----|
| 44       | L45| L46| L47| L48|
| 45       | L44| L44| L44|    |
| 46       | L44| L44|    |    |
| 47       | L44|    |    |    |
| 48       | L44|    |    | L29|
| State 11 |    |    |    |    |
| 49       | L50| L51| L52| L53|
| 50       | L49| L49| L49|    |
| 51       | L49| L49| CR54|    |
| 52       | L49| UR51|    |    |
| 53       | L49|    |    | L49|
| State 12 |    |    |    |    |
| 54       |    | R55|    |    |
| 55       |    | GR56| CL|    |
| 56       |    | R57|    |    |
| State 13 |    |    |    |    |
| 57       |    | R58|    |    |
| 58       |    | GR59| CL|    |
| 59       |    | R60|    |    |
| State 14 |    |    |    |    |
| 60       |    | R61|    |    |
| 61       | R62| GR| AL|    |
| State 15 |    |    |    |    |
| 62       |    | R63|    |    |
| 63       |    | GL65| CL|    |
| 64       |    | L65|    |    |
| State 16 |    |    |    |    |
| 65       | L66| L67| L68| L69|
| 66       | G70| L80| AR|    |
| 67       |    | UR70| R|    |
| 68       | L65|    |    |    |
| 69       |    |    | L65|    |
| State 17 |    |    |    |    |
| 70       | R71| R72| R73| R74|
| 71       |    | GL75| R70| L88|
| 72       |    |    | AL71|    |
| 73       | R70| R70| R70|    |
| 74       |    |    | L71|    |
| State 18 |    |    |    |    |
| 75       | L76| L77| L78| L79|
| 76       |    | UR70| L75|    |
| 77       |    | R76|    |    |
| 78       | L75|    |    |    |
| 79       |    |    | L65|    |
| State 19 |    |    |    |    |
| 80       | R81| R82| R83| R84|
| 81       |    | R80| L85|    |
| 82       |    | L85| L|    |
| 83       | R80|    | R80|    |
| 84       |    |    | L81|    |

(continued)
|        | A  | C  | G  | U  |     |
|--------|----|----|----|----|------|
| state 20 | -  | -  | L87 | -  | -    |
| 85     | L86 | -  | -  | R  | CL   |
| 86     | CL87| -  | -  | R  | CL   |
| 87     | GR86| L65| -  |    | -    |
| state 21 | -  | -  | R86 | L65| -    |
| 88     | L90 | L91| L92 | -  | -    |
| 89     | R102| CL88| CL90| -  | -    |
| 90     | GR97| L88| R102| -  | -    |
| 91     | GR  | AL95| R89 | -  | -    |
| 92     | R94 | L95| R96 | -  | -    |
| 93     | AL90| -  | R96 | -  | -    |
| 94     | -   | AL90| -  | -  | -    |
| 95     | L89 | L90| L97 | CL90| -   |
| 96     | R102| -  | -  | -  | -    |
| state 22 | -  | -  | R89 | -  | -    |
| 97     | L98 | L99| L100| L101| -   |
| 98     | R102| L97| CL  | -  | -    |
| 99     | -   | AL98| R98 | -  | -    |
| 100    | GR99| -  | -  | -  | -    |
| 101    | -   | R98 | -  | -  | -    |
| state 23 | -  | R103| R104| R105| -   |
| 102    | -   | R104| R105| -  | -    |
| 103    | UR102| -  | R105| R106| -   |
| 104    | UL103| UL | -  | -  | -    |
| 105    | -   | R88 | -  | -  | L120 |
| state 24 | -  | R104| R105| -  | -    |
| 106    | L107| L108| L109| L110| -   |
| 107    | L106| CR111| -  | -  | -    |
| 108    | -   | L106| -  | -  | -    |
| 109    | -   | GR107| -  | -  | -    |
| 110    | -   | L106| -  | -  | -    |
| state 25 | -  | R112| R113| R114| -   |
| 111    | -   | R113| R114| -  | -    |
| 112    | R111| -  | -  | -  | -    |
| 113    | R111| -  | -  | -  | -    |
| 114    | -   | R102| -  | -  | -    |
| state 26 | -  | R112| R113| R114| -   |
| 115    | L116| L117| L118| R119| -   |
| 116    | L115| L118| R119| -  | -    |
| 117    | -   | L115| L117| -  | -    |
| 118    | -   | L115| L117| -  | -    |
| 119    | -   | AR111| -  | -  | -    |
| state 27 | -  | AR111| -  | -  | -    |
| 120    | -   | L121| L122| -  | -    |
| 121    | -   | R124| R  | -  | AL123 |
| 122    | -   | R  | -  | -  | AL123 |
| 123    | -   | L120| -  | -  | -    |
| state 28 | -  | R126| R  | -  | R127 |
| 124    | R125| R126| R  | -  | R127 |
| 125    | GL127| R126| R  | -  | R127 |
| 126    | R124| R124| R124| R  | R127 |
| 127    | UR126| -  | R124| R  | R127 |
| state 29 | -  | R130| R  | -  | R127 |
| 128    | R129| R130| R131| R  | R132 |
| 129    | R128| R128| R131| R  | R132 |
| 130    | R128| R128| R131| R  | R132 |
| 131    | R128| GL138| R128| R  | R132 |
| 132    | -   | R133| R  | -  | R132 |

(continued)
Table A2. Continued.

|     | A   | C   | G   | U   |
|-----|-----|-----|-----|-----|
| state 30 |     |     |     |     |
| 133 | R134 |   | R135 |   | R136 |   | R137 |   |
| 134 | L137 |   | L169 |   | L169 |   |   |   |
| 135 | CL131 |   | L134 |   |   |   | L137 |   |
| 136 | L134 |   | R133 |   |   |   | R133 |   |
| 137 | L169 |   |   |   |   |   |   |   |
| state 31 |     |     |     |     |
| 138 | L139 |   | L140 |   | L141 |   | L142 |   |
| 139 | L138 |   | L138 |   | L138 |   | L138 |   |
| 140 | L138 |   | L138 |   | L138 |   | L143 |   |
| 141 | L138 |   |   |   | L138 |   | L138 |   |
| 142 | L138 |   | L138 |   | L138 |   | L138 |   |
| state 32 |     |     |     |     |
| 143 | L144 |   |   |   | L146 |   | L147 |   |
| 144 |   | L143 |   | L143 |   |   |   |
| 145 |   | AR148 |   | R |   |   |   |   |
| 146 |   |   |   | L148 |   |   |   |
| 147 |   |   |   | L148 |   |   |   |
| state 33 |     |     |     |     |
| 148 | R149 |   | R150 |   | R151 |   | R152 |   |
| 149 | R148 |   | R148 |   | R148 |   | R148 |   |
| 150 | R148 |   | R148 |   | R148 |   | R148 |   |
| 151 | R148 |   |   |   | R148 |   | R148 |   |
| 152 | R148 |   | R148 |   | R153 |   | R148 |   |
| state 34 |     |     |     |     |
| 153 | R154 |   | R155 |   | R156 |   | R157 |   |
| 154 | R153 |   | R153 |   | R153 |   | R153 |   |
| 155 | R153 |   | R153 |   | CR158 |   | R153 |   |
| 156 | R153 |   | AL155 |   | R153 |   | L158 |   |
| 157 | R153 |   |   |   | R133 |   | R153 |   |
| 158 | R133 |   |   |   | CR |   | R133 |   |
| state 35 |     |     |     |     |
| 159 | L160 |   | L161 |   | L162 |   | L163 |   |
| 160 | L159 |   | L159 |   | L159 |   | L159 |   |
| 161 | L159 |   | L159 |   | L159 |   | L164 |   |
| 162 | L159 |   |   |   | L159 |   | L159 |   |
| 163 | L159 |   | L159 |   | L159 |   | L159 |   |
| state 36 |     |     |     |     |
| 164 | L165 |   | L166 |   | L167 |   | L168 |   |
| 165 |   | L164 |   | L164 |   |   |   |
| 166 |   | UR148 |   | R |   |   |   |   |
| 167 |   |   |   |   |   |   |   |   |
| 168 |   |   |   | L164 |   |   |   |
| state 37 |     |     |     |     |
| 169 | L170 |   | L171 |   | L172 |   | L173 |   |
| 170 | L169 |   | L169 |   | L169 |   | L169 |   |
| 171 | L169 |   | L169 |   | CR172 |   | L174 |   |
| 172 | L169 |   | AL171 |   | L169 |   | L169 |   |
| 173 | L169 |   | L169 |   | CL169 |   | L169 |   |
| state 38 |     |     |     |     |
| 174 | L175 |   | L176 |   | L177 |   | L178 |   |
| 175 |   | L174 |   | L174 |   |   |   |
| 176 |   |   |   | L174 |   |   |   |
| 177 |   |   | R | R179 |   |   |   |
| 178 |   | L174 |   | L174 |   |   |   |
| state 39 |     |     |     |     |
| 179 |   | R180 |   | R181 |   | R182 |   |
| 180 | R179 |   |   | AR182 |   |   |   |
| 181 | R179 |   | R179 |   |   | CL180 |   |   |
| 182 |   | R183 |   |   |   |   |   |

(continued)
| A    | C    | G    | U    |
|------|------|------|------|
| state 40 |
| 183  | R184 | -    | R184 | -    |
| 184  | R183 | -    | AR183| R183 |
| 185  | R183 | -    | R183 | -    |
| 186  | -    | R187 | -    | -    |
| state 41 |
| 187  | -    | R188 | -    | R189 |
| 188  | L    | -    | GL191| -    |
| 189  | R187 | -    | -    | -    |
| 190  | -    | -    | -    | -    |
| state 42 |
| 191  | L192 | -    | L193 | -    |
| 192  | -    | L191 | -    | L191 |
| 193  | R194 | -    | L191 | -    |
| 194  | L191 | -    | R196 | -    |
| 195  | -    | L191 | -    | L191 |
| state 43 |
| 196  | R197 | -    | R198 | -    |
| 197  | L193 | -    | R196 | -    |
| 198  | CR183| -    | R196 | -    |
| 199  | GR183| -    | R196 | -    |
| 200  | -    | L198 | -    | R183 |
| state 44 |
| 201  | -    | L202 | -    | L203 |
| 202  | GR203| -    | R204 | -    |
| 203  | GR202| -    | R206 | -    |
| 204  | GR203| -    | AL205| -    |
| 205  | -    | L201 | -    | CL204|
| state 45 |
| 206  | -    | R207 | -    | R208 |
| 207  | R206 | -    | R183 | -    |
| 208  | L207 | -    | R206 | -    |
| 209  | R210 | -    | R206 | -    |
| state 46 |
| 210  | R211 | -    | R212 | -    |
| 211  | R210 | -    | R210 | -    |
| 212  | R210 | -    | R210 | -    |
| 213  | R210 | -    | -    | -    |
| 214  | R374 | -    | -    | R215 |
| state 47 |
| 215  | R216 | -    | R217 | -    |
| 216  | -    | R215 | -    | R215 |
| 217  | R215 | -    | L218 | -    |
| 218  | R215 | -    | UL220| -    |
| 219  | -    | -    | -    | -    |
| state 48 |
| 220  | L221 | -    | L222 | -    |
| 221  | L220 | -    | L220 | -    |
| 222  | L220 | -    | L220 | -    |
| 223  | L220 | -    | -    | L220 |
| 224  | L220 | -    | L220 | -    |
| state 49 |
| 225  | L226 | -    | L227 | -    |
| 226  | R228 | -    | L225 | -    |
| 227  | -    | R233 | -    | R    |
| state 50 |
| 228  | R229 | -    | R230 | -    |
| 229  | R228 | -    | R228 | -    |
| 230  | R228 | -    | R228 | -    |
| 231  | R228 | -    | R215 | -    |
| 232  | -    | L231 | -    | R228 |

(continued)
Table A2. Continued.

|     | A   | C   | G   | U   |
|-----|-----|-----|-----|-----|
| state 51 |     |     |     |     |
| 233 | R234 | -   | R235 | -   | R236 | -   | R237 | -   |
| 234 | R233 | -   | R233 | -   | R233 | -   | R233 | -   |
| 235 | R233 | -   | R233 | -   | R233 | -   | R233 | -   |
| 236 | R233 | -   | -   | -   | -   | -   | -   | -   |
| 237 | -   | GR238 | -   | R233 | -   | R233 | -   |
| state 52 |     |     |     |     |
| 238 | R239 | -   | R240 | -   | R241 | -   | -   | -   |
| 239 | -   | GR242 | -   | CR  | -   | -   | -   | -   |
| 240 | CL239 | -   | R238 | -   | L282 | -   | L239 | -   |
| 241 | L240 | -   | R238 | -   | -   | -   | R238 | -   |
| 242 | -   | R243 | -   | -   | -   | -   | -   | -   |
| state 53 |     |     |     |     |
| 243 | R244 | -   | R245 | -   | R246 | -   | R247 | -   |
| 244 | R243 | -   | R243 | -   | R243 | -   | R243 | -   |
| 245 | R243 | -   | R243 | -   | -   | -   | R243 | -   |
| 246 | R243 | -   | -   | -   | -   | -   | -   | -   |
| 247 | R243 | -   | -   | -   | -   | -   | R248 | -   |
| state 54 |     |     |     |     |
| 248 | R249 | -   | R250 | -   | R251 | -   | R252 | -   |
| 249 | -   | GL253 | -   | L253 | -   | -   | -   | -   |
| 250 | CL249 | -   | -   | -   | -   | CL249 | -   |
| 251 | CL249 | -   | R248 | -   | -   | -   | R248 | -   |
| 252 | -   | -   | -   | -   | -   | -   | -   | -   |
| state 55 |     |     |     |     |
| 253 | -   | L254 | -   | L255 | -   | L256 | -   |
| 254 | -   | -   | L253 | -   | -   | -   | -   |
| 255 | -   | -   | -   | -   | L257 | -   |
| 256 | -   | -   | -   | L253 | -   | -   |
| state 56 |     |     |     |     |
| 257 | L258 | -   | L259 | -   | L260 | -   | L261 | -   |
| 258 | L257 | -   | L257 | -   | L257 | -   | -   | -   |
| 259 | L257 | -   | L257 | -   | R260 | -   | R238 | -   |
| 260 | L257 | -   | R238 | -   | L257 | -   | R262 | -   |
| 261 | L257 | -   | L257 | -   | R260 | -   | L257 | -   |
| 262 | -   | -   | -   | R238 | -   | -   | -   |
| state 57 |     |     |     |     |
| 263 | R264 | -   | R265 | -   | R266 | -   | R267 | -   |
| 264 | R263 | -   | R263 | -   | R263 | -   | R263 | -   |
| 265 | R263 | -   | R263 | -   | -   | -   | R263 | -   |
| 266 | R263 | -   | -   | -   | -   | -   | R263 | -   |
| 267 | -   | R268 | -   | -   | -   | -   | R268 | -   |
| state 58 |     |     |     |     |
| 268 | -   | R269 | -   | R270 | -   | -   | -   |
| 269 | UL  | -   | GL271 | -   | -   | L   | -   |
| 270 | UL  | -   | R268 | -   | L271 | -   | R268 | -   |
| state 59 |     |     |     |     |
| 271 | L272 | -   | L273 | -   | L274 | -   | L275 | -   |
| 272 | -   | -   | L271 | -   | -   | -   | -   |
| 273 | -   | -   | -   | L271 | -   | -   |
| 274 | -   | -   | -   | -   | L276 | -   |
| 275 | -   | -   | L271 | -   | L271 | -   |
| state 60 |     |     |     |     |
| 276 | L277 | -   | L278 | -   | L279 | -   | L280 | -   |
| 277 | L276 | -   | L276 | -   | L276 | -   | L276 | -   |
| 278 | L276 | -   | L276 | -   | L276 | -   | L276 | -   |
| 279 | L276 | -   | L276 | -   | L276 | -   | R281 | -   |
| 280 | L276 | -   | L276 | -   | L276 | -   | L276 | -   |
| 281 | -   | -   | R238 | -   | -   | -   | -   |

(continued).
| State | A   | C   | G   | U   |
|-------|-----|-----|-----|-----|
| 61    | L283| L284| L285| L286|
| 62    | R288| R289| R290| R291|
| 63    | L   | GR  | CR  | AL  |
| 64    | R292|     |     |     |
| 65    | L297| L298| L299| L300|
| 66    | R308| R309| R310| R311|
| 67    | R313| R314| R315| R316|
| 68    | L318| L319| L320| L321|
| 69    | R323| R324| R325| R326|
| 70    | R327| R328| R330| R331|

(continued)
| State | A   | C   | G   | U   |
|-------|-----|-----|-----|-----|
| 71    |     |     |     |     |
| 332   | L333|     |     |     |
| 333   | CR334|     |     |     |
| 334   | R335| R336| R337|     |
| 335   |     | GL338| R334|     |
| 336   |     | R334| AL335|     |
| 337   | R334|     |     | R334|
| 72    |     |     |     |     |
| 338   | L339|     | L340|     |
| 339   |     | L338|     |     |
| 340   |     | L341|     |     |
| 73    |     |     |     |     |
| 341   | L342| L343| L344| L345|
| 342   | UR334| GR348| CR343| GR334|
| 343   | CR334| L341| R346|     |
| 344   | L341|     | R343|     |
| 345   | AR342|     | L341|     |
| 74    |     |     |     |     |
| 346   | R347| R348| R349| R350|
| 347   | CR348| GR347|     |     |
| 348   | R346| AL347| UR350| R346|
| 349   | UL351| L348| AL352|     |
| 350   | UL350| R346|     |     |
| 351   |     | CL353|     |     |
| 352   |     |     | CR348|     |
| 75    |     |     |     |     |
| 353   | L354| L355| L356| L357|
| 354   |     | L353| L353|     |
| 355   |     | L353|     |     |
| 356   |     |     | L353| L353|
| 357   |     | L353|     | L358|
| 76    |     |     |     |     |
| 358   | L359| L360| L361| L362|
| 359   | L358| L358| L358| AR363|
| 360   | L358| L358|     | L358|
| 361   | L358|     | L358| L358|
| 362   | L358| L358|     | R359|
| 77    |     |     |     |     |
| 363   | R364| R365| R366| R367|
| 364   | R363| R363| R363| R363|
| 365   | R363| R363|     | R363|
| 366   | R363|     |     | AR367|
| 367   | R368|     | AL366|     |
| 78    |     |     |     |     |
| 368   | R369| R370| R371| R372|
| 369   | R368| R368| R368| R368|
| 370   | R368| R368| AR371| R368|
| 371   | R368| R133| CL370| AR373|
| 372   |     | GL371| R368|     |
| 373   |     |     | R133|     |
| 79    |     |     |     |     |
| 374   | R375| R376| R377|     |
| 375   | CL378| R374| R374| R374|
| 376   | R374| R374|     | R374|
| 377   | R374|     |     |     |
| 378   | UL393|     |     |     |

(continued)
|    |     |     |     |     |     |
|----|-----|-----|-----|-----|-----|
| A  | C   | G   | U   |     |     |
| 379| R380| R381| R382|     |     |
| 380| R379|     |     |     |     |
| 381| R379|     |     |     |     |
| 382| R383|     |     |     |     |
| 383| R384| R385| R386| R387|     |
| 384| R383| R383| R383| R383|     |
| 385| R383| R383|     |     |     |
| 386| R383|     |     |     |     |
| 387| R388| AL386|     |     |     |
| 388| R389| R390| R391| R392|     |
| 389| CL392| R388| R388| R388|     |
| 390| R388| R388|     |     |     |
| 391| R388|     |     |     |     |
| 392| UL393|     |     |     |     |
| 393| L394| L395| L396| L397|     |
| 394| L393| L393| L393| L393|     |
| 395| L393| L393| L393| L393|     |
| 396| L393| R   | R398| L393|     |
| 397| L393| L393|     |     |     |
| 398| R399| R400| R401|     |     |
| 399| R398|     | CR402|     |     |
| 400| L399| R398|     |     |     |
| 401| UR403| R398|     |     |     |
| 402| R379|     |     |     |     |
| 403| R404| R405| R406| R407|     |
| 404| R403| R403| R403| R403|     |
| 405| R403| R403|     |     |     |
| 406| R403|     |     |     |     |
| 407| R403| GR403| CL406| L408| L409|
| 408|    |     |     |     |     |
| 409| L410| L411| L412| L413|     |
| 410| L409| L409| L409|     |     |
| 411| L409| L409|     |     |     |
| 412| L409|     | R128| R   |     |
| 413| L409| L409|     |     |     |