Fundamenta Informaticae 91 (2009) 161–177
DOI 10.3233/FI-2009-0011
IOS Press

Small Semi-Weakly Universal Turing Machines

Damien Woods[*] [†]
Department of Computer Science and Artificial Intelligence
University of Seville, Spain
d.woods@cs.ucc.ie

Turlough Neary[‡]
Boole Centre for Research in Informatics, School of Mathematics
University College Cork, Ireland
tneary@cs.nuim.ie

Abstract. We present three small universal Turing machines that have 3 states and 7 symbols, 4 states and 5 symbols, and 2 states and 13 symbols, respectively. These machines are semi-weakly universal 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, two of our 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 [27] was the first to discuss the problem 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 [11, 12, 28, 29, 30]. In 1962, Minsky [12] 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 [3].

[*]Damien Woods is supported by Junta de Andalucía grant TIC-581.
[†]Address for correspondence: Department of Computer Science and Artificial Intelligence, University of Seville, Spain
[‡]Turlough Neary is funded by the Irish Research Council for Science, Engineering and Technology, and by Science Foundation Ireland Research Frontiers Programme grant number 07/RFP/CSMF641.

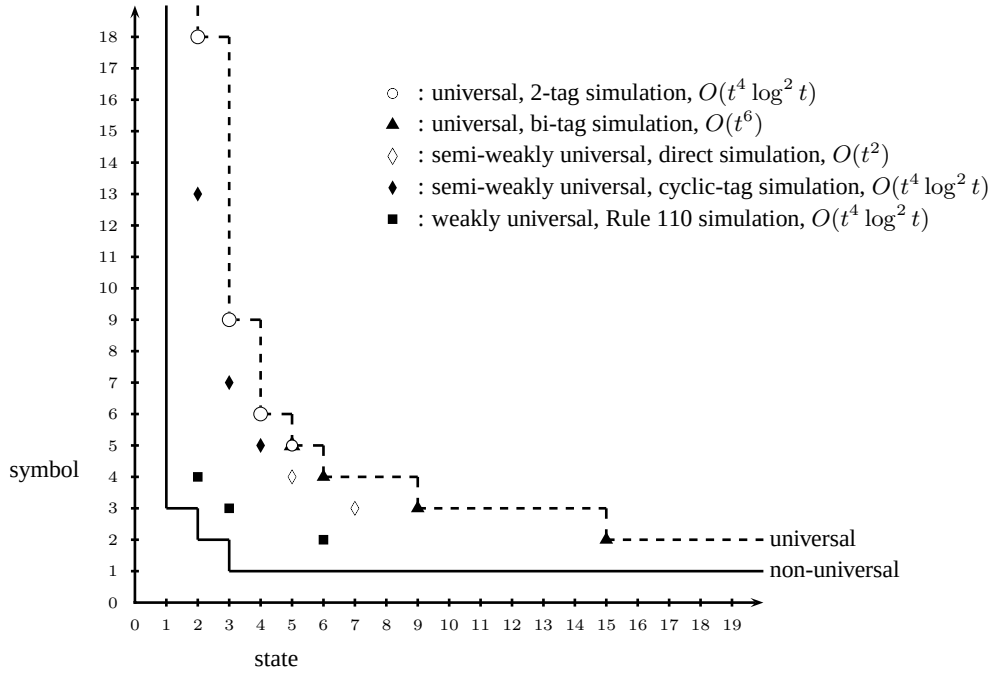

Figure 1. State-symbol plot of the smallest universal Turing machines to date. Our new semi-weak machines are shown as solid diamonds and Watanabe's are shown as hollow diamonds. Simulation time overheads are specified. The non-universal curve shows standard machines that are known to have a decidable halting problem.

Rogozhin [25] 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 [24], Rogozhin [26], Kudlek and Rogozhin [7], Baiocchi [2]. Recently, Neary and Woods [13, 14, 15, 18] have found small machines that simulate another variant of tag systems called bi-tag systems. All of the smallest known universal 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 and triangles in Figure 1.

Interestingly, Watanabe [28, 29, 30] 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 the 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 [4] and Wolfram [31] found very small weakly universal machines (Cook's paper includes a machine by Eppstein). We have improved [19] upon all of these machines to give the weakly universal machines illustrated as squares in Figure 1. These weak machines simulate the cellular automaton Rule 110. Cook [4] proved (the proof is also sketched in Wolfram [31]) 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 shows those standard Turing machines that are known to have a decidable halting problem. The 1-symbol case is trivial, and the 1-state case was shown by Shannon [27] and, via another method, Hermann [5]. Pavlotskaya [20] and, via another method, Kudlek [6], proved there are no universal 2-state, 2-symbol machines, where one transition rule is reserved for halting. Pavlotskaya [21] proved there are no universal 3-state, 2-symbol machines, and also claimed [20], without giving a 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. Lower bounds for semi-weak machines, with a one-way infinite tape, were proven for a generalised Turing machine model where a coupled finite automaton fills out the "blank" region of the tape as required. Margenstern and Pavlotskaya [9] have shown that all such machines with four instructions or less (in the Turing machine program), have a decidable halting problem, and that five instructions are sufficient for universality. Their work also provides a proof of the 2-state, 2-symbol case, regardless of halting (but for one-way infinite machines). 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 [8], Michel [10], and Baiocchi [1] 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. Kudlek [6] has given a 4-state, 4-symbol (standard) machine that accepts a context-sensitive language. These results, along with the weakly and semi-weakly universal machines, lend weight to the idea that finding non-universal lower bounds in this region is difficult. For results on other generalisations of the Turing machine model see [9, 23, 32], for example.

Figure 1 shows our three new semi-weak machines as solid diamonds. These machines simulate cyclic tag systems, which were used [4] to show that Rule 110 is universal. It is interesting to note that two of 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, one less than Watanabe's 5-state, 4-symbol machine.[1] 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 [14, 16, 17, 33, 32] for further results and discussion related to the time complexity of small universal Turing machines.

This work is an extended version of [34] and contains new results.

1.1. Notation

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 $U_{3,7}$, our 4-state, 5-symbol machine is denoted $U_{4,5}$, and our 2-state, 13-symbol machine is denoted $U_{2,13}$. 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.

[1] Note that there is a machine with 17 transistion rules by Pavlotskaya [22], that can also be considered as semi-weakly universal (it uses "periodic extensions"), but this has 4-states and 7-symbols.

2. Cyclic tag systems

We begin by defining cyclic tag systems [4].

Definition 2.1. (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 2.2. (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 $\alpha_{(m+1) \mod p}$.
- 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 $\alpha_{(m+1) \mod p}$.

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

Example 2.1. *(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.

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

Cyclic tag systems were proved universal by their ability to simulate 2-tag systems [4]. Recently we have shown that cyclic tag systems simulate Turing machines in polynomial time:

Theorem 2.1. ([16])
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. The time bound is improved to $O(t^2 \log t)$ in [14].

3. 3-state, 7-symbol 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 dataword 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 p appendants.

3.1. $U_{3,7}$

The table of behaviour for $U_{3,7}$ is given in Table 3.1.

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

Table 3.1. Table of behaviour for $U_{3,7}$. The start state is u_1 and the blank symbol is B.

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 $\langle \alpha \rangle = z\emptyset\emptyset\emptyset\emptyset b\emptyset$. 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 start state for $U_{3,7}$ is u_1, the blank symbol is B, 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 2.1 is encoded as

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

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 three steps of the cyclic tag computation from Example 2.1.

Example 3.1. 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 following configuration after 6 steps

$$\vdash^6 \quad \boldsymbol{u_1},\ \ldots z\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset b\emptyset\emptyset\emptyset b\emptyset\ \underline{z}\lambda\lambda\lambda\lambda\lambda\ \lambda 010010\ BBB\ldots$$

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.

$$\vdash^7 \quad \boldsymbol{u_1},\ \ldots z\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset b\emptyset\emptyset\emptyset b\emptyset\ bbbbbb\ b\underline{0}10010\ BBB\ldots$$

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

$$\vdash^{17} \quad \boldsymbol{u_1},\ \ldots z\emptyset\emptyset\emptyset b\emptyset\ \underline{z}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda 10010\ BBB\ldots$$

and then return to the third input symbol.

$$\vdash^{18} \quad \boldsymbol{u_1},\ \ldots z\emptyset\emptyset\emptyset b\emptyset\ bbbbbbbbbb\ bbbbbb\ bb\underline{1}0010\ BBB\ldots$$

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 move forward by 1 step, changing to state u_2

$$\vdash^1 \quad \boldsymbol{u_2},\ \ldots z\emptyset\emptyset\emptyset b\emptyset\ bbbbbbbbbb\ bbbbbb\ b\underline{b}\lambda 0010\ BBB\ldots$$

and then continue to the point where we are about to read an encoded 1 in the third appendant

$$\vdash^{19} \quad \boldsymbol{u_3},\ \ldots z\emptyset\emptyset\emptyset\underline{b}\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0010\ BBB\ldots$$

This causes $U_{3,7}$ to scan right and append a 1 to right of the (temporarily mangled) dataword,

$$\vdash^{26} \quad \boldsymbol{u_2},\ \ldots z\emptyset\emptyset\emptyset bb\ bbbbbbbbbb\ bbbbbb\ bbbBBz\underline{B}\ 1BBB\ldots$$

and return left to read the next encoded symbol in the third appendant.

$$\vdash^{26} \quad \boldsymbol{u_3},\ \ldots z\emptyset\underline{\emptyset}\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0010\ 1BBB\ldots$$

This causes a 0 to be printed to the right (using state u_2). Afterwards we return left

$$\begin{aligned}\vdash^{57} \quad & \boldsymbol{u_2},\ \ldots z\underline{\emptyset}\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0010\ 10BBB\ldots \\ \vdash^1 \quad & \boldsymbol{u_3},\ \ldots \underline{z}\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0010\ 10BBB\ldots\end{aligned}$$

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.

$$\vdash^{25} \quad \boldsymbol{u_1},\ \ldots z\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset b\emptyset\emptyset\emptyset b\emptyset\ z\emptyset\emptyset\emptyset\emptyset\emptyset\ bbbbbb\ bbbbbbbbbb\ bbbbbb\ bbb\underline{0}010\ 10BBB\ldots$$

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 2.1.

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. 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 B 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.

Let C be a cyclic tag system that runs in time t. After simulating t steps of C, machine $U_{3,7}$ has used $O(t)$ workspace. Therefore it simulates the computation of C in time $O(t^2)$. By applying Theorem 2.1 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 [16] the space used by C is only a constant times that used by M. This observation, along with an improvement to the time overhead [14], improve the time bound to $O(t^4 \log^2 t)$ for $U_{3,7}$ simulating Turing machines M.

Using the same argument, we get the same time overhead for the machines $U_{2,13}$, and $U_{4,5}$, that are given in the following two sections.

4. 4-state, 5-symbol machine

$U_{4,5}$ bears some similarities to the previous machine in that it simulates cyclic tag systems with the appendants encoded to the left of the dataword. However its computation is somewhat more complicated and we use a few extra tricks to keep the program size down. In two different cycles, $U_{4,5}$ makes special use of whether specific substrings on the tape are of odd or even length. Furthermore, $U_{4,5}$ 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 $\{0, 10\}^*$.

To see that such cyclic tag systems are (efficiently) universal, one can directly use the simulation from [16] as it is of the form just described. Alternatively, one can use the following simple construction. Given an arbitrary cyclic tag system C and input w, we create a modified system C' with input w'. Each symbol $y \in \{0, 1\}$ in the appendants of C and in the input w, is encoded as $y0$ in C' and w'. Furthermore, each appendant α_m in C is encoded as the pair of appendants α_m, ϵ in C' (ϵ is the empty word). An inductive argument shows that C' simulates the computation of C, with only a factor 2 slowdown.

4.1. $U_{4,5}$

The table of behaviour for $U_{4,5}$ is given in Table 4.1.

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 $0\lambda\mathit{1}0$, and 1 is encoded as $00\lambda\mathit{1}$. The encoded α_R is then prepended with the symbol λ. For example, if $\alpha = 100$ then this appendant is encoded as $\langle\alpha\rangle = \lambda 0\lambda\mathit{1}00\lambda\mathit{1}000\lambda\mathit{1}$. 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,

	u_1	u_2	u_3	u_4
0	$\lambda \mathrm{L} u_1$	$\lambda \mathrm{L} u_2$	$B \mathrm{R} u_3$	$B \mathrm{R} u_4$
1	$B \mathrm{R} u_2$	$1 \mathrm{L} u_2$	$1 \mathrm{R} u_3$	$1 \mathrm{R} u_4$
λ	$0 \mathrm{R} u_2$	$0 \mathrm{R} u_1$	$0 \mathrm{R} u_4$	$0 \mathrm{R} u_3$
B		$0 \mathrm{L} u_2$	$0 \mathrm{L} u_2$	$1 \mathrm{L} u_2$
1	$0 \mathrm{L} u_2$	$\lambda \mathrm{L} u_3$		

Table 4.1. Table of behaviour for $U_{4,5}$. The start state is u_1 and the blank symbol is B.

to the left, on the tape of $U_{4,5}$. The start state for $U_{4,5}$ is u_1, the blank symbol is B, 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 2.1 is encoded as

$$\boldsymbol{u_1},\ \ldots \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1} 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 00 \lambda \mathit{1} 0\ \underline{0}010010\ BBB \ldots \tag{2}$$

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 2.1. Example 4.1 shows $U_{4,5}$ reading two 0 symbols in the dataword and indexing appendants. Example 4.2 shows $U_{4,5}$ reading a 10 in the dataword, printing one appendant and indexing the next. Lemmata 4.1 and 4.2 build on these examples to give a proof of correctness.

Example 4.1. ($U_{4,5}$; 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. After 3 steps:

$$\begin{aligned}
\vdash^3 \quad & \boldsymbol{u_2},\ \ldots \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1} 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 00 \underline{\lambda} 0 \lambda\ \lambda 010010\ BBB \ldots \\
\vdash \quad & \boldsymbol{u_1},\ \ldots \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1} 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 000 \underline{0} \lambda\ \lambda 010010\ BBB \ldots \\
\vdash^4 \quad & \boldsymbol{u_1},\ \ldots \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1} 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \underline{\mathit{1}} \lambda \lambda \lambda \lambda \lambda\ \lambda 010010\ BBB \ldots
\end{aligned}$$

We continue like this, until we read $\lambda 0$ (left hand side of the first appendant), to give:

$$\vdash^5 \quad \boldsymbol{u_1},\ \ldots \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1} 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \underline{\lambda} \lambda \lambda \lambda \lambda \lambda \lambda \lambda \lambda\ \lambda 010010\ BBB \ldots \tag{3}$$

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.

$$\vdash^{10} \quad \boldsymbol{u_1},\ \ldots \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ \lambda 0 \lambda \mathit{1} 000 \lambda \mathit{1} 0 \lambda \mathit{1} 000 \lambda \mathit{1}\ 000000000\ 0\underline{0}10010\ BBB \ldots$$

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 in 35 steps to give:

$$\vdash^{35} \quad \boldsymbol{u_1},\ \ldots\lambda 0\lambda\boldsymbol{1}000\lambda\boldsymbol{1}\ \underline{\lambda}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda 10010\ BBB\ldots$$

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 28 steps later in state u_1 (with shorthand notation for consecutive 0s):

$$\vdash^{28} \quad \boldsymbol{u_1},\ \ldots\lambda 0\lambda\boldsymbol{1}000\lambda\boldsymbol{1}\ 0^{17}\ 0^{9}\ 00\underline{1}0010\ BBB\ldots \tag{4}$$

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 subsequent appendant.

Example 4.2. ($U_{4,5}$; 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.

$$\begin{aligned}
\vdash \quad & \boldsymbol{u_2},\ \ldots\lambda 0\lambda\boldsymbol{1}000\lambda\boldsymbol{1}\ 0^{17}\ 0^{9}\ 00B\underline{0}010\ BBB\ldots\\
\vdash \quad & \boldsymbol{u_2},\ \ldots\lambda 0\lambda\boldsymbol{1}000\lambda\boldsymbol{1}\ 0^{17}\ 0^{9}\ 00\underline{B}\lambda 010\ BBB\ldots\\
\vdash \quad & \boldsymbol{u_2},\ \ldots\lambda 0\lambda\boldsymbol{1}000\lambda\boldsymbol{1}\ 0^{17}\ 0^{9}\ 0\underline{0}0\lambda 010\ BBB\ldots\\
\vdash^{27} \quad & \boldsymbol{u_2},\ \ldots\lambda 0\lambda\boldsymbol{1}000\lambda\boldsymbol{1}\ \underline{0}\lambda^{16}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ BBB\ldots
\end{aligned}$$

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

$$\vdash \quad \boldsymbol{u_2},\ \ldots\lambda 0\lambda\boldsymbol{1}000\lambda\underline{\boldsymbol{1}}\ \lambda^{17}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ BBB\ldots$$

This encoded appendant tells us that the symbol 1 (encoded as $00\lambda\boldsymbol{1}$), and then the symbol 0 (encoded as $0\lambda\boldsymbol{1}0$), should be appended to the dataword.

$$\vdash \quad \boldsymbol{u_3},\ \ldots\lambda 0\lambda\boldsymbol{1}000\underline{\lambda}\lambda\ \lambda^{17}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ BBB\ldots$$

$U_{4,5}$ now scans right, switching between state u_3 and u_4, eventually appending either 0 or 1 to the dataword. In the above configuration, 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.

$$\begin{aligned} \vdash \quad & \boldsymbol{u_4},\ \ldots\lambda 0\lambda\mathbf{1}0000\underline{\lambda}\ \lambda^{17}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ BBB\ldots \\ \vdash \quad & \boldsymbol{u_3},\ \ldots\lambda 0\lambda\mathbf{1}00000\ \underline{\lambda}\lambda^{16}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ BBB\ldots \\ \vdash^{30} \quad & \boldsymbol{u_4},\ \ldots\lambda 0\lambda\mathbf{1}00000\ 0^{17}\ 0^{9}\ 00B0\underline{0}10\ BBB\ldots \end{aligned}$$

We now pass over the dataword and append a 1.

$$\begin{aligned} \vdash^{3} \quad & \boldsymbol{u_4},\ \ldots\lambda 0\lambda\mathbf{1}00000\ 0^{17}\ 0^{9}\ 00B0B1B\ \underline{B}BBB\ldots \\ \vdash \quad & \boldsymbol{u_2},\ \ldots\lambda 0\lambda\mathbf{1}00000\ 0^{17}\ 0^{9}\ 00B0B1\underline{B}\ 1BBB\ldots \end{aligned}$$

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

$$\vdash^{37} \quad \boldsymbol{u_2},\ \ldots\lambda 0\lambda\mathbf{1}\underline{0}\lambda\lambda\lambda\lambda\ \lambda^{17}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ 1BBB\ldots$$

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

$$\vdash^{2} \quad \boldsymbol{u_3},\ \ldots\lambda 0\underline{\lambda}\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda^{17}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ 1BBB\ldots$$

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.

$$\begin{aligned} \vdash^{41} \quad & \boldsymbol{u_3},\ \ldots\lambda 00000000\ 0^{17}\ 0^{9}\ 00B0B1B\ 1\underline{B}BBB\ldots \\ \vdash \quad & \boldsymbol{u_2},\ \ldots\lambda 00000000\ 0^{17}\ 0^{9}\ 00B0B1B\ \underline{1}0BBB\ldots \end{aligned}$$

$U_{4,5}$ 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.

$$\vdash^{42} \quad \boldsymbol{u_2},\ \ldots\underline{\lambda}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda^{17}\ \lambda^{9}\ \lambda\lambda 0\lambda 010\ 10BBB\ldots$$

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

$$\vdash^{37} \quad \boldsymbol{u_1},\ \ldots 000000000\ 0^{17}\ 0^{9}\ 00\underline{0}\lambda 010\ 10BBB\ldots$$

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 now duplicate the configuration immediately above (showing the next two encoded appendants to the left).

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

As already noted, we are forced to index (and erase) the next appendant:

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

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:

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

The example is complete.

The conditions under which $U_{4,5}$ completes its computation are the same as those for $U_{3,7}$; if the cyclic tag systems halts, then $U_{4,5}$ reads a blank symbol B 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 4.1. 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 $\langle 0\rangle = 0\lambda\mathbf{1}0$ or $\langle 1\rangle = 00\lambda\mathbf{1}$ 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. □

Lemma 4.2. 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 B, the head moves right and erases an extra 0 symbol, and then moves left. This B 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 $\langle 0\rangle = 0\lambda\mathbf{1}0$ from the right in state u_2, we then (i) exit to the right in state u_4. However if we enter $\langle 1\rangle = 00\lambda\mathbf{1}$ 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 4.1). 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. □

5. 2-state, 13-symbol machine

$U_{2,13}$ uses a similar algorithm to $U_{3,7}$. 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. As with the other machines, $U_{2,13}$ 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 p appendants. The machine halts using the same method as the previous two machines (i.e. after reading the blank symbol B in state u_1).

5.1. $U_{2,13}$

The table of behaviour for $U_{2,13}$ is given in Table 5.1.

5.2. Encoding

For our 2-state, 13-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 $\emptyset\mathit{1}$, and 1 is encoded as $\mathit{11}$. The encoded α_R is then prepended with the two symbols $\lambda\emptyset$. For example, if $\alpha = 100$ then this appendant is encoded as $\langle\alpha\rangle = \lambda\emptyset\mathit{1}\emptyset\mathit{1}\mathit{11}$. 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 $U_{2,13}$. The start state for $U_{2,13}$ is u_1, the blank symbol is B, and the cyclic tag system input is written directly on the tape of $U_{2,13}$. Thus the initial configuration of the cyclic tag system given in Example 2.1 is encoded as

$$\boldsymbol{u_1},\ \ldots\lambda\emptyset\mathit{11}\ \lambda\emptyset\mathit{1}\emptyset\mathit{111}\ \lambda\emptyset\mathit{1}\emptyset\mathit{1}\ \underline{0}010010\ BBB\ldots \tag{5}$$

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.

5.3. Simulation

We simulate the first three steps of the cyclic tag computation from Example 2.1.

	u_1	u_2
0	$\lambda \mathrm{L} u_1$	$0_r \mathrm{R} u_2$
1	$b \mathrm{R} u_2$	$1_r \mathrm{R} u_2$
B		$B_d \mathrm{L} u_2$
B_d	$B \mathrm{L} u_1$	$B_w \mathrm{L} u_2$
B_w	$0_r \mathrm{R} u_2$	$1_r \mathrm{R} u_2$
λ	$b \mathrm{R} u_1$	$b \mathrm{R} u_2$
b	$\lambda \mathrm{L} u_1$	$\lambda \mathrm{L} u_2$
$\emptyset$	$\lambda \mathrm{L} u_1$	$b \mathrm{R} u_1$
$\mathit{1}$	$\lambda \mathrm{L} u_1$	$b \mathrm{R} u_2$
0_ℓ	$0_r \mathrm{R} u_1$	$0_r \mathrm{R} u_2$
1_ℓ	$1_r \mathrm{R} u_1$	$1_r \mathrm{R} u_2$
0_r	$0 \mathrm{L} u_1$	$0_\ell \mathrm{L} u_2$
1_r	$1 \mathrm{L} u_1$	$1_\ell \mathrm{L} u_2$

Table 5.1. Table of behaviour for $U_{2,13}$. The start state is u_1 and the blank symbol is B.

Example 5.1. Beginning with the configuration given in Equation (5), $U_{2,13}$ 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 the left hand side of the encoded appendant, to give the following configuration 6 steps later

$$\vdash^6 \quad \boldsymbol{u_1}, \quad \ldots \lambda \emptyset\mathit{111}\ \lambda \emptyset\mathit{1110111}\ \underline{\lambda}\lambda\lambda\lambda\lambda\ \lambda 010010\ BBB \ldots$$

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.

$$\vdash^7 \quad \boldsymbol{u_1}, \quad \ldots \lambda \emptyset\mathit{111}\ \lambda \emptyset\mathit{1110111}\ \mathit{bbbbbb}\ b\underline{0}10010\ BBB \ldots$$

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

$$\vdash^{17} \quad \boldsymbol{u_1}, \quad \ldots \lambda \emptyset\mathit{111}\ \underline{\lambda}\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda 10010\ BBB \ldots$$

and then return to the third input symbol.

$$\vdash^{18} \quad \boldsymbol{u_1}, \quad \ldots \lambda \emptyset\mathit{111}\ \mathit{bbbbbbbbbb}\ \ \mathit{bbbbbb}\ bb\underline{1}0010\ BBB \ldots$$

The input symbol 1 causes $U_{2,13}$ to scan right and enter a 'print cycle' which iterates the following. In state u_2 we replace the leftmost B with B_d, which signifies that we are currently deciding whether or not to append a symbol to the dataword. We then scan left in u_2, if we read $\mathit{1}$ then we scan right in u_2 and replace B_d with B_w, which signifies that we are committed to appending a symbol. We then scan left in u_2, if we find $\emptyset$ then we go right in u_1 and print an encoded 0, otherwise if we find $\mathit{1}$ then we go

right in u_2 and print an encoded 1. We then iterate this procedure. If at any point we read $\lambda\emptyset$ then we exit the cycle (and restore the B_d symbol to B). We now contine our simulation to the point where we have changed the blank symbol to B_d.

$$\vdash^6 \quad \boldsymbol{u_2},\ \ldots \lambda\emptyset\emptyset 111\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0_r 0_r 1_r \underline{0_r}\ B_d BBB \ldots$$

We move left in u_2 to begin reading the encoded third appendant.

$$\vdash^{23} \quad \boldsymbol{u_2},\ \ldots \lambda\emptyset\emptyset 11\underline{1}\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0_\ell 0_\ell 1_\ell 0_\ell\ B_d BBB \ldots$$

An encoded appendant, that does not represent the empty word, always begins with 1 and this causes $U_{2,13}$ to scan right, change B_d to B_w,

$$\vdash^{25} \quad \boldsymbol{u_2},\ \ldots \lambda\emptyset\emptyset 11b\ bbbbbbbbbb\ bbbbbb\ bbb 0_r 0_r 1_r \underline{0_r}\ B_w BBB \ldots$$

and then return left to read the encoded symbol.

$$\vdash^{24} \quad \boldsymbol{u_2},\ \ldots \lambda\emptyset\emptyset 1\underline{1}\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0_\ell 0_\ell 1_\ell 0_\ell\ B_w BBB \ldots$$

As before, reading a 1 causes a scan right in u_2, however this time we encouter a B_w which gets overwritten with 1_r (an encoded 1).

$$\vdash^{26} \quad \boldsymbol{u_2},\ \ldots \lambda\emptyset\emptyset 1bb\ bbbbbbbbbb\ bbbbbb\ bbb 0_r 0_r 1_r 0_r\ 1_r \underline{B} BB \ldots$$

We write a new B_d symbol and scan left, to search for the next encoded symbol in the appendant.

$$\vdash^{27} \quad \boldsymbol{u_2},\ \ldots \lambda\emptyset\emptyset\underline{1}\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0_\ell 0_\ell 1_\ell 0_\ell\ 1_\ell B_d BBB \ldots$$

As before, reading a 1 triggers a scan right. We commit to appending a symbol by changing B_d to B_w, and then return to read the encoded symbol to append.

$$\vdash^{55} \quad \boldsymbol{u_2},\ \ldots \lambda\emptyset\underline{\emptyset}\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0_\ell 0_\ell 1_\ell 0_\ell\ 1_\ell B_w BBB \ldots$$

We are reading an encoded 0, this causes a scan right in u_1. This is the first time we have used u_1 in the print cycle, and reading B_w causes an encoded 0 to be appended to the dataword.

$$\begin{aligned} \vdash^{28} \quad & \boldsymbol{u_1},\ \ldots \lambda\emptyset bbbb\ bbbbbbbbbb\ bbbbbb\ bbb 0_r 0_r 1_r 0_r\ 1_r \underline{B_w} BBB \ldots \\ \vdash^{1} \quad & \boldsymbol{u_2},\ \ldots \lambda\emptyset bbbb\ bbbbbbbbbb\ bbbbbb\ bbb 0_r 0_r 1_r 0_r\ 1_r 0_r \underline{B} BB \ldots \end{aligned}$$

We write a new B_d symbol and scan left, to search for the next encoded symbol in the appendant.

$$\vdash^{30} \quad \boldsymbol{u_2},\ \ldots \lambda\underline{\emptyset}\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda\lambda\lambda\lambda\ \lambda\lambda\lambda 0_\ell 0_\ell 1_\ell 0_\ell\ 1_\ell 0_\ell B_d BBB \ldots$$

However, instead we read a $\emptyset$, signifying the end of the appendant and this triggers a right scan in u_1. At the end of this scan we meet B_d, and in this case we are not appending any more symbols, so we overwrite with B.

$$\vdash^{31} \quad \boldsymbol{u_1},\ \ldots \lambda bbbbb\ bbbbbbbbbb\ bbbbbb\ bbb 0_r 0_r 1_r 0_r\ 1_r \underline{0_r} BBBB \ldots$$

We then scan left in u_1, restoring the dataword.

$$\vdash^{30} \quad \boldsymbol{u_1}, \quad \ldots \underline{\lambda}\lambda\lambda\lambda\lambda\lambda \; \lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda\lambda \; \lambda\lambda\lambda\lambda\lambda\lambda \; \lambda\lambda\lambda 0010 \; 10BBBB \ldots$$

Reading λ in u_1 sends us to the right, ready to begin the next index cycle.

$$\vdash^{25} \quad \boldsymbol{u_1}, \quad \ldots \lambda 00111 \; \lambda 001110111 \; \lambda 00101 \; bbbbbb \; bbbbbbbbbb \; bbbbbb \; bbb\underline{0}010 \; 10BBBB \ldots$$

The latter configuration shows the next set of encoded appendants to the left. At this point we have simulated the third computation step of the cyclic tag system in Example 2.1. This completes Example 5.1.

As can be seen in the preceding example, the computation of $U_{2,13}$ 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. The first condition (empty dataword) is simulated by $U_{2,13}$ in a very straightforward way: if the dataword is empty then $U_{2,13}$ reads a blank symbol B in state u_1, and immediately halts. The second condition (repeating sequence of cyclic tag configurations) causes $U_{2,13}$ to simulate this loop in an easily detectable way, where some fixed sequence of appendants are repeatedly appended to the dataword.

Acknowledgement

We thank Maurice Margenstern for comments that helped to improve the presentation.

References

[1] C. Baiocchi. 3N+1, UTM e tag-system. Technical Report Pubblicazione 98/38, Dipartimento di Matematico, Università di Roma, 1998. (In Italian).

[2] 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.

[3] 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.

[4] M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

[5] 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.

[6] M. Kudlek. Small deterministic Turing machines. *Theoretical Computer Science*, 168(2):241–255, 1996.

[7] 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.

[8] M. Margenstern. Frontier between decidability and undecidability: a survey. *Theoretical Computer Science*, 231(2):217–251, Jan. 2000.

[9] 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.

[10] P. Michel. Small Turing machines and generalized busy beaver competition. *Theoretical Computer Science*, 326:45–56, Oct. 2004.

[11] M. Minsky. A 6-symbol 7-state universal Turing machines. Technical Report 54-G-027, MIT, Aug. 1960.

[12] 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.

[13] 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.

[14] T. Neary. *Small universal Turing machines*. PhD thesis, National University of Ireland, Maynooth, 2008.

[15] T. Neary and D. Woods. Four small universal Turing machines. *Fundamenta Informaticae*. 91:105–126. 2009

[16] T. Neary and D. Woods. P-completeness of cellular automaton Rule 110. In M. Bugliesi et al., editor, *International Colloquium on Automata Languages and Programming (ICALP)*, volume 4051 (Part I) of *Lecture Notes in Computer Science*, pages 132–143. Springer, July 2006.

[17] T. Neary and D. Woods. Small fast universal Turing machines. *Theoretical Computer Science*, 362(1–3):171–195, Oct. 2006.

[18] T. Neary and D. Woods. Four small universal Turing machines. In J. Durand-Lose and M. Margenstern, editors, *Machines, Computations and Universality (MCU)*, volume 4664 of *LNCS*, pages 242–254, Orléans, France, Sept. 2007. Springer.

[19] T. Neary and D. Woods. Small weakly universal Turing machines, July 2007. arXiv:0707.4489v1 [cs.CC].

[20] 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. 899–909, June, 1973).

[21] L. Pavlotskaya. Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring. *Problemy Kibernetiki*, 33:91–118, 1978. (Sufficient conditions for the halting problem decidability of Turing machines. In Russian).

[22] L. Pavlotskaya. On machines, universal by extensions. *Theoretical Computer Science*, 168(2):257–266, Nov. 1996.

[23] L. Priese. Towards a precise characterization of the complexity of universal and nonuniversal Turing machines. *SIAM Journal on Computing*, 8(4):508–523, 1979.

[24] R. M. Robinson. Minsky's small universal Turing machine. *International Journal of Mathematics*, 2(5):551–562, 1991.

[25] Y. Rogozhin. Sem' universal'nykh mashin T'juringa. *Systems and theoretical programming, Mat. Issled.*, 69:76–90, 1982. (Seven universal Turing machines. In Russian).

[26] Y. Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, Nov. 1996.

[27] C. E. Shannon. A universal Turing machine with two internal states. *Automata Studies, Annals of Mathematics Studies*, 34:157–165, 1956.

[28] S. Watanabe. On a minimal universal Turing machine. Technical report, MCB Report, Tokyo, Aug. 1960.

[29] S. Watanabe. 5-symbol 8-state and 5-symbol 6-state universal Turing machines. *Journal of the ACM*, 8(4):476–483, Oct. 1961.

[30] S. Watanabe. 4-symbol 5-state universal Turing machine. *Information Processing Society of Japan Magazine*, 13(9):588–592, Sept. 1972.

[31] S. Wolfram. *A new kind of science*. Wolfram Media, Inc., 2002.

[32] D. Woods and T. Neary. The complexity of small universal Turing machines. *Theoretical computer Science*, 410(4-5):443–450, Feb. 2009.

[33] 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.

[34] D. Woods and T. Neary. Small semi-weakly universal Turing machines. In J. Durand-Lose and M. Margenstern, editors, *Machines, Computations and Universality (MCU)*, volume 4664 of *LNCS*, pages 303–315, Orléans, France, Sept. 2007. Springer.