Small semi-weakly universal Turing machines

Damien Woods[1] and Turlough Neary[2]

[1] Department of Computer Science,
University College Cork, Ireland. d.woods@cs.ucc.ie
[2] TASS, Department of Computer Science,
National University of Ireland Maynooth, Ireland. tneary@cs.may.ie

Abstract. We present two small universal Turing machines that have 3 states and 7 symbols, and 4 states and 5 symbols respectively. These machines are semi-weak which means that on one side of the input they have an infinitely repeated word and on the other side there is the usual infinitely repeated blank symbol. This work can be regarded as a continuation of early work by Watanabe on semi-weak machines. One of our machines has only 17 transition rules making it the smallest known semi-weakly universal Turing machine. Interestingly, our two machines are symmetric with Watanabe's 7-state and 3-symbol, and 5-state and 4-symbol machines, even though we use a different simulation technique.

1 Introduction

Shannon [22] was the first to consider the question of finding the smallest possible universal Turing machine, where size is the number of states and symbols. From the early sixties, Minsky and Watanabe had a running competition to see who could come up with the smallest machines [10, 11, 23–25]. In 1962, Minsky [11] found a small 7-state, 4-symbol universal Turing machine. Minsky's machine worked by simulating 2-tag systems, which where shown to be universal by Cocke and Minsky [2]. Rogozhin [20] extended Minsky's technique of 2-tag simulation and found small machines with a number of state-symbol pairs. Subsequently, some of Rogozhin's machines were reduced in size or improved by Robinson [19], Rogozhin [21], Kudlek and Rogozhin [6], Baiocchi [1]. Neary and Woods [12, 15] have recently found small machines that simulate another variant of tag systems called bi-tag systems. All of the smallest known Turing machines, that obey the standard definition (deterministic, one tape, one head), simulate either 2-tag or bi-tag systems. They are plotted as circles in Figure 1.

Interestingly, Watanabe [23–25] managed to find small machines (some were smaller than Minsky's) by generalising the standard Turing machine definition. Instead of having an infinitely repeated blank symbol to the left and right of the input, Watanabe's machines have an infinitely repeated word to one side of the input and an infinitely repeated blank symbol to the other side. We call such machines semi-weak. Watanabe found 7-state, 3-symbol, and 5-state, 4-symbol semi-weakly universal machines that are plotted as hollow diamonds in Figure 1.

A further generalisation are weak machines where we allow an infinitely repeated word to the left of the input and another to the right. Cook [3] and

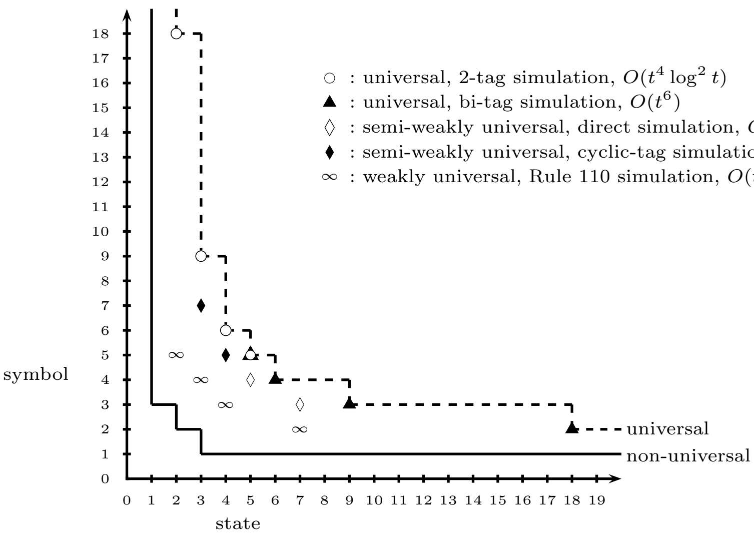

Fig. 1. State-symbol plot of the smallest universal Turing machines to date. Our semi-weak machines are shown as solid diamonds and Watanabe's as hollow diamonds. The standard and semi-weak machines are symmetric about the line where state = symbol.

Wolfram [26] have found very small weakly universal machines which are illustrated as ∞ symbols in Figure 1. These weak machines simulate the cellular automaton Rule 110. Cook [3] proved (the proof is also sketched in Wolfram [26]) that Rule 110 is universal by showing that it simulates cyclic tag systems, which in turn simulate 2-tag systems.

The non-universal curve in Figure 1 is shown for the standard Turing machine definition. The 1-symbol case is trivial, and the 1-state case was shown by Shannon [22] and, via another method, Hermann [4]. Pavlotskaya [16] and, via another method, Kudlek [5], proved there are no universal 2-state, 2-symbol machines, where one transition rule is reserved for halting. Pavlotskaya [17] proved there are no universal 3-state, 2-symbol machines, and also claimed [16], without proof, that there are no universal 2-state, 3-symbol machines. Both cases assume that one transition rule is reserved for halting. It is not difficult to generalise these results to (semi-)weak machines with 1 state or 1 symbol. It is currently unknown if all lower bounds in Figure 1 generalise to (semi-)weak machines.

It is also known from the work of Margenstern [7] and Michel [9] that the region between the non-universal curve and the smallest standard universal machines contains (standard) machines that simulate the $3x+1$ problem and other related problems. These results, along with the weakly and semi-weakly universal machines, lend weight to the idea that finding non-universal lowerbounds in this region is difficult. For results on other generalisations of the Turing machine model see [8, 18], for example.

Figure 1 shows our two new semi-weak machines as solid diamonds. These machines simulate cyclic tag systems, which were used [3] to show that Rule 110 is universal. It is interesting to note that our machines are symmetric with those of Watanabe, despite the fact that we use a different simulation technique. Our 4-state, 5-symbol machine has only 17 transition rules, making it the smallest known semi-weakly universal machine (Watanabe's 5-state, 4-symbol machine has 18 transition rules). The time overhead for our machines is polynomial. More precisely, if M is a single tape deterministic Turing machine that runs in time t, then M is simulated by each of our semi-weak machines in time $O(t^4 \log^2 t)$. See [13, 14, 27, 28] for further results and discussion related to the time complexity of small universal Turing machines.

1.1 Preliminaries

All of the Turing machines considered in this paper are deterministic and have one tape. Our 3-state, 7-symbol universal Turing machine is denoted $U_{3,7}$ and our 4-state, 5-symbol machine is denoted $U_{4,5}$. We let $\langle x \rangle$ denote the encoding of x. We write $c_1 \vdash c_2$ when configuration c_2 follows from c_1 in 1 computation step, and $c_1 \vdash^t c_2$ when c_2 follows from c_1 in t steps.

2 Cyclic tag systems

We begin by defining cyclic tag systems [3].

Definition 1 (cyclic tag system). *A cyclic tag system $C = \alpha_0, \alpha_1, \ldots, \alpha_{p-1}$ is a list of binary words $\alpha_m \in \{0,1\}^*$ called appendants.*

A *configuration* of a cyclic tag system consists of (i) a *marker* $m \in \{0, 1, \ldots, p-1\}$ that points to a single appendant α_m in C, and (ii) a *dataword* $w = x_0 x_1 \ldots x_{l-1} \in \{0,1\}^*$. Intuitively the list C is a program with the marker pointing at instruction α_m. At the initial configuration the marker points at appendant α_0 and w is the binary input word.

Definition 2 (computation step of a cyclic tag system). *A computation step is deterministic and acts on a configuration in one of two ways:*

- *If $x_0 = 0$ then x_0 is deleted and the marker moves to appendant $\alpha_{(m+1) \bmod p}$.*
- *If $x_0 = 1$ then x_0 is deleted, the word α_m is appended onto the right end of w, and the marker moves to appendant $\alpha_{(m+1) \bmod p}$.*

A cyclic tag system completes its computation if (i) the dataword is the empty word, or (ii) it enters a repeating sequence of configurations. The complexity measures of time and space are defined in the obvious way.

Example 1. (cyclic tag system computation) Let $C = 00, 1010, 10$ be a cyclic tag system with input word 0010010. Below we give the first four steps of the computation. In each configuration C is given on the left with the marked appendant highlighted in bold font.

$$
\begin{array}{llllll}
 & \mathbf{00}, 1010, 10 & 0010010 & \vdash & 00, \mathbf{1010}, 10 & 010010 \\
\vdash & 00, 1010, \mathbf{10} & 10010 & \vdash & \mathbf{00}, 1010, 10 & 001010 \\
\vdash & 00, \mathbf{1010}, 10 & 01010 & \vdash & \ldots &
\end{array}
$$

Cyclic tag systems were proved universal by their ability to simulate 2-tag systems [3]. Recently we have shown that cyclic tag systems simulate Turing machines in polynomial time:

Theorem 1 ([13]). *Let M be a single-tape deterministic Turing machine that computes in time t. There is a cyclic tag system C_M that simulates the computation of M in time $O(t^3 \log t)$.*

Note that in order to calculate this upper bound we substitute space bounds for time bounds whenever possible in the analysis.

3 3-state, 7-symbol universal Turing machine

$U_{3,7}$ simulates cyclic tag systems. The cyclic tag system binary input dataword is written directly to the tape, no special encoding is required. The cyclic tag system's list of appendants is reversed and encoded to the left of the input. This encoded list is repeated infinitely often to the left. $U_{3,7}$ computes by erasing one encoded appendant for each 0 on the dataword. If the symbol 1 is read then the next available (encoded) appendant to the left is appended to the dataword, and the appendant is erased. Since the appendants are repeated to the left, this process increments (mod p) through the list of appendants.

3.1 $U_{3,7}$

	u_1	u_2	u_3
0	$\lambda\mathrm{L}u_1$	$\cancel{1}\mathrm{R}u_2$	$\cancel{1}\mathrm{R}u_3$
1	$\lambda\mathrm{L}u_2$	$z\mathrm{R}u_2$	$z\mathrm{R}u_3$
λ	$b\mathrm{R}u_1$	$b\mathrm{R}u_2$	$b\mathrm{R}u_3$
$\emptyset$	$\lambda\mathrm{L}u_1$	$\lambda\mathrm{L}u_3$	$b\mathrm{R}u_2$
$\cancel{1}$		$0\mathrm{L}u_2$	$1\mathrm{L}u_2$
z	$b\mathrm{R}u_1$	$1\mathrm{L}u_2$	$b\mathrm{R}u_1$
b	$\lambda\mathrm{L}u_1$	$\lambda\mathrm{L}u_2$	$b\mathrm{R}u_3$

Table of behaviour for $U_{3,7}$. The start state is u_1 and the blank symbol is $\cancel{1}$.

3.2 Encoding

For our 3-state, 7-symbol machine an appendant $\alpha \in \{0,1\}^*$ is encoded in the following manner. Firstly, the order of the symbols in α is reversed to give α_R. Then the symbol 0 is encoded as $\emptyset\emptyset$, and 1 is encoded as $b\emptyset$. The encoded α_R is then prepended with the two symbols $z\emptyset$. For example, if $\alpha = 100$ then this appendant is encoded as $\langle\alpha\rangle = z\emptyset\emptyset\emptyset\emptyset\emptyset b\emptyset$. Finally the order of appendants are also reversed so that the list of appendants $\alpha_0, \alpha_1, \ldots, \alpha_{p-1}$ are encoded as $\langle\alpha_{p-1}\rangle\langle\alpha_{p-2}\rangle \ldots \langle\alpha_0\rangle$. This encoded list is repeated infinitely often, to the left, on the tape of $U_{3,7}$. The blank symbol for $U_{3,7}$ is $\cancel{1}$ and the cyclic tag system input is written directly on the tape of $U_{3,7}$. Thus the initial configuration of the cyclic tag system given in Example 1 is encoded as

$$\boldsymbol{u_1},\quad \ldots z\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset b\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset\emptyset\emptyset\ \underline{0}010010\ \not1\not1\not1\ldots \tag{1}$$

where the underline denotes the tape head position, the three encoded appendants are repeated infinitely to the left, and the extra whitespace is for human readability purposes only.

3.3 Simulation

To show how $U_{3,7}$ computes we simulate the first 3 steps of the cyclic tag computation from Example 1.

Example 2. Beginning with the configuration given in Equation (1), $U_{3,7}$ reads the leftmost 0 in the input, in state u_1, and then indexes the second encoded appendant to the left, changing each symbol to λ until it reaches z, to give the configuration

$$\boldsymbol{u_1},\quad \ldots z\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset b\emptyset\emptyset\emptyset b\emptyset\ \underline{z}\lambda\lambda\lambda\lambda\lambda\ \lambda010010\ \not1\not1\not1\ldots$$

These steps have the effect of reading and erasing the first 0 in the dataword (input), and simulating the incrementing of the marker to the next (second) appendant. The head then scans right, to read the second dataword symbol.

$$\boldsymbol{u_1},\quad \ldots z\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset b\emptyset\emptyset\emptyset b\emptyset\ bbbbbb\ b\underline{0}10010\ \not1\not1\not1\ldots$$

Again we read 0 in the dataword which causes us to index the third appendant

$$\boldsymbol{u_1},\quad \ldots z\emptyset\emptyset\emptyset b\emptyset\ \underline{z}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda10010\ \not1\not1\not1\ldots$$

and then return to the third input symbol.

$$\boldsymbol{u_1},\quad \ldots z\emptyset\emptyset\emptyset b\emptyset\ bbbbbbbb\ bbbbbb\ bb\underline{1}0010\ \not1\not1\not1\ldots$$

The input symbol 1 causes $U_{3,7}$ to enter a 'print cycle' which iterates the following: we scan left in state u_2, if we read $\emptyset\emptyset$ then we scan right in state u_2 and print 0, if we read $b\emptyset$ then we scan right in state u_3 and print 1. We exit the cycle if we read $z\emptyset$. We now contine our simulation to the point where we are about to read an encoded 1 in the third appendant

$$\boldsymbol{u_3},\quad \ldots z\emptyset\emptyset\emptyset\underline{b}\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda0010\ \not1\not1\not1\ldots$$

This causes $U_{3,7}$ to scan right, append a 1 to the dataword, and return left to read the next encoded symbol in the third appendant

$$\boldsymbol{u_3},\quad \ldots z\emptyset\underline{\emptyset}\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda0010\ 1\not1\not1\not1\ldots$$

which causes a 0 to be printed, and we return left

$$\boldsymbol{u_3},\quad \ldots \underline{z}\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda0010\ 10\not1\not1\not1\ldots$$

where the string $z\emptyset$ marks the end of the encoded appendant and causes $U_{3,7}$ to exit the print cycle and return to state u_1; the index cycle.

$$\boldsymbol{u_1}, \quad \ldots z\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset b\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset\emptyset\emptyset\ bbbbbb\ bbbbbbbb\ bbbbbb\ bbb\underline{0}010\ 10\not{1}\not{1}\not{1}\ldots$$

The latter configuration shows the next set of encoded appendants to the left. At this point we have simulated the third computation step in Example 1. □

As can be seen in the preceding example, the computation of $U_{3,7}$ is relatively straightforward, so we refrain from giving a full proof of correctness.

Section 2 gives two conditions for a cyclic tag system completing its computation (halting). The first condition (empty dataword) is simulated by $U_{3,7}$ in a very straightforward way: if the dataword is empty then $U_{3,7}$ reads a blank symbol $\not{1}$ in state u_1, and immediately halts. The second condition (repeating sequence of cyclic tag configurations) causes $U_{3,7}$ to simulate this loop in an easily detectable way, where some fixed sequence of appendants are repeatedly appended to the dataword.

4 4 state, 5 symbol machine

$U_{4,5}$ bears some similarities to the previous machine in that it simulates cyclic tag systems which are encoded to the left. However its computation is somewhat more complicated. $U_{4,5}$ simulates a restricted cyclic tag system where the dataword does not contain consecutive 1 symbols. In particular, we say that the dataword and all appendants are words from $\{0, 10\}^*$. Such a cyclic tag system simulates an arbitrary cyclic tag system with only a small constant factor slowdown (using the simulation from [13]). Furthermore, in two different cycles, $U_{4,5}$ makes special use of whether specific substrings on the tape are of odd or even length. Intuitively this kind of encoding helps to keep the program small.

4.1 $U_{4,5}$

	u_1	u_2	u_3	u_4
0	$\lambda \mathrm{L} u_1$	$\lambda \mathrm{L} u_2$	$\emptyset \mathrm{R} u_3$	$\emptyset \mathrm{R} u_4$
1	$\emptyset \mathrm{R} u_2$	$1 \mathrm{L} u_2$	$1 \mathrm{R} u_3$	$1 \mathrm{R} u_4$
λ	$0 \mathrm{R} u_2$	$0 \mathrm{R} u_1$	$0 \mathrm{R} u_4$	$0 \mathrm{R} u_3$
$\emptyset$		$0 \mathrm{L} u_2$	$0 \mathrm{L} u_2$	$1 \mathrm{L} u_2$
$\not{1}$	$0 \mathrm{L} u_2$	$\lambda \mathrm{L} u_3$		

Table of behaviour for $U_{4,5}$. The start state is u_1 and the blank symbol is $\emptyset$.

4.2 Encoding

An appendant $\alpha \in \{0, 10\}^*$ is encoded in the following manner. Firstly, the order of the symbols in α is reversed to give α_R. Then the symbol 0 is encoded as $0\lambda\not{1}0$, and 1 is encoded as $00\lambda\not{1}$. The encoded α_R is then prepended with the symbol λ. For example, if $\alpha = 100$ then this appendant is encoded as $\langle\alpha\rangle = \lambda 0 \lambda \not{1} 00 \lambda \not{1} 000 \lambda \not{1}$. Finally the order of appendants are also reversed so that the list of appendants $\alpha_0, \alpha_1, \ldots, \alpha_{p-1}$ are encoded as $\langle\alpha_{p-1}\rangle\langle\alpha_{p-2}\rangle \ldots \langle\alpha_0\rangle$. This encoded list is repeated infinitely often, to the left, on the tape of $U_{4,5}$. The blank symbol for $U_{4,5}$ is $\emptyset$ and the cyclic tag system input, an element of $\{0, 10\}^*$, is written directly on the tape of $U_{4,5}$. Thus the initial configuration of the cyclic tag system given in Example 1 is encoded as

$$\boldsymbol{u_1}, \quad \ldots \lambda 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 000 \lambda \not{1} 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 00 \lambda \not{1} 0\ \underline{0}010010\ \emptyset\emptyset\emptyset \ldots \tag{2}$$

where the underline denotes the tape head position, the three encoded appendants are repeated infinitely to the left, and the extra whitespace is for human readability purposes only.

4.3 Simulation

In order to show how $U_{4,5}$ computes, we simulate the first 4 steps of the cyclic tag computation from Example 1. Example 3 shows $U_{4,5}$ reading two 0 symbols in the dataword and indexing appendants. Example 4 shows $U_{4,5}$ reading a 10 in the dataword, printing one appendant and indexing the next. Lemmata 1 and 2 build on these examples to give a proof of correctness.

Example 3 ($U_{4,5}$; reading 0). Beginning with the configuration given in Equation (2), $U_{4,5}$ reads the leftmost 0 in the input, in state u_1, and begins the process of indexing the second appendant to the left, using states u_1 and u_2.

$$\begin{aligned}
&\vdash^3 \ \boldsymbol{u_2}, \quad \ldots \lambda 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 000 \lambda \not{1} 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 00 \underline{\lambda} 0 \lambda\ \lambda 010010\ \emptyset\emptyset\emptyset \ldots \\
&\vdash \quad \boldsymbol{u_1}, \quad \ldots \lambda 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 000 \lambda \not{1} 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 000 \underline{0} \lambda\ \lambda 010010\ \emptyset\emptyset\emptyset \ldots \\
&\vdash^4 \ \boldsymbol{u_1}, \quad \ldots \lambda 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 000 \lambda \not{1} 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \underline{\not{1}} \lambda\lambda\lambda\lambda\lambda\ \lambda 010010\ \emptyset\emptyset\emptyset \ldots
\end{aligned}$$

until we read $\lambda 0$, to give the configuration

$$\vdash^5 \ \boldsymbol{u_1}, \quad \ldots \lambda 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 000 \lambda \not{1} 0 \lambda \not{1} 000 \lambda \not{1}\ \underline{\lambda}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda 010010\ \emptyset\emptyset\emptyset \ldots \tag{3}$$

Upon reading λ in state u_1, $U_{4,5}$ scans right, switching between states u_1 and u_2. There are an even number of consecutive λ symbols, thus we exit the string of λ symbols in state u_1, ready to read the next input symbol.

$$\vdash^{10} \ \boldsymbol{u_1}, \quad \ldots \lambda 0 \lambda \not{1} 000 \lambda \not{1}\ \lambda 0 \lambda \not{1} 000 \lambda \not{1} 0 \lambda \not{1} 000 \lambda \not{1}\ 000000000\ 0\underline{0}10010\ \emptyset\emptyset\emptyset \ldots$$

It can be seen from the proceeding configurations, that whenever $U_{4,5}$ enters an encoded appendant from the right and in state u_1, then the encoded appendant is erased. Assume, for the moment, that every symbol in the dataword is 0. Then for each erased appendant it is the case that exactly one dataword symbol has also been erased. Encoded appendants are of odd length. Therefore the string of consecutive λ symbols is always of even length immediately after erasing an appendant, e.g. in configurations of the form given in Equation (3). Thus it can be seen that even though $U_{4,5}$ switches between two states, u_1 and u_2, while scanning right through the string of λ symbols, it always exits this string on the right to read the next binary dataword symbol in state u_1.

We continue our simulation: the next dataword symbol (again 0) is erased and the next appendant is erased to give:

$$\vdash^{35} \quad \boldsymbol{u_1}, \ \ldots\lambda0\lambda\not{1}000\lambda\not{1}\ \underline{\lambda}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda10010\ \emptyset\emptyset\emptyset\ldots$$

We then scan right through the (even length) string of λ symbols, switching between states u_1 and u_2, to read the next dataword symbol in state u_1:

$$\vdash^{28} \quad \boldsymbol{u_1}, \ \ldots\lambda0\lambda\not{1}000\lambda\not{1}\ 00000000000000000\ 000000000\ 00\underline{1}0010\ \emptyset\emptyset\emptyset\ldots \tag{4}$$

The example is complete. □

The following example illustrates how $U_{4,5}$ simulates the reading of 10 in the dataword. Specifically, the 10 is erased from the dataword, we append and erase the indexed appendant, and finally we erase the following appendant.

Example 4 ($U_{4,5}$; reading 10*).* Recall that, for $U_{4,5}$, any 1 in the dataword is immediately followed by a 0. When $U_{4,5}$ reads a 1 in the dataword it then (i) erases the 10 pair, (ii) enters a print cycle (to simulate appending the indexed appendant) and then enters (iii) an index cycle (to simulate the reading of the 0 and indexing the next appendant).

We continue from configuration (4) above.

$$\begin{aligned}
&\vdash \quad \boldsymbol{u_2}, \ \ldots\lambda0\lambda\not{1}000\lambda\not{1}\ 00000000000000000\ 000000000\ 00\emptyset\underline{0}010\ \emptyset\emptyset\emptyset\ldots\\
&\vdash \quad \boldsymbol{u_2}, \ \ldots\lambda0\lambda\not{1}000\lambda\not{1}\ 00000000000000000\ 000000000\ 00\underline{\emptyset}\lambda010\ \emptyset\emptyset\emptyset\ldots\\
&\vdash \quad \boldsymbol{u_2}, \ \ldots\lambda0\lambda\not{1}000\lambda\not{1}\ 00000000000000000\ 000000000\ 0\underline{0}0\lambda010\ \emptyset\emptyset\emptyset\ldots\\
&\vdash^{27} \quad \boldsymbol{u_2}, \ \ldots\lambda0\lambda\not{1}000\lambda\not{1}\ \underline{0}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda0\lambda010\ \emptyset\emptyset\emptyset\ldots
\end{aligned}$$

We now begin reading the encoded appendant, which encodes 10.

$$\vdash \quad \boldsymbol{u_2}, \ \ldots\lambda0\lambda\not{1}000\lambda\underline{\not{1}}\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda0\lambda010\ \emptyset\emptyset\emptyset\ldots$$

This encoded appendant tells us that the symbol 1 (encoded as $00\lambda\not{1}$), and then the symbol 0 (encoded as $0\lambda\not{1}0$), should be appended to the dataword.

$$\vdash \quad \boldsymbol{u_3}, \ \ldots\lambda0\lambda\not{1}000\underline{\lambda}\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda0\lambda010\ \emptyset\emptyset\emptyset\ldots$$

$U_{4,5}$ now scans right, switching between state u_3 and u_4, eventually appending either 0 or 1 to the dataword. If there are an odd number of λ symbols on, and to the right of, the tape head then 1 is appended, if there is an even number then 0 is appended. Such a printing mechanism uses a relatively small number of transition rules.

$$\begin{aligned}
&\vdash \quad \boldsymbol{u_4}, \ \ldots\lambda0\lambda\not{1}0000\underline{\lambda}\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda0\lambda010\ \emptyset\emptyset\emptyset\ldots\\
&\vdash \quad \boldsymbol{u_3}, \ \ldots\lambda0\lambda\not{1}00000\ \underline{\lambda}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda0\lambda010\ \emptyset\emptyset\emptyset\ldots\\
&\vdash^{30} \quad \boldsymbol{u_4}, \ \ldots\lambda0\lambda\not{1}00000\ 00000000000000000\ 000000000\ 00\emptyset0\underline{0}10\ \emptyset\emptyset\emptyset\ldots
\end{aligned}$$

We now pass over the dataword and append a 1.

$$\vdash^{3} \quad \boldsymbol{u_4}, \ \ldots \lambda 0 \lambda \cancel{1} 00000\ 00000000000000000\ 000000000\ 00\cancel{0}0\cancel{0}1\cancel{0}\ \underline{\cancel{0}}\cancel{0}\cancel{0}\cancel{0} \ldots$$
$$\vdash \quad \boldsymbol{u_2}, \ \ldots \lambda 0 \lambda \cancel{1} 00000\ 00000000000000000\ 000000000\ 00\cancel{0}0\cancel{0}1\underline{\cancel{0}}\ 1\cancel{0}\cancel{0}\cancel{0} \ldots$$

We now scan left to find the next symbol to be appended

$$\vdash^{37} \quad \boldsymbol{u_2}, \ \ldots \lambda 0 \lambda \cancel{1} \underline{0} \lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda 0 \lambda 010\ 1\cancel{0}\cancel{0}\cancel{0} \ldots$$

which is an encoded 0. We erase this encoded 0:

$$\vdash^{2} \quad \boldsymbol{u_3}, \ \ldots \lambda 0 \underline{\lambda} \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda 0 \lambda 010\ 1\cancel{0}\cancel{0}\cancel{0} \ldots$$

Now we are ready to scan right, switching between states u_3 and u_4. There are an even number of λ symbols on, and to the right of, the tape head. This will result in a 0 being appended to the dataword.

$$\vdash^{41} \quad \boldsymbol{u_3}, \ \ldots \lambda 00000000\ 00000000000000000\ 000000000\ 00\cancel{0}0\cancel{0}1\cancel{0}\ 1\underline{\cancel{0}}\cancel{0}\cancel{0}\cancel{0} \ldots$$
$$\vdash \quad \boldsymbol{u_2}, \ \ldots \lambda 00000000\ 00000000000000000\ 000000000\ 00\cancel{0}0\cancel{0}1\cancel{0}\ \underline{1}0\cancel{0}\cancel{0}\cancel{0} \ldots$$

$U_{4,5}$ now scans left, in state u_2, and since there are no more encoded 0 or 1 symbols, it eventually reads the 'end of appendant' marker λ.

$$\vdash^{42} \quad \boldsymbol{u_2}, \ \ldots \underline{\lambda} \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda 0 \lambda 010\ 10\cancel{0}\cancel{0}\cancel{0} \ldots$$

Reading this λ in state u_2 sends us to the right in the index cycle (switching between states u_2 and u_1); however we enter the cycle in the 'incorrect' state u_2 (we usually enter this cycle in state u_1), but when we read the leftmost 0 in the dataword

$$\vdash^{37} \quad \boldsymbol{u_1}, \ \ldots 000000000\ 00000000000000000\ 000000000\ 00\underline{0}\lambda 010\ 10\cancel{0}\cancel{0}\cancel{0} \ldots$$

this forces us to index another appendant (after which we will enter the next index cycle in state u_1; the 'correct' state). This is the main reason why we insist that each 1 in the dataword is immediately followed by a 0.

We duplicate the configuration immediately above (while introducing some shorthand notation for erased appendants and showing the next two encoded appendants to the left).

$$\boldsymbol{u_1}, \ \ldots \lambda 0 \lambda \cancel{1} 000 \lambda \cancel{1} 0 \lambda \cancel{1} 000 \lambda \cancel{1}\ \lambda 0 \lambda \cancel{1} 00 \lambda \cancel{1} 0\ 0^{9}\ \ 0^{17}\ \ 0^{9}\ \ 00\underline{0}\lambda 010\ 10\cancel{0}\cancel{0}\cancel{0} \ldots$$

As already noted, we are forced to index the next appendant:

$$\vdash^{50} \quad \boldsymbol{u_1}, \ \ldots \lambda 0 \lambda \cancel{1} 000 \lambda \cancel{1} 0 \lambda \cancel{1} 000 \lambda \cancel{1}\ \underline{\lambda} \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda^{9}\ \ \lambda^{17}\ \ \lambda^{9}\ \ \lambda\lambda\lambda\lambda 010\ 10\cancel{0}\cancel{0}\cancel{0} \ldots$$

We then scan right through the (even length) string of λ symbols, switching between states u_1 and u_2 to read the next dataword read symbol in state u_1:

$$\vdash^{48} \quad \boldsymbol{u_1}, \ \ldots \lambda 0 \lambda \cancel{1} 000 \lambda \cancel{1} 0 \lambda \cancel{1} 000 \lambda \cancel{1}\ 000000000\ 0^{9}\ \ 0^{17}\ \ 0^{9}\ \ 0000\underline{0}10\ 10\cancel{0}\cancel{0}\cancel{0} \ldots$$

The example is complete. $\square$

The halting conditions for $U_{4,5}$ are the same as those for $U_{3,7}$; if the cyclic tag systems halts then $U_{4,5}$ reads a $\emptyset$ in state u_1 and halts, if the cyclic tag systems enters a repeating sequence of configurations then $U_{4,5}$ simulates this loop in an easily detectable way.

The previous two examples provide the main mechanics for the workings of $U_{4,5}$. The two lemmata below generalise these examples, and cover the cases of read symbols 0 and 1 respectively. We assume that the cyclic tag dataword and appendants are from $\{0, 10\}^*$, as described at the beginning of Section 4.

Lemma 1. *Let c_1 be a configuration of cyclic tag system C with read symbol* 0*, and let c_2 be the unique configuration that follows c_1 using C (i.e. $c_1 \vdash_C c_2$). Given an encoding of C and c_1, then $U_{4,5}$ computes the encoding of c_2.*

Proof. In the encoding of c_1, $U_{4,5}$ is reading 0 in state u_1. This causes the head to move left leaving a string of λ symbols. An encoded appendant is a word over $\lambda\{\langle 0\rangle, \langle 1\rangle\langle 0\rangle\}^*$. Notice if we enter either $\langle 0\rangle = 0\lambda\cancel{1}0$ or $\langle 1\rangle = 00\lambda\cancel{1}$ from the right, in state u_1, then we exit to the left, in the same state, leaving $\lambda\lambda\lambda\lambda$ on the tape. Eventually the entire appendant is erased (converted into a string of λ symbols), and $U_{4,5}$ is reading the leftmost λ in the encoded appendant, in state u_1.

From the encoding, the length of each encoded appendant is odd. Furthermore, the number of erased appendants is equal to number of erased dataword symbols. Thus, the sum of the number of erased dataword symbols plus the number of symbols in the erased appendants is even. We begin reading this even length string of λ symbols from the left in state u_1, alternating between states u_1 and u_2 as we scan right. We exit the string of λ symbols in state u_1. We have completed the index cycle and are reading the the leftmost (next read) symbol from the dataword in state u_1. From above, the next appendant is indexed. Thus the tape encodes configuration c_2. $\square$

Lemma 2. *Let c_1 be a configuration of cyclic tag system C with read symbol* 1*, and let c_2 be the unique configuration that follows c_1 using C (i.e. $c_1 \vdash_C c_2$). Given an encoding of C and c_1, then $U_{4,5}$ computes the encoding of c_2.*

Proof. Recall that any 1 in the dataword is immediately followed by a 0. Thus our proof has two parts, a print cycle followed by an index cycle.

In the encoding of c_1, $U_{4,5}$ is reading 1 in state u_1. This 1 is changed to $\emptyset$, and the head moves to the right and erases an extra 0 symbol. The $\emptyset$ is changed to 0 (which is used to trigger an extra index cycle below). The head then scans left in state u_2 leaving a string of λ symbols until we read the first (rightmost) non-erased encoded appendant. An encoded appendant is a word over $\lambda\{\langle 0\rangle, \langle 1\rangle\langle 0\rangle\}^*$.

Notice that if we enter $\langle 0\rangle = 0\lambda\cancel{1}0$ from the right in state u_2, we then (i) exit to the right in state u_4. However if we enter $\langle 1\rangle = 00\lambda\cancel{1}$ from the right in state u_2 we then (ii) exit to the right in state u_3. In both cases we then scan to the right, reading an *odd* number of λ symbols (a string of the form $\lambda^{2i}0\lambda$, $i \in \mathbb{N}$), while switching between states u_3 and u_4. We pass to the right over the dataword, which does not cause us to change state. Then in case (i) we append 0 to the dataword and in case (ii) we append a 1 to the dataword.

We continue appending 0 or 1 symbols until we reach the leftmost end of the (currently indexed) appendant by reading the symbol λ in state u_2. We then scan right, through a string of the form $\lambda^{2j+1}0\lambda, j \in \mathbb{N}$, switching between states u_2 and u_1. After $2j+1$ steps we read 0 in state u_1, which triggers an index cycle (Lemma 1). After the index cycle we pass over the rightmost λ (which occupies the location of the extra erased 0 mentioned above) and we are reading the next encoded dataword symbol in state u_1. Thus the tape encodes configuration c_2. □

Let C be a cyclic tag system that runs in time t. After simulating t steps of C, machines $U_{3,7}$ and $U_{4,5}$ have used $O(t)$ workspace. Therefore both machines simulate the computation of C in time $O(t^2)$. By applying Theorem 1 directly we find that given a single-tape deterministic Turing machine M that computes in time t, then machines $U_{3,7}$ and $U_{4,5}$ both simulate M in time $O(t^6 \log^2 t)$. We observe that in the simulation from [13] the space used by C is only a constant times that used by M. This observation, along with an (as yet unpublished) improvement to [13], improve the time bound to $O(t^4 \log^2 t)$ for $U_{3,7}$ and $U_{4,5}$ simulating Turing machines M.

Acknowledgements

DW is supported by Science Foundation Ireland grant number 04/IN3/1524. TN is supported by the Irish Research Council for Science, Engineering and Technology.

References

1. C. Baiocchi. Three small universal Turing machines. In M. Margenstern and Y. Rogozhin, editors, *Machines, Computations, and Universality*, volume 2055 of *LNCS*, pages 1–10, Chişinău, Moldova, May 2001. MCU, Springer.
2. J. Cocke and M. Minsky. Universality of tag systems with $P = 2$. *Journal of the Association for Computing Machinery*, 11(1):15–20, Jan. 1964.
3. M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.
4. G. T. Hermann. The uniform halting problem for generalized one state Turing machines. In *Proceedings of the ninth annual Symposium on Switching and Automata Theory (FOCS)*, pages 368–372, Schenectady, New York, Oct. 1968. IEEE Computer Society Press.
5. M. Kudlek. Small deterministic Turing machines. *Theoretical Computer Science*, 168(2):241–255, 1996.
6. M. Kudlek and Y. Rogozhin. A universal Turing machine with 3 states and 9 symbols. In *Developments in Language Theory (DLT) 2001*, volume 2295 of *LNCS*, pages 311–318, Vienna, May 2002. Springer.
7. M. Margenstern. Frontier between decidability and undecidability: a survey. *Theoretical Computer Science*, 231(2):217–251, Jan. 2000.

8. M. Margenstern and L. Pavlotskaya. On the optimal number of instructions for universality of Turing machines connected with a finite automaton. *International Journal of Algebra and Computation*, 13(2):133–202, Apr. 2003.
9. P. Michel. Small Turing machines and generalized busy beaver competition. *Theoretical Computer Science*, 326:45–56, Oct. 2004.
10. M. Minsky. A 6-symbol 7-state universal Turing machines. Technical Report 54-G-027, MIT, Aug. 1960.
11. M. Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, volume 5, pages 229–238, Provelence, 1962. AMS.
12. T. Neary. Small polynomial time universal Turing machines. In *Fourth Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT'06)*, pages 325–329, Ireland, 2006. University College Cork.
13. T. Neary and D. Woods. P-completeness of cellular automaton Rule 110. In *International Colloquium on Automata Languages and Programming (ICALP)*, volume 4051 (Part I) of *LNCS*, pages 132–143. Springer, July 2006.
14. T. Neary and D. Woods. Small fast universal Turing machines. *Theoretical Computer Science*, 362(1–3):171–195, Oct. 2006.
15. T. Neary and D. Woods. Four small universal Turing machines. In M. Margenstern and Y. Rogozhin, editors, *Machines, Computations and Universality (MCU)*, LNCS, Orléans, France, Sept. 2007. Springer. This volume.
16. L. Pavlotskaya. Solvability of the halting problem for certain classes of Turing machines. *Mathematical Notes (Springer)*, 13(6):537–541, June 1973. (Translated from Matematicheskie Zametki, Vol. 13, No. 6, pp. 899909, June, 1973).
17. L. Pavlotskaya. Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring. *Avtomaty i Mashiny*, pages 91–118, 1978. (Sufficient conditions for the halting problem decidability of Turing machines) (in Russian).
18. L. Priese. Towards a precise characterization of the complexity of universal and nonuniversal Turing machines. *SIAM J. Comput.*, 8(4):508–523, 1979.
19. R. M. Robinson. Minsky's small universal Turing machine. *International Journal of Mathematics*, 2(5):551–562, 1991.
20. Y. Rogozhin. Sem' universal'nykh mashin T'juringa. *Systems and theoretical programming, Mat. Issled.*, 69:76–90, 1982. (Seven universal Turing machines. In Russian).
21. Y. Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, Nov. 1996.
22. C. E. Shannon. A universal Turing machine with two internal states. *Automata Studies, Annals of Mathematics Studies*, 34:157–165, 1956.
23. S. Watanabe. On a minimal universal Turing machines. Technical report, MCB Report, Tokyo, Aug. 1960.
24. S. Watanabe. 5-symbol 8-state and 5-symbol 6-state universal Turing machines. *Journal of the ACM*, 8(4):476–483, Oct. 1961.
25. S. Watanabe. 4-symbol 5-state universal Turing machines. *Information Processing Society of Japan Magazine*, 13(9):588–592, 1972.
26. S. Wolfram. *A new kind of science*. Wolfram Media, Inc., 2002.
27. D. Woods and T. Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 439–446, Berkeley, California, Oct. 2006. IEEE.
28. D. Woods and T. Neary. The complexity of small universal Turing machines. In *Computation and Logic in the Real World: Third Conference of Computability in Europe, CiE 2007*, volume 4497 of *LNCS*, Siena, Italy, June 2007. Springer. Invited.