Date: 2026-01-13

Turing Machines

Abstract

Mathematics Subject Classification: 68Q25

Keywords: Turing Machine (TM).

1 Introduction

A development is given here for the analysis techniques for non-terminating Turing Machines (TM's) that I described earlier in [1] and [2]. This technique was motivated by my earlier work but can be largely described without it resulting in a very short paper where this prior material has been removed. This consists of a simple procedure I call the development algorithm because it seems to be a very natural and general outworking of the rules of a Turing Machine (TM). The material removed and is available as an old version ([5],[6]) to show where the ideas came from, and this much shortened version of the paper has taken its place as the latest version under active editing. Comments are welcome. Please send them to john.h.nixon1@gmail.com

A (non-terminating) Turing Machine (TM) consists of a read-write head that moves by one square at a time on a one-dimensional tape that is infinite in both directions marked off into squares on each of which which one of a set of symbols is written. Any symbol on the tape remains there until it is overwritten by the TM. The action of the TM is as follows. First the current symbol is read, and depending on this and the machine's internal state, three actions are performed in a cycle. (1) The new symbol is printed at the current square, (2) the TM enters a new state, and (3) the TM moves right or left by one square. These cycles (steps) are repeated indefinitely. In order to describe the action of a TM, configuration sets (CS's) were introduced. A CS is a set of complete configurations (all the tape symbols with pointer position indicated, and the machine state of the TM) such that the CS is specified by giving a finite set of symbols in a set of contiguous pointer positions together with the machine state and such that the pointer position is where one of the given symbols is given or adjacent to one. In a CS all possible configurations that are consistent with the specified symbols and machine state are included. The notation is the specified symbol string with the pointer indicated by an underscore (if it is just off the end of the symbol string) or an underline and the machine state is on the left. For example with machine states 1,2,3, etc. and symbols as lower case letters the following are CSs: $\texttt{2a\underline{b}ca}$, $\texttt{1\_aabbcac}$. The length of the CS is the length of the symbol string which is finite.

A computation rule or rule is a pair of CS's linked by $\to$ indicating the forward direction of the computation. It indicates that one or more TM steps will take any configuration in the CS on the left of $\to$ to a configuration in the CS on the right. In this paper rules will not include symbols on the tape that are not read by the TM. A very convenient way of a specifying a TM is to use a set rules involving CS's of length 1.

2 A very simple example

The following example of a Turing Machine (TM) and its analysis illustrate all the ideas of the paper. The TM is given by

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

and the analysis developed from it is as follows.

$$\begin{array}{l}
\texttt{1}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2b\_}\begin{cases} \xrightarrow{\texttt{a}} \texttt{1bb\_}(\texttt{1\underline{a}a} \to \texttt{1bb\_}) \\ \xrightarrow{\texttt{b}} \texttt{2\_aa},(\alpha^*) \end{cases} \\ \xrightarrow{\texttt{b}} \texttt{2\_a}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2bb\_}(\texttt{2\underline{a}a} \to \texttt{2bb\_}) \\ \xrightarrow{\texttt{b}} \texttt{2\_aa}(\alpha^*) \end{cases} \end{cases} \\
\texttt{2}\begin{cases} \xrightarrow{\texttt{a}} \texttt{1b\_}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2bb\_}(\texttt{2\underline{a}a} \to \texttt{2bb\_}) \\ \xrightarrow{\texttt{b}} \texttt{2\_aa}(\alpha^*) \end{cases} \\ \xrightarrow{\texttt{b}} \texttt{2\_a}(\texttt{2\underline{b}} \to \texttt{2\_a},\alpha) \end{cases} \\
\texttt{3}\begin{cases} \xrightarrow{\texttt{a}} \texttt{3a\_}(\texttt{3\underline{a}} \to \texttt{3a\_}) \\ \xrightarrow{\texttt{b}} \texttt{1\_b}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2\_aa}(\alpha^*) \\ \xrightarrow{\texttt{b}} \texttt{2\_ab}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2\_aaa}\begin{cases} \xrightarrow{\texttt{a}} \texttt{2bbbb\_} \\ \xrightarrow{\texttt{b}} \texttt{2\_aaaa} \end{cases} \\ \xrightarrow{\texttt{b}} \texttt{2\_aab}(\alpha^*) \end{cases} \end{cases} \end{cases}
\end{array} \tag{2}$$

This can be continued using the rules in Table 1 showing how the length of the CS's can increase indefinitely and the cycles increase in size without limit. Here the pseudo-sate D arises separately from A,B and C.

Table 1: Computation rules derived from TM 1

A: 2_a^n	B: 1b^n_	C: 2b^n_	D: 3a^n_
a $\to$ B (n even)	a $\to$ C	a $\to$ B	a $\to$ D
a $\to$ C (n odd)	b $\to$ A	b $\to$ A	b $\to$ B*.
b $\to$ A			

Here each column corresponds to the CS at its head labelled from A to D. The meaning of the entries in the body of the table is that if the symbol on the left is at the pointer in the CS (extending the string by one symbol), the result of the following computation is the CS indicated on the right with n replaced by n+1. The result at * is actually 1a^{n-2}bbb_ if $n > 2$ so that the new n is now 3. The other cases are dealt with by the rules $\texttt{3}\underline{\texttt{a}}\texttt{b} \to \texttt{3a}\underline{\texttt{b}} \to$ 2_aa, $\texttt{3b}\underline{\texttt{b}} \to$ 2_ab, and $\texttt{3}\underline{\texttt{a}}\texttt{ab} \to \texttt{3aa}\underline{\texttt{b}} \to$ 1bbb_. Notice that when the pointer swaps ends going right, only b's are produced, and if it goes left only a's are produced. Also the TM cannot reach a configuration in D by steps in Table 1 unless it started from a configuration in D. From there as long as the new symbol is a, the TM continues to the right. If a b is reached, it goes to a configuration in B, from where successive a's alternate between CS's B and C. If a b is reached in either B or C then it swaps sides leaving a's and getting to A. Then more b's just leave it in A, but an a makes it swap sides to B or C according to whether n is even or odd respectively. Thus the description of TM (5) is clearly explained as an expanding cycle, which to me seems a satisfactory place to leave the general analysis of this example.

3 The development algorithm

The algorithm is very simple and is as follows. Starting from any CS, every possible symbol is added at the pointer position leading to a set of branches and in each case the computation continues as far as possible until a symbol in a new position needs to be read (a final CS). The resulting CS forms the next starting point. Whenever a new final CS is reached, if a repeating condition has occurred with other CS's (*) on the path to it from the root (an initial CS only specifying the machine state), i.e. 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. Then 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 symbol from the first CS. If this happens the repeating cycle is identified and written within parentheses and optionally labelled by a greek letter. This terminates this branch of the development because any further development on the branch is a special case of analysis that has already been done. This assumes all branches from CS

* have already been developed. Likewise if a CS is reached that matches a repeating cycle already found elsewhere, the label for that cycle is given and an asterisk indicating where to refer to for its continuation. The algorithm starts from every CS (root) that consists of just the TM state i.e. the string of symbols has length 0.

4 Formulating the condition for a repetition

In these trees (i.e. results of the development algorithm e.g (6), (7) and (8) for TM (5) or (2) for TM (1)), if on any branch, the final CS matches an earlier CS in such a way that the loop can be repeated (this might only work after some symbols not involved in the loop are ignored in which case the repeating loop needs to be first applied to the simpler case where these extra symbols are absent) 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 in the loop should be listed in order in parentheses, so that many other results of the TM can be found easily.

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

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 $\widehat{\mathtt{xx}}\underline{\mathtt{x}}\mathtt{xxx} \to \mathtt{xxxx}\widehat{\mathtt{xx}}\_$ 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.

Suppose

$$\mathtt{CS}_0 \overset{\alpha_1}{\to} \mathtt{CS}_1 \overset{\alpha_2}{\to} \dots \tag{3}$$

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 $i+1$ 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). $\mathtt{CS}_0$ is just a CS of length 0 which only defines the machine state and is the root of a tree of CS's developed from it by the development algorithm. After l_2 steps in (3) giving a CS of length $i = l_2$ the condition for a repetition of an earlier CS of length $i = l_1$ is as follows. This involves $l_2 - l_1$ steps in (3).

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 $l_1 + 1$ to read the next symbol and first goes to $l_1 + 2$ via $l_1 - r(l_1 + 1) + 2$ (i.e. a segment of length $r(l_1 + 1)$). The complete potentially repeating computation reaches the following extreme pointer positions in this order $l_1 + 1, l_1 + 2 - r(l_1 + 1), l_1 + 2, l_1 + 3 - r(l_1 + 2), \ldots l_2 + 1 - r(l_2), l_2$ 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

$$p = \min_{l_1 + 1 \leq i \leq l_2} \{i + 1 - r(i)\} \,. \tag{4}$$

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 $\mathtt{CS}_{l_1}$ and $\mathtt{CS}_{l_2}$ 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 (4) i.e. $t = l_2 - p + 1$. Therefore $t - m = l_2 - l_1 \geq 1$. One of the shortest possible examples is $\mathtt{3\underline{c}} \rightarrow \mathtt{3c\_}$ in which, $l_1 = 0$ and $l_2 = 1$ and $p = 1$ therefore $m = 0$ and $t = 1$ so in general $t > m \geq 0$.

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

5 A more substantial example

The following example was also studied.

$$\begin{array}{l}
\mathtt{1\underline{a}} \to \mathtt{2b\_} \\
\mathtt{1\underline{b}} \to \mathtt{3\_b} \\
\mathtt{1\underline{c}} \to \mathtt{1b\_} \\
\mathtt{2\underline{a}} \to \mathtt{3b\_} \\
\mathtt{2\underline{b}} \to \mathtt{2c\_} \\
\mathtt{2\underline{c}} \to \mathtt{1\_c} \\
\mathtt{3\underline{a}} \to \mathtt{1\_a} \\
\mathtt{3\underline{b}} \to \mathtt{1\_a} \\
\mathtt{3\underline{c}} \to \mathtt{3c\_}
\end{array} \tag{5}$$

By applying the development algorithm to TM 5 the results in (6), (7) and (8) were obtained which seem to adequately describe the TM and lead to Table 2 which is summarised by Figure 2 and shows how the TM can be 'trapped' in a steady movement to its left.

$$\mathtt{1\_}\begin{cases}
\xrightarrow{\mathtt{a}} \mathtt{2|b\_}\begin{cases}
\xrightarrow{\mathtt{a}} \mathtt{3b|b\_}(\epsilon)\begin{cases}
\xrightarrow{\mathtt{d}} \mathtt{1\_aba|}(\mathtt{1\underline{a}ad} \to \mathtt{1\_aba}) \\
\xrightarrow{\mathtt{c}} \mathtt{3bb|c\_}(\theta^*)
\end{cases} \\
\xrightarrow{\mathtt{b}} \mathtt{2b|c\_}(\zeta^*) \\
\xrightarrow{\mathtt{c}} \mathtt{3\_bc|}(\beta^*)
\end{cases} \\
\xrightarrow{\mathtt{b}} \mathtt{3\_b|}\begin{cases}
\xrightarrow{\mathtt{d}} \mathtt{1\_a|b}(\mathtt{1d\underline{b}} \to \mathtt{1\_ab}) \\
\xrightarrow{\mathtt{c}} \mathtt{2|bb\_}(\alpha)\begin{cases}
\xrightarrow{\mathtt{ad}} \mathtt{3\_b|aba}(\mathtt{3\underline{c}bad} \to \mathtt{3bbb\underline{d}} \to \mathtt{3\_baba}) \\
\xrightarrow{\mathtt{ac}} \mathtt{3bbb|c\_}(\theta^*) \\
\xrightarrow{\mathtt{b}} \mathtt{2bb|c\_}(\zeta^*) \\
\xrightarrow{\mathtt{c}} \mathtt{1\_abc|}(\eta^*, \text{ignore x})
\end{cases}
\end{cases} \\
\xrightarrow{\mathtt{c}} \mathtt{1|b\_}(\mathtt{1\underline{c}} \to \mathtt{1b\_}, \gamma)
\end{cases} \tag{6}$$

$$\mathtt{2\_}\begin{cases}
\xrightarrow{\mathtt{a}} \mathtt{3|b\_}(\epsilon)\begin{cases}
\xrightarrow{\mathtt{d}} \mathtt{3\_ba|}(\delta^*) \\
\xrightarrow{\mathtt{c}} \mathtt{3b|c\_}(\theta^*)
\end{cases} \\
\xrightarrow{\mathtt{b}} \mathtt{2|c\_}(\mathtt{2\underline{b}} \to \mathtt{2c\_}, \zeta) \\
\xrightarrow{\mathtt{c}} \mathtt{1\_c|}\begin{cases}
\xrightarrow{\mathtt{d}} \mathtt{3\_bc|}(\beta) \xrightarrow{\mathtt{x=a,b,c}} \mathtt{1\_abc}(\eta)\begin{cases}
\xrightarrow{\mathtt{a}} \mathtt{1\_aba|c}(\mathtt{1ab\underline{b}} \to \mathtt{1\underline{a}ab} \to \mathtt{1b\underline{b}a} \to \mathtt{1\_aba}) \\
\xrightarrow{\mathtt{b}} \mathtt{3\_b|abc}\begin{cases}
(\mathtt{x} = \mathtt{d}, \mathtt{3b\underline{d}} \to \mathtt{3\_ba}) \\
\xrightarrow{\mathtt{d,x=c}} \mathtt{1\_a|babc}(\mathtt{1d\underline{b}} \to \mathtt{1\_ab}) \\
\xrightarrow{\mathtt{c,x=c}} \mathtt{3\_baba|c}(\mathtt{3\underline{c}bab} \to \mathtt{3\_baba})
\end{cases} \\
\xrightarrow{\mathtt{c}} \mathtt{1b^4\_}(\gamma^*)
\end{cases} \\
\xrightarrow{\mathtt{c}} \mathtt{1bb\_}(\gamma^*)
\end{cases}
\end{cases} \tag{7}$$

$$\texttt{3\_}\begin{cases} \xrightarrow{\texttt{d}} \texttt{1\_a}| \begin{cases} \xrightarrow{\texttt{a}} \texttt{3}|\texttt{bb\_}(\epsilon^*) \\ \xrightarrow{\texttt{b}} \texttt{3\_b}|\texttt{a}(\texttt{3b\underline{d}} \to \texttt{3\_ba}, \delta) \\ \xrightarrow{\texttt{c}} \texttt{2}|\texttt{bb\_}(\alpha^*) \end{cases} \\ \xrightarrow{\texttt{c}} \texttt{3}|\texttt{c\_}(\texttt{3\underline{c}} \to \texttt{3c\_}, \theta) \end{cases} \tag{8}$$

By taking the longest results of the repeating cycles on the RHS's of (6),(7) and (8) i.e. $\texttt{1\_aba}, \texttt{3\_baba}$ and adding every symbol in turn gives $\texttt{1\underline{a}aba} \to \texttt{1\_abaa}, \texttt{1\underline{a}abaa} \to \texttt{C}, \texttt{1\underline{b}abaa} \to \texttt{E}, \texttt{1\underline{c}abaa} \to \texttt{3b}^5\texttt{\_}, \texttt{1\underline{b}aba} \to \texttt{3\_baba}, \texttt{3\underline{a}baba} \to \texttt{A}, \texttt{3\underline{b}baba} \to \texttt{A}, \texttt{3\underline{c}baba} \to \texttt{E}, \texttt{1\underline{c}aba} \to \texttt{2bbbb\_}$ where the capital letter notation for pseudo-states in Table 2 has been used. Another round of this generates all the results of Table 2 because in Table 2 all possible new symbols to be read are included.

Table 2: A finite state machine going left derived from TM 5

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

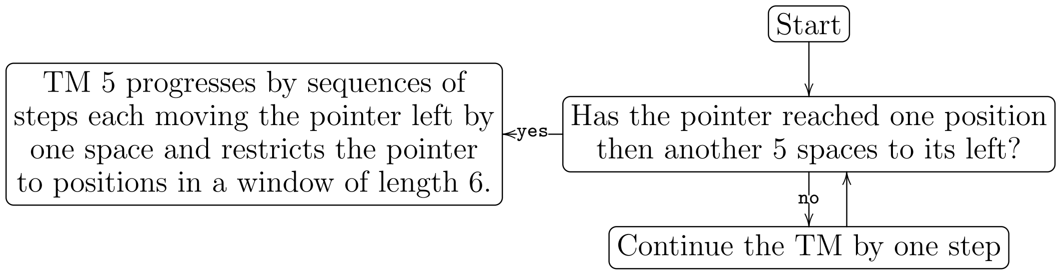
Start
TM 5 progresses by sequences of steps each moving the pointer left by one space and restricts the pointer to positions in a window of length 6.
yes
Has the pointer reached one position then another 5 spaces to its left?
no
Continue the TM by one step

Figure 2: Summary of the results of the analysis of TM(5)

Table 2 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. This is because Table 2 includes every symbol added at the left in every pseudo-state and the effect in each case is to leave the pointer to the left of the string. 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 window applies. This condition of course will not necessarily happen but once started (depending on the initial contents of the tape) must continue indefinitely.

Consider how the development of `2|bb_` in (6) is obtained from that of `2|b_`. Putting a `b` on the left gives the first result `1`$\underline{\texttt{b}}$`aba` $\rightarrow$ `3_baba`. This results in two loops because both starting points `3`$\underline{\texttt{c}}$`bad` and `3bbb`$\underline{\texttt{d}}$ match the endpoint `3_baba`. This is indicated by `3`$\underline{\texttt{c}}$`bad` $\rightarrow$ `3bbb`$\underline{\texttt{d}}$ $\rightarrow$ `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.

The behaviours of the two examples of TM's is completely different though each description does depend somewhat on the configuration the TM starts off in.

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] Turing machine notes 2025.pdf

[6] Turing machine notes 2025.txt