Date: 2025-12-30

Turing Machines

Abstract

Developments are given here for the analysis technique for non-terminating Turing Machines (TM's) that I described earlier in [1] and [2]. The main new ideas are the introduction of IRR patterns i.e. constraints satisfied by large sets of IRRs (Irreducible Regular Rules) and the logical relationships between them as a result of the general method for deriving IRR's from others described in my earlier paper. These logical relationships will be referred to as IGR's (IRR Generating Rules). IGR's have been reduced to their minimal form in a way analogous to the way in which regular rules were reduced to IRR's by taking out symbol strings that played no essential role. In the case of IGR's these symbol strings (actually pairs) will be referred to as context pairs. A new version of my computer program extending the previous analysis is described and is freely available that generates these IGR's up to a given length of IRR's that they generate. The results show repetition of the left hand halves (Left IGR's or LIGR's) of IGR's associated with different right hand halves. Because the LIGR's can be derived independently of the right hand halves of IGR's, this should be done separately and can be done using the currently known IRR's as previously described in my earlier papers. The LIGR's can be used to calculate all the IRR's of a TM. A procedure for the generation of all the LIGR's for a TM has been suggested and is expressed here by a detailed analysis of a TM though not yet as computer code.

Mathematics Subject Classification: 68Q25

Keywords: Turing Machine (TM), Irreducible Regular Rule (IRR), IRR Generating Rule (IGR), LIGR (Left IGR).

1 Introduction

A lot of material that probably will not be needed has been removed. It will be available as an old version to show where the ideas came from, and this much shortened version will take its place as the latest version under active editing. The following example was studied because old attempts at the analysis of (3) became very complicated.

$$\begin{array}{l}
\mathtt{1\underline{a}} \rightarrow \mathtt{2b\_} \\
\mathtt{1\underline{b}} \rightarrow \mathtt{3\_b} \\
\mathtt{1\underline{c}} \rightarrow \mathtt{1b\_} \\
\mathtt{2\underline{a}} \rightarrow \mathtt{3b\_} \\
\mathtt{2\underline{b}} \rightarrow \mathtt{2c\_} \\
\mathtt{2\underline{c}} \rightarrow \mathtt{1\_c} \\
\mathtt{3\underline{a}} \rightarrow \mathtt{1\_a} \\
\mathtt{3\underline{b}} \rightarrow \mathtt{1\_a} \\
\mathtt{3\underline{c}} \rightarrow \mathtt{3c\_}
\end{array} \tag{1}$$

The main results are contained in (9), (10) and (11) which were updated together with some text on page 59. Table 1 is summarised by Figure 2 and shows how the TM can be 'trapped' in a steady movement to its left. The results (9), (10), (11) seem to be adequately describing the TM.

section **??** needs some discussion because the two examples are so different.

Some general heuristics are given right at the end regarding using the IGR's to generate a description of the action of a TM in terms of ever expanding cycles (when possible). The TM (2) is an example of this while TM (1) is not.

A lot of material has been removed to 2017's Notes on Turing Machines. These notes are now mostly superseded, but there may be a little there that is of use.

Comments are welcome. Please send them to john.h.nixon1@gmail.com

Consider another example given by

$$\begin{array}{ll}
\mathtt{1\underline{a}} \rightarrow \mathtt{2b\_} & \mathtt{1\underline{b}} \rightarrow \mathtt{2\_a} \\
\mathtt{2\underline{a}} \rightarrow \mathtt{1b\_} & \mathtt{2\underline{b}} \rightarrow \mathtt{2\_a}\ . \\
\mathtt{3\underline{a}} \rightarrow \mathtt{3a\_} & \mathtt{3\underline{b}} \rightarrow \mathtt{1\_b}
\end{array} \tag{2}$$

The analysis techniques were initially applied to the following TM (3) which was generated randomly with 5 states and 5 symbols. This TM, being much larger than any that I have analysed before, has proved to be a much more challenging case.

$$\begin{array}{lllll}
\mathtt{1\underline{a}} \rightarrow \mathtt{2\_d} & \mathtt{2\underline{a}} \rightarrow \mathtt{1c\_} & \mathtt{3\underline{a}} \rightarrow \mathtt{4c\_} & \mathtt{4\underline{a}} \rightarrow \mathtt{3\_b} & \mathtt{5\underline{a}} \rightarrow \mathtt{2\_e} \\
\mathtt{1\underline{b}} \rightarrow \mathtt{4\_d} & \mathtt{2\underline{b}} \rightarrow \mathtt{4\_c} & \mathtt{3\underline{b}} \rightarrow \mathtt{4\_c} & \mathtt{4\underline{b}} \rightarrow \mathtt{4b\_} & \mathtt{5\underline{b}} \rightarrow \mathtt{3\_e} \\
\mathtt{1\underline{c}} \rightarrow \mathtt{3\_a} & \mathtt{2\underline{c}} \rightarrow \mathtt{1d\_} & \mathtt{3\underline{c}} \rightarrow \mathtt{2\_a} & \mathtt{4\underline{c}} \rightarrow \mathtt{3c\_} & \mathtt{5\underline{c}} \rightarrow \mathtt{3a\_} \\
\mathtt{1\underline{d}} \rightarrow \mathtt{2b\_} & \mathtt{2\underline{d}} \rightarrow \mathtt{1a\_} & \mathtt{3\underline{d}} \rightarrow \mathtt{5\_c} & \mathtt{4\underline{d}} \rightarrow \mathtt{5\_c} & \mathtt{5\underline{d}} \rightarrow \mathtt{4\_a} \\
\mathtt{1\underline{e}} \rightarrow \mathtt{2b\_} & \mathtt{2\underline{e}} \rightarrow \mathtt{3\_c} & \mathtt{3\underline{e}} \rightarrow \mathtt{3b\_} & \mathtt{4\underline{e}} \rightarrow \mathtt{5a\_} & \mathtt{5\underline{e}} \rightarrow \mathtt{3a\_}
\end{array} \tag{3}$$

I have just started (but not finished) the same analysis of the old example (3) to see how easy the method will be will be for larger TM's.

$$\texttt{1}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2\_d} \begin{cases} \xrightarrow{\texttt{a}} \texttt{2\_aa(2}\underline{\texttt{a}}\texttt{d} \rightarrow \texttt{2\_aa)} \\ \xrightarrow{\texttt{b}} \texttt{4\_cd} \\ \xrightarrow{\texttt{c}} \texttt{2db\_(2}\underline{\texttt{c}}\texttt{d} \rightarrow \texttt{2db\_)} \\ \xrightarrow{\texttt{d}} \texttt{2ab\_(2}\underline{\texttt{d}}\texttt{d} \rightarrow \texttt{2ab\_)} \\ \xrightarrow{\texttt{e}} \texttt{3\_cd} \end{cases} \\ \xrightarrow{\texttt{b}} \texttt{4\_d} \\ \xrightarrow{\texttt{c}} \texttt{3\_a} \\ \xrightarrow{\texttt{d}} \texttt{2b\_} \\ \xrightarrow{\texttt{e}} \texttt{2b\_} \end{cases} \tag{4}$$

$$\texttt{2}\begin{cases} \xrightarrow{\texttt{a}} \texttt{1c\_} \\ \xrightarrow{\texttt{b}} \texttt{4\_c} \\ \xrightarrow{\texttt{c}} \texttt{1d\_} \\ \xrightarrow{\texttt{d}} \texttt{1a\_} \\ \xrightarrow{\texttt{e}} \texttt{3\_c} \end{cases} \tag{5}$$

$$\texttt{3}\begin{cases} \xrightarrow{\texttt{a}} \texttt{4c\_} \\ \xrightarrow{\texttt{b}} \texttt{4\_c} \\ \xrightarrow{\texttt{c}} \texttt{2\_a} \\ \xrightarrow{\texttt{d}} \texttt{5\_c} \\ \xrightarrow{\texttt{e}} \texttt{3b\_(3}\underline{\texttt{e}} \rightarrow \texttt{3b\_)} \end{cases} \tag{6}$$

$$\texttt{4}\begin{cases} \xrightarrow{\texttt{a}} \texttt{3\_b} \\ \xrightarrow{\texttt{b}} \texttt{4b\_(4}\underline{\texttt{b}} \rightarrow \texttt{4b\_)} \\ \xrightarrow{\texttt{c}} \texttt{3c\_} \\ \xrightarrow{\texttt{d}} \texttt{5\_c} \\ \xrightarrow{\texttt{e}} \texttt{5a\_} \end{cases} \tag{7}$$

$$\texttt{5}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2\_e} \\ \xrightarrow{\texttt{b}} \texttt{3\_e} \\ \xrightarrow{\texttt{c}} \texttt{3a\_} \\ \xrightarrow{\texttt{d}} \texttt{4\_a} \\ \xrightarrow{\texttt{e}} \texttt{3a\_} \end{cases} \tag{8}$$

A branch can end because of three conditions, (1) a repetition or loop (indicating that there is no point in continuing) because a substring in the same branch has been developed before and (2) a reference to a loop. This is indicated by the loop label (a greek letter) and an asterisk and (3) when the computation from the same CS appears on another branch. If the computation ends in a subset of a CS previously developed, the extra symbol(s) need to be added resulting in another subtree. A very simple case follows. To get the results for 3b|b_ from those for 3|b_ (ϵ an extra b must be put on the left. The first branch gives rise to the loop δ going left and the added b gives $\mathtt{3\underline{b}ba} \rightarrow$ 1_aba and in its new location it gives rise to a new loop $\mathtt{1\underline{a}ad} \rightarrow$ 1_aba. The branch to 3b|c_ going to the loop θ going to the right, gives this same loop after the b is added on the left because the extra symbol is added on the left. If the new symbol is added on the opposite side to the direction of travel in a cycle of the loop the same loop with be obtained after the symbol is added, but not otherwise.

How to handle reversals of direction add the extra symbol to extend the tree as above or start a new one? a reversal of direction so that the pointer is on the other side of the string. In this case the computation continues on another tree because every possible state and symbol pair (accounting for all possible cases) is at the root of a tree

Another example is how the development of 2|bb_ is obtained from that of 2|b_. Putting a b on the left gives the first result $\mathtt{1\underline{b}aba} \rightarrow$ 3_baba. This results in two loops because both starting points $\mathtt{3\underline{c}bad}$ and $\mathtt{3bbb\underline{d}}$ match the endpoint 3_baba. This is indicated by $\mathtt{3\underline{c}bad} \rightarrow \mathtt{3bbb\underline{d}} \rightarrow$ 3_baba. The other cases are very easy. Note that the | has no meaning unless each symbol is added one at a time so they are omitted if this does not happen.

In case (1) in what follows the numbers in typewriter font represent different repeating conditions. The number is the length l of the string over which a repetition can occur. This includes the symbol added so it is always ≥ 1 and if $\mathtt{l} = \mathtt{1}$ there is no symbol string to be matched. The repeating condition is where the state, pointer position (right or left) and the symbol string match between a CS and another CS that is in the path to it from the root of the tree. Between these two CS's the pointer moves in a range. A repetition also requires every symbol in the second CS in the range to match the corresponding string from the first CS.

In case (2) a branch ends because the computation (taken as far as possible) goes in the opposite direction. This is indicated by an italicised identification number. These numbers are also repeated where a matching CS appears in another tree from where the computation can continue.

$$\mathtt{1\_}\begin{cases} \xrightarrow{\mathtt{a}} \mathtt{2|b\_} \begin{cases} \xrightarrow{\mathtt{a}} \mathtt{3b|b\_}(\epsilon)\begin{cases}\xrightarrow{\mathtt{d}} \mathtt{1\_aba}(\mathtt{1\underline{a}ad}\rightarrow\mathtt{1\_aba})\\ \xrightarrow{\mathtt{c}}\mathtt{3bb|c\_}(\theta^*)\end{cases}\\ \xrightarrow{\mathtt{b}}\mathtt{2b|c\_}(\zeta^*)\\ \xrightarrow{\mathtt{c}}\mathtt{3\_bc|}(\beta^*)\end{cases}\\ \xrightarrow{\mathtt{b}}\mathtt{3\_b|}\begin{cases}\xrightarrow{\mathtt{d}}\mathtt{1\_a|b}(\mathtt{1d\underline{b}}\rightarrow\mathtt{1\_ab})\\ \xrightarrow{\mathtt{c}}\mathtt{2|bb\_}(\alpha)\begin{cases}\xrightarrow{\mathtt{ad}}\mathtt{3\_b|aba}(\mathtt{3\underline{c}bad}\rightarrow\mathtt{3bbb\underline{d}}\rightarrow\mathtt{3\_baba})\\ \xrightarrow{\mathtt{ac}}\mathtt{3bbb|c\_}(\theta^*)\\ \xrightarrow{\mathtt{b}}\mathtt{2bb|c\_}(\zeta^*)\\ \xrightarrow{\mathtt{c}}\mathtt{1\_abc|}(\eta^*,\text{ignore x})\end{cases}\end{cases}\\ \xrightarrow{\mathtt{c}}\mathtt{1|b\_}(\mathtt{1\underline{c}}\rightarrow\mathtt{1b\_},\gamma)\end{cases}\tag{9}$$

$$\mathtt{2\_}\begin{cases} \xrightarrow{\mathtt{a}}\mathtt{3|b\_}(\epsilon)\begin{cases}\xrightarrow{\mathtt{d}}\mathtt{3\_ba|}(\delta^*)\\ \xrightarrow{\mathtt{c}}\mathtt{3b|c\_}(\theta^*)\end{cases}\\ \xrightarrow{\mathtt{b}}\mathtt{2|c\_}(\mathtt{2\underline{b}}\rightarrow\mathtt{2c\_},\zeta)\\ \xrightarrow{\mathtt{c}}\mathtt{1\_c|}\begin{cases}\xrightarrow{\mathtt{d}}\mathtt{3\_bc|}(\beta)\overset{\mathtt{x=a,b,c}}{\rightarrow}\mathtt{1\_abc}(\eta)\begin{cases}\xrightarrow{\mathtt{a}}\mathtt{1\_aba|c}(\mathtt{1ab\underline{b}}\rightarrow\mathtt{1\underline{a}ab}\rightarrow\mathtt{1b\underline{b}a}\rightarrow\mathtt{1\_aba})\\ \xrightarrow{\mathtt{b}}\mathtt{3\_b|abc}\begin{cases}(\mathtt{x}=\mathtt{d},\mathtt{3b\underline{d}}\rightarrow\mathtt{3\_ba})\\ \overset{\mathtt{d,x=c}}{\rightarrow}\mathtt{1\_a|babc}(\mathtt{1d\underline{b}}\rightarrow\mathtt{1\_ab})\\ \overset{\mathtt{c,x=c}}{\rightarrow}\mathtt{3\_baba|c}(\mathtt{3\underline{c}bab}\rightarrow\mathtt{3\_baba})\end{cases}\\ \xrightarrow{\mathtt{c}}\mathtt{1b}^4\mathtt{\_}(\gamma^*)\end{cases}\\ \xrightarrow{\mathtt{c}}\mathtt{1bb\_}(\gamma^*)\end{cases}\end{cases}\tag{10}$$

$$\mathtt{3\_}\begin{cases}\xrightarrow{\mathtt{d}}\mathtt{1\_a|}\begin{cases}\xrightarrow{\mathtt{a}}\mathtt{3|bb\_}(\epsilon^*)\\ \xrightarrow{\mathtt{b}}\mathtt{3\_b|a}(\mathtt{3b\underline{d}}\rightarrow\mathtt{3\_ba},\delta)\\ \xrightarrow{\mathtt{c}}\mathtt{2|bb\_}(\alpha^*)\end{cases}\\ \xrightarrow{\mathtt{c}}\mathtt{3|c\_}(\mathtt{3\underline{c}}\rightarrow\mathtt{3c\_},\theta)\end{cases}\tag{11}$$

Do the results (9),(10) and (11) adequately characterise TM (1)?

By taking the longest results of the repeating cycles on the RHS's of (9),(10) and (11) i.e. $\mathtt{1\_aba},\mathtt{3\_baba}$ and adding every symbol in turn gives $\mathtt{1\underline{a}aba}\rightarrow\mathtt{1\_abaa}$, $\mathtt{1\underline{a}abaa}\rightarrow\mathtt{C}$, $\mathtt{1\underline{b}abaa}\rightarrow\mathtt{E}$, $\mathtt{1\underline{c}abaa}\rightarrow\mathtt{3b}^5\mathtt{\_}$, $\mathtt{1\underline{b}aba}\rightarrow\mathtt{3\_baba}$, $\mathtt{3\underline{a}baba}\rightarrow\mathtt{A}$, $\mathtt{3\underline{b}baba}\rightarrow\mathtt{A}$, $\mathtt{3\underline{c}baba}\rightarrow\mathtt{E}$, $\mathtt{1\underline{c}aba}\rightarrow\mathtt{2bbbb\_}$ where I am using the capital letter notation for pseudo-states in Table 1. Another round of this will generate all the results of Table 1.

2 Formulating the condition for a repetition

In these trees if on any branch, the final CS matches an earlier CS in such a way that the loop can be repeated (this might work after some symbols not involved in the loop itself are ignored) then the algorithm terminates the branch because continuing is a special case of what has already been done. If this happens, all the CS's that match the final CS should be listed in order in parentheses, so that many other results of the TM can be found easily. Suppose

$$\mathtt{CS}_1 \overset{\alpha_1}{\to} \mathtt{CS}_2 \overset{\alpha_2}{\to} \dots \tag{12}$$

is such a branch that ends in a repeating loop and the pointer is at the right in each CS where $\mathtt{CS}_i$ has length i and in step i from $\mathtt{CS}_i$ to $\mathtt{CS}_{i+1}$ the pointer reaches and uses $r(i)$ symbols (this excludes the last symbol arrived at that is not yet read). $\mathtt{CS}_1$ is a CS of length 1 with the pointer on the right and is just one TM step away from the root which is a state and symbol pair. After l_2-1 steps in (12) giving a CS of length $i=l_2$ what is the condition for a repetition of an earlier CS of length $i=l_1$? This involves l_2-l_1 steps in (12).

Because the symbols are added on the right, the tape positions will be counted going to the right and the leftmost position is position 1 in all the CS's. The pointer starts at l_1+1 and first goes to l_1+2 via $l_1-r(l_1)+2$ (the symbol positions used go from $l_1-r(l_1)+2$ to l_1+1 i.e. a segment of length $r(l_1)$). The complete potentially repeating computation reaches the following extreme pointer positions in this order $l_1+1, l_1+2-r(l_1), l_1+2, l_1+3-r(l_1+1), \dots l_2$ because in the final step to get $\mathtt{CS}_{l_2}$ the pointer does not go beyond l_2. The range of the tape affected by the computation is from position p to l_2 inclusive (see Figure 1) where

$$p = \min_{l_1 \le i \le l_2-1} \{i+2-r(i)\}\,. \tag{13}$$

The repeating condition implies that the states match between the start and end of the computation and there is a pair of matching substrings of m symbols in the two CS's such that each substring lies within the range p to l_2 and must include all the symbols in that range on the left hand end otherwise the computation could not be repeated due to a mismatch. Therefore $p=l_1-m+1$ is the leftmost symbol position involved in the matching i.e. $m=l_1-p+1$. The length of the potentially repeating rule is the length of tape involved in (13) i.e. $t=l_2-p+1$. Therefore $t-m=l_2-l_1\ge 1$. One of the shortest possible examples is $\mathtt{3\underline{c}} \to \mathtt{3c\_}$. If there are no other symbols on the left, $l_1=0$ and $l_2=1$ and $p=1$ therefore $m=0$ and $t=1$ so in general $t>m\ge 0$.

The notation | was introduced in the CS's to indicate the limit beyond which the pointer did not go to obtain the CS from the preceding one. This is a visual indication of $r(i)$ which is the number of symbols between | and the

$i+2-r(i)$	$r(i)$	i	
4	2	4	`x x x x _`
3	4	5	`x x x | x x _`
5	3	6	`x x | x x x x _`
8	1	7	`x x x x | x x x _`
minimum: $p=3$		8	`x x x x x x x | x _`
			`    ↑`

Figure 1: **A schematic example of a repetition (states omitted)**. Here $l_1 = 4$,$l_2 = 8$, and $p = 3$ therefore $m = 2$ and $t = 6$ and the repeating rule has the form $\widehat{\texttt{xx}}\underline{\texttt{x}}\texttt{xxx} \rightarrow \texttt{xxxx}\widehat{\texttt{xx}}\_$ where the `x`'s represent any symbols, and the _'s are where the symbols are added at the pointer position. The strings of symbols under the widehat $\widehat{}$ must be the same. These are the m symbols that are repeated. The $\uparrow$ is where $p = 3$ giving a visual indication of the end of range of the symbols that are involved in the repeating computation rule.

end of the string where the symbol _ is, where i is the length of the *preceding* CS.

Table 1: A finite state machine going left derived from TM 1

A: `1_ababa`	B: `1_abaab`	C: `1_abaaa`	D: `3_babab`	E: `3_babaa`
`a` → B	`a` → C	`a` → C	`a` → A	`a` → A
`b` → D	`b` → E	`b` → E	`b` → A	`b` → A
`c` → D	`c` → D	`c` → D	`c` → D	`c` → E

From this it appears that there are no IRR's of length ≥ 7 of the type RLR, then all such IRR's have type RLL, LRL, or LRR. If this is generally true, and if the TM reaches position 6 followed by position 1 it cannot subsequently reach position ≥ 7 because this would require a subsequence of CS's of the form $\texttt{n} \rightarrow \texttt{1} \rightarrow \texttt{n}+\texttt{1}$ with $\texttt{n} = 6$ which would be an IRR by lemma **??** of type RLR of length $n+1$ contradicting the assumption. The pointer is then constrained to positions ≤ 6, and if it reaches position 0 then because it has reached position 5 previously, the same argument can be applied showing that it cannot then reach position ≥ 6 etc.. This implies that the pointer is constrained to being in a moving window of length 6 that moves left by one space when the pointer moves just to its left. Because of this, if a snapshot is taken of its behaviour whenever the TM reaches just beyond the left hand end of the window, whatever symbol it finds there, the result will be at the next snapshot that the symbols of the window have changed depending on the previous symbols there and the new symbol. Therefore if the TM reaches position 6 followed by position 1 then the above argument involving the moving

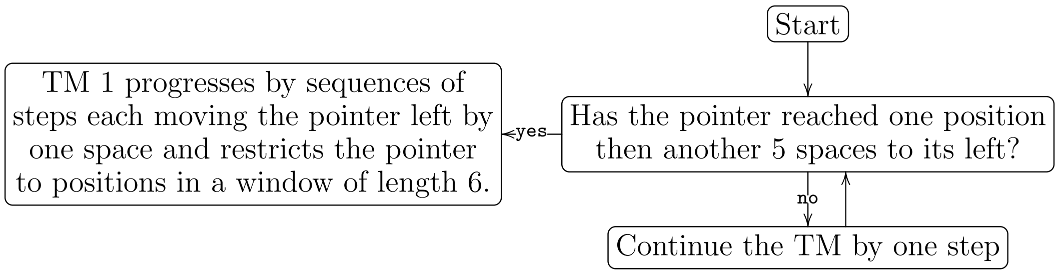
Start
Has the pointer reached one position then another 5 spaces to its left?
yes
TM 1 progresses by sequences of steps each moving the pointer left by one space and restricts the pointer to positions in a window of length 6.
no
Continue the TM by one step

Figure 2: Summary of the results of the analysis of TM(1)

window applies. This condition of course will happen depending on the initial contents of the tape of the TM that could start the TM doing one of the iterations going right mentioned above.

Now it is obvious that this effective finite state machine is defined by all IRR's of the type LRL of length $\leq n$ for some length n. This behaviour going left corresponds to the sequence $\mathtt{A} = \mathtt{p} \rightarrow \mathtt{2} \rightarrow \mathtt{p} - \mathtt{1} \rightarrow \mathtt{1} \rightarrow \mathtt{p} - \mathtt{2} \rightarrow \mathtt{0}$ etc. for some positive integer $\mathtt{p}$ and requires $\mathtt{B}$ (a subsequence of $\mathtt{A}$) i.e. $\mathtt{2} \rightarrow \mathtt{p} - \mathtt{1} \rightarrow \mathtt{1}$ to exist which is an IRR of type LRL of length $\mathtt{p}$. Also the sequence $\mathtt{p} - \mathtt{2} \rightarrow \mathtt{2} \rightarrow \mathtt{p} - \mathtt{1}$ would have to exist which is an IRR of type RLR of length $(\mathtt{p} - \mathtt{1}) - \mathtt{2} + \mathtt{1} = \mathtt{p} - \mathtt{2}$ so for this TM $\mathtt{p}$ could not be larger than 8 and the longest IRR of type RLR needed could not have length greater than 6. This is an effective finite state machine with internal state corresponding to the set of symbols in the window and its actual machine state, and it continues indefinitely unless a stationary cycle occurs which would halt it.

In this example the sequence $\mathtt{6} \rightarrow \mathtt{1} \rightarrow \mathtt{7}$ is impossible but $\mathtt{6} \rightarrow \mathtt{1} \rightarrow \mathtt{0}$ and $\mathtt{6} \rightarrow \mathtt{1} \rightarrow \mathtt{5} \rightarrow \mathtt{0}$ are not ruled out.

References

[1] Methods for Understanding Turing Machine Computations

[2] Reverse engineering Turing Machines and the Collatz Conjecture

[3] The previous version in D of the computer program for analysis of Turing Machines

[4] Program for just doing the backward search for a single CS

[5] The new program tie v3.2 for doing the computations in this paper