Fundamenta Informaticae 91 (2009) 123–144
DOI 10.3233/FI-2009-0036
IOS Press

Four Small Universal Turing Machines

Turlough Neary[*†]

Boole Centre for Research in Informatics, University College Cork, Ireland.

tneary@cs.nuim.ie

Damien Woods[‡]

Department of Computer Science and Artificial Intelligence, University of Seville, Spain.

d.woods@cs.ucc.ie

Abstract. We present universal Turing machines with state-symbol pairs of $(5, 5)$, $(6, 4)$, $(9, 3)$ and $(15, 2)$. These machines simulate our new variant of tag system, the bi-tag system and are the smallest known single-tape universal Turing machines with 5, 4, 3 and 2-symbols, respectively. Our 5-symbol machine uses the same number of instructions (22) as the smallest known universal Turing machine by Rogozhin. Also, all of the universal machines we present here simulate Turing machines in polynomial time.

Keywords: small universal Turing machine, 2-tag system, bi-tag systems, Post system, computational complexity, polynomial time.

1. Introduction

Shannon [24] was the first to discuss the problem of finding the smallest possible universal Turing machine. In 1962 Minsky [11] constructed a 7-state, 4-symbol universal Turing machine that simulates Turing machines via 2-tag systems [2]. Minsky's technique of 2-tag simulation was extended by Rogozhin [23] to construct small universal Turing machines with state-symbol pairs of $(24, 2)$, $(10, 3)$,

[*]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.
[†]Address for correspondence: Boole Centre for Research in Informatics, University College Cork, Ireland.
[‡]Damien Woods is supported by Junta de Andalucía grant TIC-581.

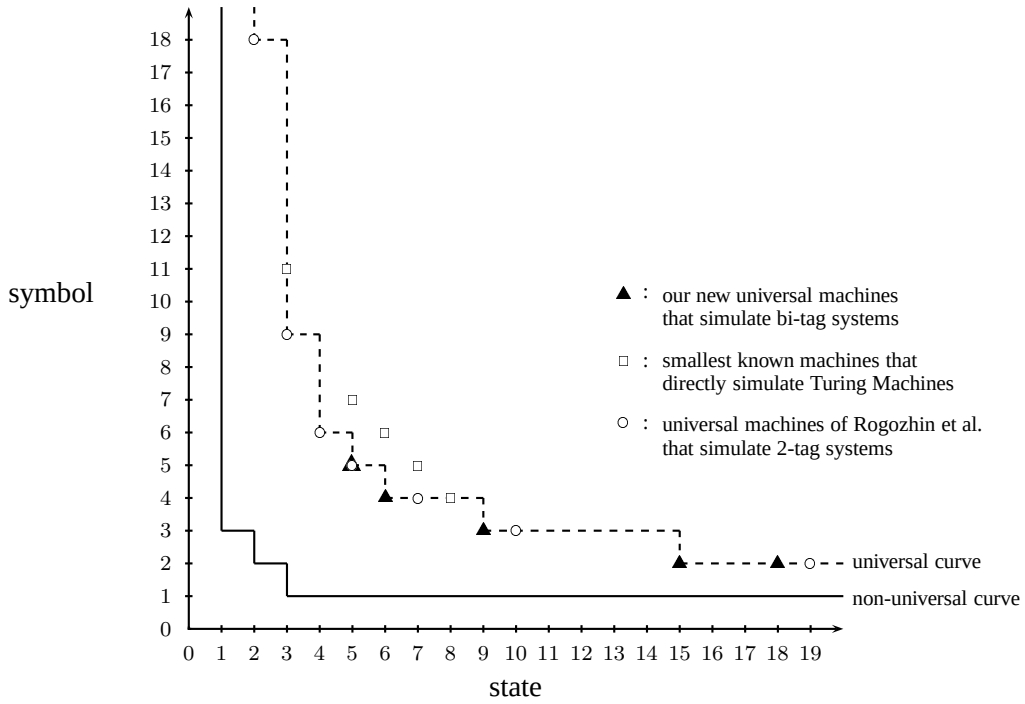


Figure 1: Current state-symbol plot of small universal Turing machines. The non-universal curve shows Turing machines that are known to have a decidable halting problem.

$(7, 4)$, $(5, 5)$, $(4, 6)$, $(3, 10)$ and $(2, 18)$. Subsequently some of these machines were reduced in size to give machines with state-symbol pairs of $(3, 9)$ [7], $(19, 2)$ [1] and $(7, 4)$ [1]. Figure 1 is a state-symbol plot where the current smallest 2-tag simulators of Rogozhin et al. are plotted as circles.

Here we present universal Turing machines with state-symbol pairs of $(5, 5)$, $(6, 4)$, $(9, 3)$ and $(15, 2)$. The 5, 4, and 3-symbol machines have previously appeared in [16]. The new 15-state, 2-symbol machine we present here is a significant improvement on the 18-state, 2-symbol machine that appeared in [16]. All of these machines simulate Turing machines via bi-tag systems and are plotted as triangles in Figure 1. These machines improve the state of the art in small universal Turing machines and reduce the space between the universal and non-universal curves. Our 5-symbol machine uses the same number of instructions (22) as the current smallest known universal Turing machine (Rogozhin's 6-symbol machine [23]). Also, our 5-symbol machine has less instructions than Rogozhin's 5-symbol machine. Since Minsky [11] constructed his 7-state, 4-symbol machine, a number of authors [1, 21, 23] have given 4-symbol machines. Rogozhin [23] improved on Minsky's result by giving a 7-state, 4-symbol machine with 26 instructions and Baiocchi [1] further improved on this result to give a 7-state, 4-symbol machine with 25 instructions. Our 4-symbol machine is the first reduction in the number of states since Minsky's machine. In fact, in 1991 Robinson [21] noted that when considering the numbers of states and symbols of the machines constructed since Minsky's machine "there is no known such machine which decreases one parameter without increasing the other." It is interesting to note that the current universal curve in Figure 1 is no longer symmetric about the line where the number of states is equal to the number of symbols. (For a brief period, the universal curve was symmetric following the work in [16].)

Recently, the simulation time overhead of Turing machines by 2-tag systems was improved from exponential [2] to polynomial [30]. More precisely, if M is a single tape deterministic Turing machine that runs in time t, then the universal Turing machines of Minsky and Rogozhin et al. now simulate M in $O(t^8(\log t)^4)$ time. It turns out that the time overhead can be improved to $O(t^4(\log t)^2)$ [13]. In earlier work [15] we gave the smallest known universal machines that directly simulate Turing machines. These machines run in time $O(t^2)$ and are plotted as squares in Figure 1. Our new universal Turing machines are polynomial time simulators of Turing machines. Specifically, our new machines simulate one-tape deterministic Turing machines, with a time overhead of $O(t^6)$: they simulate bi-tag systems (quadratic time overhead), which in turn simulate one tape deterministic Turing machines (cubic time overhead).

The halting problem has been shown to be decidable for the following state-symbol pairs: $(2, 2)$ [5, 18], $(3, 2)$ [19], $(2, 3)$ (Pavlotskaya, unpublished), $(1, n)$ [4], and $(n, 1)$ (trivial) for $n \geqslant 1$. Thus, these results induce the non-universal curve which is illustrated in Figure 1. More on small universal Turing machines, and related notions, can be found in [8, 9, 13, 29, 28].

In Section 2 we show that bi-tag systems simulate Turing machines. We begin by introducing the clockwise Turing machine, and then prove that it simulates Turing machines. Following this we introduce bi-tag systems and prove that they simulate clockwise Turing machines. Section 3 begins with the input encodings to each of the universal Turing machines. This is followed by an overview of the simulation algorithm used by our machines. Then, each of the universal Turing machines are given along with a more detailed look at their operation. The final part of the paper, Section 4, contains some discussion and conclusions. This paper is an extended version of the paper that appeared in [16], it contains new results, extra proofs and discussion.

1.1. Preliminaries

The Turing machines considered in this paper are deterministic and have one tape. Our universal Turing machine with m states and n symbols is denoted $U_{m,n}$. We write $c_1 \vdash c_2$ if a configuration c_2 is obtained from c_1 via a single computation step. We let $c_1 \vdash^t c_2$ denote a sequence of t computation steps and let $c_1 \vdash^* c_2$ denote 0 or more computation steps. Also, we let $\langle x \rangle$ denote the encoding of object x and ϵ denote the empty word.

2. Bi-tag systems simulate Turing machines

2.1. Clockwise Turing machines simulate Turing machines

A clockwise Turing machine is a Turing machine that has a single tape, which is circular, and whose tape head moves only in a clockwise direction. The operation of clockwise Turing machines is quite similar to that of the circular Post machines of Kudlek and Rogozhin [6].

Definition 2.1. (Clockwise Turing machine [14])
A clockwise Turing machine is a tuple $C = (Q, \Sigma, f, q_1, q_{|Q|})$. Q and Σ are the finite sets of states and tape symbols, respectively. $q_1 \in Q$ is the start state and $q_{|Q|} \in Q$ is the halt state. The transition function $f : Q \times \Sigma \to \{\Sigma \cup \Sigma\Sigma\} \times Q$ is undefined on state $q_{|Q|}$ and is defined for all $q \in Q$, $q \neq q_{|Q|}$.

We write f as a list of clockwise transition rules. Each clockwise transition rule is a quadruple $t = (q_x, \sigma_1, v, q_y)$, with initial state q_x, read symbol σ_1, write value $v \in \{\Sigma \cup \Sigma\Sigma\}$ and next state q_y.

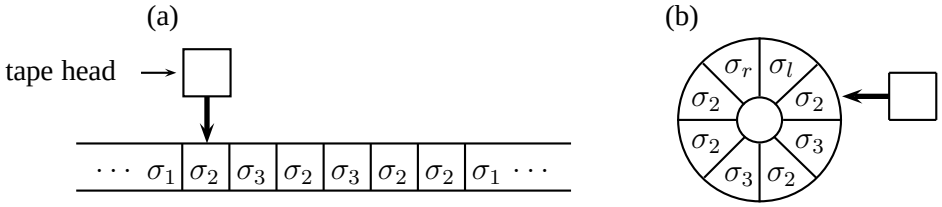


Figure 2: (a) Example Turing machine tape contents. The Turing machine's blank symbol is σ_1. (b) Clockwise Turing machine encoding of the Turing machine tape contents in (a), the symbols σ_r and σ_l encode the infinite sequence of blank symbols to the right and left of M's encoded tape contents.

A clockwise transition rule is executed as follows: If the write value v is from Σ then the tape cell containing the read symbol is overwritten by v, if v is from $\Sigma\Sigma$ then the cell containing the read symbol becomes two cells, each of which contain a symbol from v. The machine's state becomes q_y and the tape head moves clockwise by one tape cell. Here we define clockwise Turing machines to be deterministic.

Lemma 2.1. Let M be a deterministic Turing machine with a single tape that computes in time t, then there is clockwise Turing machine C_M that simulates the computation M in time $O(t^2)$ and space $O(t)$.

Proof:
Let $M = (\{q_1, \ldots, q_{|Q|}\}, \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}, \sigma_1, f, q_1, \{q_{|Q|}\})$. Without loss of generality we can assume that M is a Turing machine that has the following restrictions: (i) the blank symbol σ_1 does not appear as input to M, (ii) M may read the blank symbol σ_1 but is not permitted to write it to the tape, (iii) M has exactly one final state. Due to the restrictions placed on M we know that when M reads a blank symbol it is either at the left or right end of its tape contents. We construct a clockwise Turing machine $C_M = (Q_C, \Sigma_C, f_C, q_1, q_{|Q|})$ that simulates M, where Q_C, Σ_C, f_C are defined below.

$$\Sigma_C = \{\sigma_2, \ldots, \sigma_{|\Sigma|}, \sigma_r, \sigma_l, \sigma_m\}$$

The symbol σ_m is a special marker symbol and symbols σ_r and σ_l encode the infinite sequence of blank symbols to the right and left of M's encoded tape contents, respectively (see Figure 2).

$$\begin{aligned} Q_C = \{&q_1, q_{1,2}, \ldots, q_{1,|\Sigma|}, q_{1,r}, q_{1,r'}, q_{1,l}, \\ &q_2, q_{2,2}, \ldots, q_{2,|\Sigma|}, q_{2,r}, q_{2,r'}, q_{2,l}, \\ &\vdots \\ &q_{|Q|}, q_{|Q|,2}, \ldots, q_{|Q|,|\Sigma|}, q_{|Q|,r}, q_{|Q|,r'}, q_{|Q|,l}\} \end{aligned}$$

We can think of right moves of M's tape head as clockwise moves of C_M's tape head. Here we give right move transition rules followed by the clockwise transition rules that simulate them.

$$q_x, \sigma_k, \sigma_j, R, q_y \quad : \quad q_x, \sigma_k, \sigma_j, q_y \tag{1}$$

$$q_x, \sigma_1, \sigma_j, R, q_y \quad : \quad q_x, \sigma_l, \sigma_l\sigma_j, q_y \tag{2}$$

where $\sigma_k, \sigma_j \neq \sigma_1$. The clockwise transition rule in Equation (2) simulates M printing the write symbol σ_j over the blank symbol immediately to the left of its tape contents. The clockwise transition rule's write value $\sigma_l\sigma_j \in \Sigma\Sigma$ also preserves σ_l; the symbol that encodes the infinite sequence of blank symbols to the left of the tape contents.

The remaining right moving case is when M's tape head is over the blank symbol immediately to the right of its tape contents. In such a case C_M's tape head is initially over σ_r, and then immediately after simulation of the transition rule, C_M's tape head is again over σ_r. Immediately below are the clockwise transition rules that simulate this case.

$$q_x, \sigma_1, \sigma_j, R, q_y \quad : \quad \begin{array}{ll} q_x, \sigma_r, \sigma_j\sigma_r, q_{y,r'} & (*) \\ q_{y,r'}, \sigma_i, \sigma_i, q_{y,r'} & (**) \end{array}$$

where $\sigma_i \in \Sigma_C - \{\sigma_m, \sigma_r\}$. The clockwise transition rule (*) prints M's encoded write symbol σ_j and sends C_M's control into state $q_{y,r'}$. State $q_{y,r'}$ moves C_M's tape head around the tape to the cell containing σ_r. This completes the simulation of the transition rule.

Left moving transition rules are more difficult to simulate as C_M's tape head moves only clockwise. C_M begins by marking the current location of the tape head with the symbol σ_m. C_M now moves each symbol clockwise by one cell. When C_M's tape head reads σ_m the left move is complete. This process moves the tape head anti-clockwise relative to the tape contents, thus simulating a left move. Immediately below is given the clockwise transition rules that mark the tape head's location with the symbol σ_m.

$$\begin{array}{lcl} q_x, \sigma_1, \sigma_j, L, q_y & : & q_x, \sigma_l, \sigma_l\sigma_m, q_{y,j} \\ q_x, \sigma_1, \sigma_j, L, q_y & : & q_x, \sigma_r, \sigma_m\sigma_j, q_{y,r} \\ q_x, \sigma_k, \sigma_j, L, q_y & : & q_x, \sigma_k, \sigma_m, q_{y,j} \end{array}$$

The clockwise transition rules that move each symbol clockwise by one cell are of the form:

$$q_{y,n}, \sigma_s, \sigma_n, q_{y,s}$$

where $\sigma_s, \sigma_n \in \Sigma_C - \{\sigma_m\}$. When C_M's tape head reads σ_m then C_M is in a state of the form $q_{y,s}$ and the unique clockwise transition rule defined by the state-symbol pair $(q_{y,s}, \sigma_m)$ will begin simulation of the next transition rule. This transition rule is of the form $(q_y, \sigma_1, \sigma_k, D, q_z)$ if $\sigma_s = \sigma_r, \sigma_l$ and of the form $(q_y, \sigma_s, \sigma_k, D, q_z)$ if $\sigma_s \neq \sigma_r, \sigma_l$.

Input to M is encoded for C_M by a finite state transducer. Given this encoded input C_M simulates the sequence of t transition rules in M's computation and halts in state $q_{|Q|}$ the encoding of M's halt state $q_{|Q|}$. C_M uses space of $O(t)$. A single computation step of M is simulated in $O(t)$ steps of C_M. Thus the computation time of C_M is $O(t^2)$. □

2.2. Bi-tag systems simulate clockwise Turing machines

In this section we present the bi-tag system, our new variant on the tag system, and prove that it simulates Turing machines via clockwise Turing machines. The operation of a bi-tag system is similar to that of a standard tag system [12]. Bi-tag systems are essentially 1-tag systems (and so they read and delete one symbol per timestep), augmented with additional context sensitive rules that read, and delete, two symbols per timestep.

Definition 2.2. (Bi-tag system)
A bi-tag system is a tuple (A, E, e_h, P). Here A and E are disjoint finite sets of symbols and $e_h \in E$ is the halt symbol. P is the finite set of productions. Each production is of one of the following 3 forms:

$$P(a) = a, \quad P(e, a) \in AE, \quad P(e, a) \in AAE,$$

where $a \in A$, $e \in E$, and P is defined on all elements of $\{A \cup ((E - \{e_h\}) \times A)\}$ and undefined on all elements of $\{e_h\} \times A$. Bi-tag systems are deterministic.

A configuration of a bi-tag system is a word of the form $w = A^*(AE \cup EA)A^*$. We call w the dataword.

Definition 2.3. (BTS computation step)
A production is applied in one of two ways:

(i) if $s = as'$ then $as' \vdash s'P(a)$,

(ii) if $s = eas'$ then $eas' \vdash s'P(e, a)$.

A bi-tag system computation is a finite sequence of computation steps that are consecutively applied to an initial dataword. If e_h is the leftmost symbol in the dataword then the computation halts.

Example 2.1. *(Bi-tag system computation.)* Let bi-tag system $B_1 = (\{a_0, a_1\}, \{e_0, e_1, e_2\}, e_2, P)$ where the set $P = \{a_0 \to a_0,\ a_1 \to a_1,\ e_0a_0 \to a_1e_0,\ e_0a_1 \to a_1e_2,\ e_1a_0 \to a_0e_0,\ e_1a_1 \to a_1e_2\}$. Given the word $a_1e_0a_0$, the computation of B_1 proceeds as follows:

$$a_1e_0a_0 \vdash e_0a_0a_1 \vdash a_1a_1e_0 \vdash a_1e_0a_1 \vdash e_0a_1a_1 \vdash a_1a_1e_2 \vdash a_1e_2a_1 \vdash e_2a_1a_1$$

The computation halts as the halt symbol e_2 has become the leftmost symbol.

Lemma 2.2. Let C be a clockwise Turing machine that runs in time t, then there is a bi-tag system B_C that simulates the computation of C in time $O(t^2)$ and space $O(t)$.

Before giving the proof of Lemma 2.2 we explain the proof idea. Each A symbol of B_C encodes a symbol of C's tape alphabet. Each E symbol of B_C encodes a state of C. The location of the E symbol in the dataword represents the location of C's tape head, as illustrated in Figure 3.

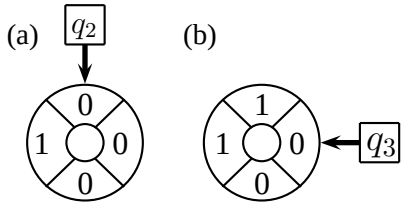
Each clockwise transition rule of C is simulated in the following way. The change of state, symbol and tape head position is simulated by executing a P production over the $E \times A$ pair that encodes the current state and read symbol (see Figure 3(c)). A production is then applied to each symbol in the dataword. This moves the new $E \times A$ pair to the left of the dataword, in order to prepare for the simulation of the next clockwise transition rule.

Proof:
Let clockwise Turing machine C=$(\{q_1, \ldots, q_{|Q|}\}, \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}, f, q_1, q_{|Q|})$. We construct a bi-tag system B_C that simulates C's computation.

$$B_C = (A_C,\ E_C,\ e_{|Q|},\ P_C)$$

where A_C, E_C, P_C are defined below.

$$A_C = \{a_1, \ldots, a_{|\Sigma|}\}$$



(c) $e_2\,0\;\;0\;\;0\;\;1$ apply $e_2 0 \to 1 e_3$

$0\;\;0\;\;1\;\;1\;\;e_3$ apply $0 \to 0$

$0\;\;1\;\;1\;\;e_3\,0$ apply $0 \to 0$

$1\;\;1\;\;e_3\,0\;\;0$ apply $1 \to 1$

$1\;\;e_3\,0\;\;0\;\;1$ apply $1 \to 1$

(d) $e_3\,0\;\;0\;\;1\;\;1$ simulation complete

Figure 3: Bi-tag system simulating the clockwise transition rule $(q_2, 0, 1, q_3)$. The clockwise Turing machine states q_2 and q_3 are encoded as e_2 and e_3, respectively. The e symbols also mark the location of the simulated tape head. (a) A configuration of the clockwise Turing machine before execution of the clockwise transition rule. (b) A configuration of the clockwise Turing machine after execution of the clockwise transition rule. (c) Bi-tag system encoding of the configuration in (a). (d) Bi-tag system encoding of the configuration in (b).

C's tape symbols $\sigma_1, \ldots, \sigma_{|\Sigma|}$ are encoded as $a_1, \ldots, a_{|\Sigma|}$, respectively.

$$E_C = \{e_1, \ldots, e_{|Q|}\}$$

C's states $q_1, \ldots, q_{|Q|}$ are encoded as $e_1, \ldots, e_{|Q|}$, respectively, and the encoded halt state $e_{|Q|}$ is the halt symbol of B_C.

$$P_C = \{a_1 \to a_1, \ldots, a_{|\Sigma|} \to a_{|\Sigma|}\} \cup P'_C$$

P'_C is the set of productions defined on $(E - \{e_{|Q|}\}) \times A$. There is one production in P'_C for each clockwise transition rule in C. Clockwise transition rules fall in two categories, those that write a single symbol from Σ and those that write a pair of symbols from $\Sigma\Sigma$. The two possible clockwise transition rules, and their encodings as productions, are as follows

$$\begin{array}{rcl} (q_x, \sigma_i, \sigma_j, q_y) & : & e_x a_i \to a_j e_y \\ (q_x, \sigma_i, \sigma_j \sigma_k, q_y) & : & e_x a_i \to a_j a_k e_y \end{array}$$

We have constructed a bi-tag system B_C that simulates C. B_C uses $O(t)$ space. To simulate a computation step of C, a production is applied to each symbol in the dataword that encodes the current configuration of C, as the example in Figure 3 illustrates. This takes $O(t)$ steps and yields a new dataword that encodes the next configuration of C's computation. In this way B_C simulates t steps of C's computation in time $O(t^2)$. The simulation halts when the halt symbol $e_{|Q|}$ that encodes the halt state of C becomes the leftmost symbol in the dataword. □

Given a single tape deterministic Turing machine M that runs in time t, we conclude from Lemmata 2.1 and 2.2 that M is simulated by a bi-tag system in time $O(t^4)$. However this overhead is easily improved to $O(t^3)$ as the next theorem shows.

Theorem 2.1. Let M be a deterministic Turing machine with a single tape that computes in time t, then there is a bi-tag system B_M that simulates the computation of M in time $O(t^3)$ and space $O(t)$.

Proof:
From Lemmata 2.1 and 2.2 a bi-tag system simulates the computation of M via a clockwise Turing machine C_M. From Lemma 2.1 C_M simulates M in time $O(t^2)$. However C_M uses $O(t)$ space, hence B_M uses $O(t)$ space. B_M applies $O(t)$ productions to simulate a clockwise transition rule of C_M. Thus B_M simulates $O(t^2)$ clockwise transition rules to simulate M via C_M in time $O(t^3)$. □

3. Universal Turing machines

In this section we give the input encoding to our universal Turing machines. Following this we give each machine and describe its operation by explaining how it simulates bi-tag systems. Let $B = (A, E, e_h, P)$ be a bi-tag system where $A = \{a_1, \ldots, a_q\}$ and $E = \{e_1, \ldots, e_h\}$. The encoding of B as a word is denoted $\langle B \rangle$. The encodings of symbols $a \in A$ and $e \in E$ are denoted $\langle a \rangle$ and $\langle e \rangle$, respectively. The encodings of productions $P(a)$ and $P(e, a)$ are denoted as $\langle P(a) \rangle$ and $\langle P(e, a) \rangle$, respectively.

Definition 3.1. The encoding of a configuration of B is of the form

$$\ldots ccc\langle B\rangle S^* G(\langle A\rangle N)^* \Big(\langle A\rangle N\langle E\rangle \cup \langle E\rangle\langle A\rangle N\Big)(\langle A\rangle N)^* Dccc\ldots \tag{3}$$

where $\langle B \rangle$ is given by Equation (4) and Tables 1, 2 and 3, S^* and G are given by Table 1, and the word $(\langle A\rangle N)^* \Big(\langle A\rangle N\langle E\rangle \cup \langle E\rangle\langle A\rangle N\Big)(\langle A\rangle N)^* D$ encodes B's dataword via Table 1.

$$\begin{aligned} \langle B\rangle = & H\langle P(e_{h-1}, a_q)\rangle V\langle P(e_{h-1}, a_{q-1})\rangle \ldots V\langle P(e_{h-1}, a_1)\rangle \\ & \vdots \\ & V\langle P(e_1, a_q)\rangle V\langle P(e_1, a_{q-1})\rangle \ldots V\langle P(e_1, a_1)\rangle \\ & V^2\langle P(a_q)\rangle V^2\langle P(a_{q-1})\rangle \ldots V^2\langle P(a_1)\rangle V^3 \end{aligned} \tag{4}$$

where V and H are given by Table 1. In Equation (3) the position of the tape head is over the rightmost symbol of G for $U_{15,2}$ and is immediately to the right of $\langle B\rangle S^* G$ for each of the other Turing machines. The initial state is u_1 and the blank symbol is c.

	$\langle a_i \rangle$	$\langle e_j \rangle$	$\langle e_h \rangle$	S	G	N	D	V	H
$U_{5,5}$	b^{4i-1}	b^{4jq}	$b^{4hq+3}\delta$	d^2	ϵ	δ	ϵ	δ	$cd\delta$
$U_{6,4}$	b^{8i-5}	b^{8jq}	$b^{8q(h+1)+5}\delta$	g^2	ϵ	δ	b	δ	Equation (5)
$U_{9,3}$	b^{4i-1}	b^{4jq}	b^{4hq}	c^2	ϵ	δ	ϵ	δcc	$bccbc$
$U_{15,2}$	$(cb)^{8i-5}$	$(cb)^{8jq}$	$(cb)^{8hq+3}bb$	$(cc)^2$	bc	bb	ϵ	cb	$bbcccb$

Table 1: Symbol values for Equations (3) and (4). The value of H for $U_{6,4}$ is given by Equation (5) in Section 3.4.

		$\langle P(e_j, a_i)\rangle$	$\langle P(e_j, a_i)\rangle$
	$\langle P(a_i)\rangle$	$P(e_j, a_i) = a_k e_m$	$P(e_j, a_i) = a_k e_h$
$U_{5,5}$	$\delta\delta d^{16i-6}$	$\delta\delta d^{16mq}\delta d^{16k-6}$	$\delta\delta d^{16hq+14}\delta d^{16k-6}$
$U_{6,4}$	$\delta^5 g^{12i-10}\delta$	$\delta^4 g^{12mq}\delta\delta g^{12k-10}\delta$	$\delta^4 g^{12q(h+1)+8}\delta\delta g^{12k-10}\delta$
$U_{9,3}$	$\delta\delta cc\delta c^{8i}$	$\delta cc\delta\delta c^{8mq+2}\delta c^{8k}$	$\delta cc\delta\delta c^{8hq+2}\delta c^{8k}$
$U_{15,2}$	$(cb)^4(cccb)^2(cc)^{8i-5}$	$(cb)^5(cc)^{8mq}(cccb)^2(cc)^{8k-5}$	$(cb)^3(cccb)^2(cc)^{8hq+3}(cccb)^2(cc)^{8k-5}$

Table 2: Encoding of P productions. Here $a_i, a_k, a_v \in A$ and $e_j, e_m, e_h \in E$. Given in the rightmost column is the special encoding for productions which cause the halt symbol e_h to be printed. Note $U_{9,3}$ encodes such productions, that print e_h, in the same way as its other productions.

	$\langle P(e_j, a_i)\rangle$	$\langle P(e_j, a_i)\rangle$
	$P(e_j, a_i) = a_v a_k e_m$	$P(e_j, a_i) = a_v a_k e_h$
$U_{5,5}$	$\delta d^{16mq}\delta d^{16k-2}\delta d^{16v-6}$	$\delta d^{16hq+14}\delta d^{16k-2}\delta d^{16v-6}$
$U_{6,4}$	$\delta^2 g^{12mq}\delta\delta g^{12hq+12k-4}\delta\delta g^{12v-10}\delta$	$\delta^2 g^{12q(h+1)+8}\delta\delta g^{12hq+12k-4}\delta\delta g^{12v-10}\delta$
$U_{9,3}$	$\delta\delta c^{8mq+2}\delta c^{8k}\delta c^{8v}$	$\delta\delta c^{8hq+2}\delta c^{8k}\delta c^{8v}$
$U_{15,2}$	$(cb)^3(cc)^{8mq}(cccb)^2(cc)^{8k-5}(cccb)^2(cc)^{8v-5}$	$cb(cccb)^2(cc)^{8hq+3}(cccb)^2(cc)^{8k-5}(cccb)^2(cc)^{8v-5}$

Table 3: See caption text for Table 2.

3.1. Universal Turing machine algorithm overview

Each of our universal Turing machines use the same basic simulation algorithm. Here we give a brief description of the algorithm by explaining how our machines locate and simulate a production. The encoded production to be simulated is located using a unary indexing method as illustrated in Figure 4. The encoded production, $\langle P(a_i)\rangle$ or $\langle P(e_j, a_i)\rangle$ in Equation (4), is indexed (pointed to) by the number of symbols contained in the leftmost encoded symbol or pair of symbols in the encoded dataword (Equation (3)). For illustration purposes we assume that we are using $U_{9,3}$. If the leftmost encoded symbol is $\langle a_i\rangle = b^{4i-1}$ (Table 1) then the value $4i-1$ is used to index $\langle P(a_i)\rangle$. If the leftmost encoded symbol is $\langle e_j\rangle = b^{4jq}$, and $\langle a_i\rangle = b^{4i-1}$ is adjacent, then the value $4jq + 4i - 1$ is used to index $\langle P(e_j, a_i)\rangle$. The number of b symbols in the encoded symbol, or pair of encoded symbols, is equal to the number of δc^* words between the leftmost encoded symbol and the encoded production to be simulated. To locate this production, $U_{9,3}$ simply changes one δc^* word to δb^*, for each b in the leftmost encoded symbol or pair of encoded symbols. This process continues until the δ that separates two encoded symbols in the dataword is read. Note from Equation (3) that there is no δ marker between each $\langle e_j\rangle$ and the $\langle a_i\rangle$ to its right, thus allowing $\langle e_j\rangle\langle a_i\rangle$ to be read together during indexing. After indexing, our machines print the indexed production immediately to the right of the encoded dataword as shown in Figure 5. After the indexed production has been printed, then $\langle B\rangle$, the encoding of B, is restored to its original value as illustrated in configurations (ii) and (iii) of Figure 5. This completes the simulation of the production.

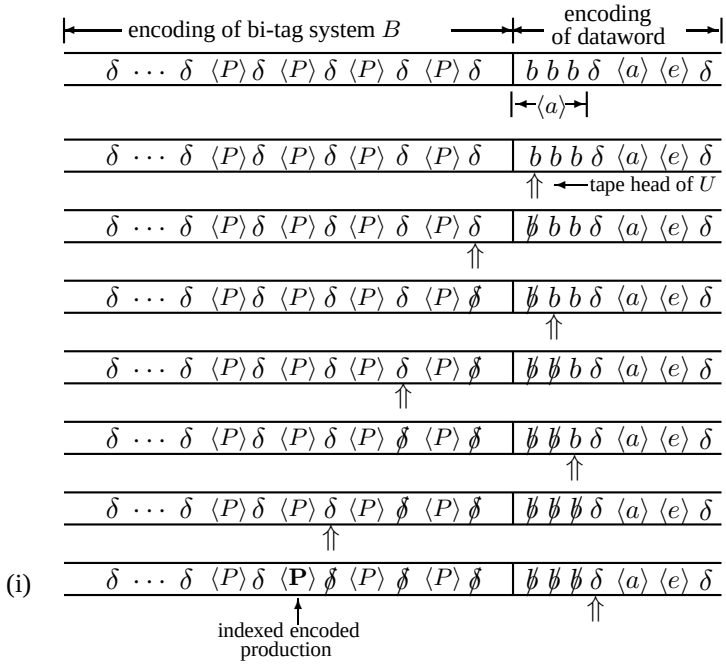


Figure 4: Indexing of an encoded production during simulation of a production of B. The encoded production $\langle P\rangle$, to be executed, is indexed by reading the leftmost encoded symbol $\langle a\rangle$ in the encoded dataword and marking off δ symbols in the encoding of B.

Extensive computer testing has been carried out on each of our universal Turing machines.

3.2. $U_{9,3}$

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
c	bRu_1	cLu_3	cLu_3	bLu_9	cRu_6	bLu_4	δLu_4	cRu_7	bLu_5
b	cLu_2	cLu_2	bLu_4	bLu_4		bRu_6	bRu_7	cRu_9	cRu_8
δ	δRu_3	δLu_2	δRu_1	δLu_4	δLu_8	δRu_6	δRu_7	δRu_8	cRu_1

Table 4: Table of behaviour for $U_{9,3}$.

Example 3.1. ($U_{9,3}$ simulating the execution of the production $P(a_1)$)
This example is presented using three cycles. The tape head of $U_{9,3}$ is given by an underline. The current state of $U_{9,3}$ is given to the left in bold. The dataword $a_1e_ja_i$ is encoded via Equation (3) and Table 1 as $bbb\delta b^{4jq}b^{4i-1}\delta$ and $P(a_1)$ is encoded via Table 2 as $\langle P(a_1)\rangle = \delta\delta cc\delta c^8$. From Equation (3) we get the initial configuration:

$$\boldsymbol{u_1}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8\delta cc\delta cc\delta cc\underline{b}bb\delta b^{4jq}b^{4i-1}\delta ccc\ldots$$

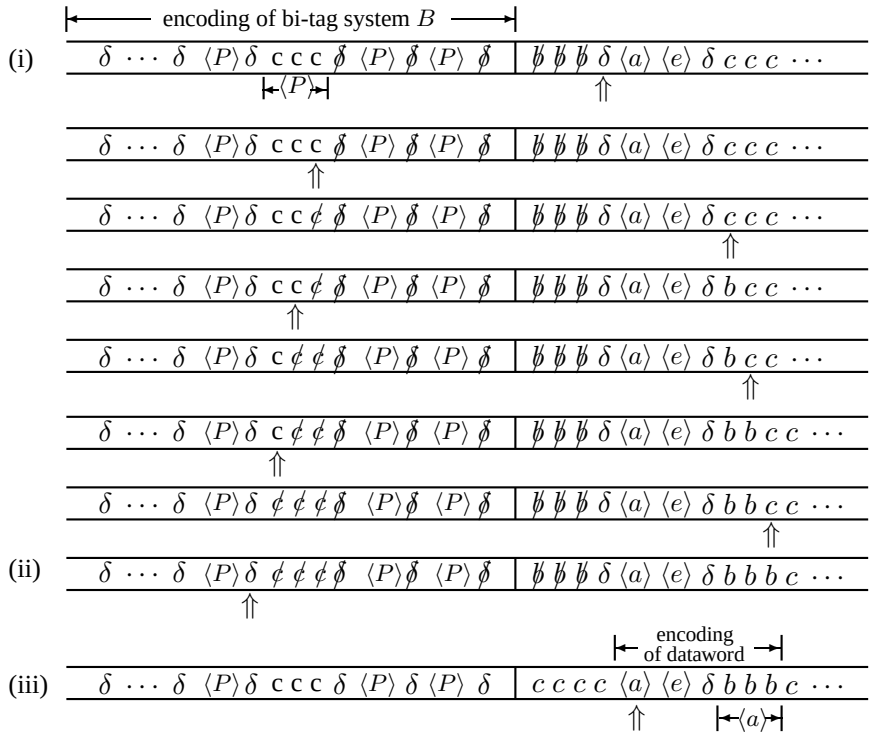


Figure 5: Printing of an encoded production during simulation of a production of B. Over a number of timesteps, the encoded production $\langle P \rangle$ that was indexed in configuration (i) of Figure 4, is printed to the right of the encoded dataword.

Cycle 1 (Index next production). In Cycle 1 (Table 5), $U_{9,3}$ reads the leftmost encoded symbol and locates the next encoded production to execute (see Figure 4). $U_{9,3}$ scans right until it reads b in state u_1. Then $U_{9,3}$ scans left in states u_2 and u_3 until it reads the subword δc^*. This subword is changed to δb^* as $U_{9,3}$ scans right in states u_1 and u_3. The process is repeated until $U_{9,3}$ reads b in state u_3. This indicates that we have finished reading the leftmost encoded symbol, or pair of encoded symbols, and that the encoded production to be executed has been indexed. This signals the end of Cycle 1 and the beginning of Cycle 2.

	u_1	u_2	u_3
c	bRu_1	cLu_3	cLu_3
b	cLu_2	cLu_2	bLu_4
δ	δRu_3	δLu_2	δRu_1

Table 5: Cycle 1 of $U_{9,3}$.

	u_4	u_5	u_6	u_7	u_8	u_9
c	bLu_9	cRu_6	bLu_4	δLu_4	cRu_7	bLu_5
b	bLu_4		bRu_6	bRu_7		
δ	δLu_4	δLu_8	δRu_6	δRu_7	δRu_8	

Table 6: Cycle 2 of $U_{9,3}$.

$$
\begin{aligned}
\vdash \quad & \boldsymbol{u_2}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^8 \delta cc\delta cc\delta c\underline{c}cbb\delta b^{4jq} b^{4i-1} \delta ccc \ldots \\
\vdash^2 \quad & \boldsymbol{u_3}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^8 \delta cc\delta cc\underline{\delta}cccbb\delta b^{4jq} b^{4i-1} \delta ccc \ldots \\
\vdash^4 \quad & \boldsymbol{u_1}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^8 \delta cc\delta cc\delta bbb\underline{b}b\delta b^{4jq} b^{4i-1} \delta ccc \ldots \\
\vdash^{44} \quad & \boldsymbol{u_1}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^8 \delta bb\delta bb\delta bbbbb\underline{\delta} b^{4jq} b^{4i-1} \delta ccc \ldots \\
\vdash^2 \quad & \boldsymbol{u_4}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^8 \delta bb\delta bb\delta bbbbb\underline{\delta} b^{4jq} b^{4i-1} \delta ccc \ldots
\end{aligned}
$$

In the configuration immediately above the encoded production $\langle P(a_1)\rangle$ has been indexed and we have entered Cycle 2.

Cycle 2 (Print production). Cycle 2 (Table 6) prints the encoded production, that was indexed in Cycle 1, immediately to the right of the encoded dataword (see Figure 5). $U_{9,3}$ scans left in state u_4 and records the next symbol of the encoded production to be printed. If $U_{9,3}$ reads the subword ccc it enters state u_6, scans right, and prints b at the right end of the encoded dataword. A single b is printed for each cc pair that does not have δ immediately to its left. If $U_{9,3}$ reads the subword $c\delta cc$ it scans right in state u_7 and prints δ at the right end of the encoded dataword. This process is repeated until the end of the encoded production is detected by reading the subword $\delta\delta cc$ which causes $U_{9,3}$ to enter Cycle 3.

$$
\begin{aligned}
\vdash^{13} \quad & \boldsymbol{u_4}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^6 c\underline{c}(\delta bb)^3 bbb\delta b^{4jq} b^{4i-1} \delta ccc \ldots \\
\vdash^3 \quad & \boldsymbol{u_6}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^6 \underline{b}b(\delta bb)^3 bbb\delta b^{4jq} b^{4i-1} \delta ccc \ldots \\
\vdash^{4(jq+i)+14} \quad & \boldsymbol{u_6}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^6 bb(\delta bb)^3 bbb\delta b^{4jq} b^{4i-1} \delta\underline{c}cc \ldots \\
\vdash \quad & \boldsymbol{u_4}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^6 bb(\delta bb)^3 bbb\delta b^{4jq} b^{4i-1} \underline{\delta} bccc \ldots
\end{aligned}
$$

In the configuration immediately above the first symbol of the encoded production $\langle P(a_1)\rangle$ has been printed. Following the printing of the final symbol of the encoded production we get:

$$
\begin{aligned}
\vdash^* \quad & \boldsymbol{u_4}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta c\underline{c}\delta b^8 (\delta bb)^3 bbb\delta b^{4jq} b^{4i-1} \delta b^3 \delta ccc \ldots \\
\vdash^3 \quad & \boldsymbol{u_8}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \underline{\delta}\delta bb\delta b^8 (\delta bb)^3 bbb\delta b^{4jq} b^{4i-1} \delta b^3 \delta ccc \ldots
\end{aligned}
$$

In the configuration immediately above we have finished printing the encoded production $\langle P(a_1)\rangle$ to the right of the dataword and we have entered Cycle 3.

Cycle 3 (Restore tape). Cycle 3 (Table 7) restores $\langle B\rangle$ to its original value (see configurations (ii) and (iii) in Figure 5). The tape head of $U_{9,3}$ scans right switching between states u_8 and u_9 changing b symbols to c symbols. This continues until $U_{9,3}$ reads the δ marking the leftmost end of the dataword in u_9. Note from Tables 2 and 3 and Equation (4) that there is an even number of b symbols between each pair of δ symbols in $\langle B\rangle$ hence each δ symbol in $\langle B\rangle$ will be read in state u_8. Each a_i symbol in the dataword is encoded by an odd number of b symbols ($\langle a_i\rangle = b^{4i-1}$) and hence the first δ symbol in the dataword will be read in state u_9. This δ symbol marks the left end of the new dataword and causes $U_{9,3}$ to enter state u_1 thus completing Cycle 3 and the production simulation.

$$
\begin{aligned}
\vdash^{25} \quad & \boldsymbol{u_9}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^8 (\delta cc)^3 ccc\underline{\delta} b^{4jq} b^{4i-1} \delta b^3 \delta ccc \ldots \\
\vdash \quad & \boldsymbol{u_1}, \ldots \langle P(a_2)\rangle (\delta cc)^2 \delta\delta cc\delta c^8 (\delta cc)^3 cccc\underline{b} b^{4jq-1} b^{4i-1} \delta b^3 \delta ccc \ldots
\end{aligned}
$$

	u_8	u_9
b	cRu_9	cRu_8
δ	δRu_8	cRu_1

Table 7: Cycle 3 of $U_{9,3}$.

In the last configuration of Cycle 3 our example simulation of production $P(a_1)$ is complete.

Theorem 3.1. Given a bi-tag system B that runs in time t the computation of B is simulated by $U_{9,3}$ in time $O(t^2)$.

Proof:
In order to prove the correctness of $U_{9,3}$ we prove that $U_{9,3}$ simulates any possible $P(a)$ or $P(e,a)$ production of an arbitrary bi-tag system and, that $U_{9,3}$ also simulates halting when the encoded halt symbol $\langle e_h \rangle$ is encountered. In Example 3.1 $U_{9,3}$ simulates $P(a_1)$ for an arbitrary bi-tag system where a_1 is the leftmost symbol in a fixed dataword. This example easily generalises to any production $P(a_i)$ where a_i is the leftmost symbol in an arbitrary dataword. When some $e \in E$ is the leftmost symbol in the dataword then some production $P(e,a)$ must be executed. The simulation of $P(a_1)$ in Example 3.1 is also used to verify the simulation of $P(e,a)$. Note from Equation (3) that there is no δ marker between each $\langle e_j \rangle$ and the adjacent $\langle a_i \rangle$ to its right, thus $\langle e_j \rangle$ and $\langle a_i \rangle$ are read together during Cycle 1. Using the encoding in Definition 3.1, the number of b symbols in $\langle e_j \rangle \langle a_i \rangle$ indexes $\langle P(e,a) \rangle$. Thus, the indexing of $\langle P(e,a) \rangle$ is carried out in the same manner as the indexing of $\langle P(a) \rangle$. The printing of production $\langle P(e,a) \rangle$ during Cycle 2 and the subsequent restoring of $\langle B \rangle$ during Cycle 3 proceed in the same manner as with $P(a_1)$.

If the encoded halt symbol $\langle e_h \rangle = b^{4hq}$ is the leftmost symbol in the encoded dataword, and $\langle a_i \rangle = b^{4-i}$ is adjacent, this is encoded via Definition 3.1 as follows:

$$\boldsymbol{u_1},\ bccbc\langle P(e_{h-1},a_q)\rangle\delta cc\ldots\langle P(a_1)\rangle(\delta cc)^3(cc)^*\underline{b}b^{4hq-1}b^{4i-1}\delta(\langle A\rangle\delta)^*ccc\ldots$$

During Cycle 1, immediately after reading the $(4hq+3)^{\text{th}}$ b symbol in the dataword, $U_{9,3}$ scans left in u_2 and we get the following:

$$\begin{aligned}
&\vdash^* \ \boldsymbol{u_2},\ bccb\underline{c}\langle P(e_{h-1},a_q)\rangle\delta cc\ldots\langle P(a_1)\rangle(\delta cc)^3(cc)^*c^{4hq+3}b^{4i-4}\delta(\langle A\rangle\delta)^*ccc\ldots\\
&\vdash^4 \ \boldsymbol{u_5},\ \underline{b}bbbc\langle P(e_{h-1},a_q)\rangle\delta cc\ldots\langle P(a_1)\rangle(\delta cc)^3(cc)^*c^{4hq+3}b^{4i-4}\delta(\langle A\rangle\delta)^*ccc\ldots
\end{aligned}$$

There is no transition rule in Table 4 for the case 'when in u_5 read b', hence the computation halts. □

The proof of correctness given for $U_{9,3}$ can be applied to the remaining machines in a straightforward way, so we do not restate it.

3.3. $U_{5,5}$

The dataword $a_1e_ja_i$ is encoded via Equation (3) and Table 1 as $bbb\delta b^{4jq}b^{4i-1}\delta$, and $P(a_1)$ is encoded via Table 2 as $\langle P(a_1)\rangle = \delta\delta d^{10}$. From Equation (3) we get the initial configuration:

$$\boldsymbol{u_1},\ \ldots\delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}\delta\delta\delta\underline{b}bb\delta b^{4jq}b^{4i-1}\delta ccc\ldots$$

	u_1	u_2	u_3	u_4	u_5
g	bLu_1	gRu_1	bLu_3		
b	gLu_1	gRu_2	dRu_5	gRu_4	dRu_3
δ	cRu_2	cRu_2	δRu_3	cRu_4	dRu_1
c	δLu_1	bLu_3	δLu_3	δLu_3	
d	bLu_1	gRu_2	bLu_5	bLu_2	bLu_4

Table 8: Table of behaviour for $U_{5,5}$.

Cycle 1 (Index next production). In Cycle 1 (Table 9) when $U_{5,5}$ reads b in state u_1, it changes it to g and scans left until it reads δ. This δ is changed to c and $U_{5,5}$ then enters state u_2 and scans right until it reads g which causes it to re-enter state u_1. This process is repeated until $U_{5,5}$ reads the δ that separates a pair of encoded symbols in the encoded dataword. This signals the end of Cycle 1 and the beginning of Cycle 2.

	u_1	u_2
g	bLu_1	gRu_1
b	gLu_1	gRu_2
δ	cRu_2	cRu_2
c	δLu_1	
d	bLu_1	

Table 9: Cycle 1 of $U_{5,5}$.

$U_{5,5}$	u_2	u_3	u_4	u_5
g		bLu_3		
b	gRu_2		gRu_4	
δ	cRu_2	δRu_3	cRu_4	
c	bLu_3	δLu_3	δLu_3	
d	gRu_2	bLu_5	bLu_2	bLu_4

Table 10: Cycle 2 of $U_{5,5}$.

$U_{5,5}$	u_3	u_5
b	dRu_5	dRu_3
δ	δRu_3	dRu_1

Table 11: Cycle 3 of $U_{5,5}$.

$$\begin{aligned}
&\vdash^3 && \boldsymbol{u_1}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}\delta\delta cg\underline{b}b\delta b^{4jq}b^{4i-1}\delta ccc\ldots\\
&\vdash^{18} && \boldsymbol{u_1}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}cccggg\underline{\delta}b^{4jq}b^{4i-1}\delta ccc\ldots\\
&\vdash && \boldsymbol{u_2}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}cccggc\underline{b}b^{4jq-1}b^{4i-1}\delta ccc\ldots
\end{aligned}$$

Cycle 2 (Print production). Cycle 2 (Table 10) begins with $U_{5,5}$ scanning right and printing b to the right of the encoded dataword. Following this, $U_{5,5}$ scans left in state u_3 and records the next symbol of the encoded production to be printed. If $U_{5,5}$ reads the subword $dddd$ it enters state u_2, scans right, and prints b at the right end of the encoded dataword. If $U_{5,5}$ reads the subword δdd it scans right in state u_4 and prints δ at the right end of the encoded dataword. This process is repeated until the end of the encoded production is detected by reading δ in state u_3, which causes $U_{5,5}$ to enter Cycle 3.

$$\begin{aligned}
&\vdash^* && \boldsymbol{u_3}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^6 ddd\underline{d}\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bccc\ldots\\
&\vdash^3 && \boldsymbol{u_2}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^6\underline{d}bbb\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bccc\ldots\\
&\vdash^* && \boldsymbol{u_3}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d\underline{d}b^8\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bbbccc\ldots\\
&\vdash^2 && \boldsymbol{u_4}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\underline{\delta}bbb^8\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bbbccc\ldots\\
&\vdash^* && \boldsymbol{u_3}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\underline{\delta}\delta bbb^8\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bbb\delta ccc\ldots
\end{aligned}$$

Cycle 3 (Restore tape). In Cycle 3 (Table 11) the tape head of $U_{5,5}$ scans right switching between states u_3 and u_5 changing b symbols to d symbols. This continues until $U_{5,5}$ reads the δ marking the leftmost

end of the encoded dataword in u_5. Note from Tables 2 and 3 and Equation (4) that there is an even number of d symbols between each pair of δ symbols in $\langle B\rangle$ hence each δ symbol in $\langle B\rangle$ will be read in state u_3. Each a_i symbol in the dataword is encoded by an odd number of symbols ($\langle a_i\rangle = b^{4i-1}$) and hence the first δ symbol in the dataword will be read in state u_5. This causes $U_{5,5}$ to enter state u_1 thus completing Cycle 3 and the production simulation.

$$\vdash^{19} \qquad \boldsymbol{u_1},\ \ldots\delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}\delta\delta\delta ddd\underline{d}b^{4jq-1}b^{4i-1}\delta bbb\delta ccc\ldots$$

Halting for $\mathbf{U_{5,5}}$. If the encoded halt symbol $\langle e_h\rangle = b^{4hq+3}\delta$ is the leftmost symbol in the encoded dataword then this is encoded via Definition 3.1 as follows:

$$\boldsymbol{u_1},\ cd\delta\langle P(e_{h-1},a_q)\rangle\delta\ldots\delta^2\langle P(a_1)\rangle\delta^3(dd)^*\underline{b}b^{4hq+2}\delta(\langle A\rangle\delta)^*ccc\ldots$$

The computation continues as before until $U_{5,5}$ enters Cycle 2 and scans left in u_3. Immediately after $U_{5,5}$ reads the leftmost d during this leftward scan we get:

$$\vdash^* \qquad \boldsymbol{u_5},\ \underline{c}b\delta\langle P(e_{h-1},a_q)\rangle'\delta\ldots\delta^2\langle P(a_1)\rangle'\delta^3(dd)^*b^{4hq+3}\delta(\langle A\rangle\delta)^*bccc\ldots$$

In the configuration above, $\langle P\rangle'$ denotes the word in which all the d symbols in $\langle P\rangle$ are changed to b symbols. There is no transition rule in Table 8 for the case 'when in u_5 read c' hence the computation halts.

3.4. $U_{6,4}$

	u_1	u_2	u_3	u_4	u_5	u_6
g	bLu_1	gRu_1	bLu_3	bRu_2	bLu_6	bLu_4
b	gLu_1	gRu_2	bLu_5	gRu_4	gRu_6	gRu_5
δ	cRu_2	cRu_2	δLu_5	cRu_4	δRu_5	gRu_1
c	δLu_1	gRu_5	δLu_3	cRu_5	bLu_3	

Table 12: Table of behaviour for $U_{6,4}$.

The dataword $a_1e_ja_i$ is encoded via Equation (3) and Table (1) as $bbb\delta b^{8jq}b^{8i-5}\delta b$. From Equation (3) we get the initial configuration:

$$\boldsymbol{u_1},\ \ldots\delta^2\langle P(a_2)\rangle\delta^2\langle P(a_1)\rangle\delta\delta\delta\underline{b}bb\delta b^{8jq}b^{8i-5}\delta bccc\ldots$$

Cycle 1 (Index next production). In Cycle 1 (Table 13) when $U_{6,4}$ reads b in state u_1 it scans left until it reads δ. This δ is changed to c and $U_{6,4}$ then enters state u_2 and scans right until it reads g which causes it to re-enter state u_1. This process is repeated until $U_{6,4}$ reads the δ that separates a pair of encoded symbols in the encoded dataword. This signals the end of Cycle 1 and the beginning of Cycle 2.

Cycle 2 (Print production). Cycle 2 (Table 14) begins with $U_{6,4}$ scanning right and printing bb to the right of the encoded dataword. Following this, $U_{6,4}$ scans left in state u_3 and records the next symbol of the encoded production to be printed. If $U_{6,4}$ reads the subword $ggg\delta$ or $gggb$ it enters state u_2, scans

	u_1	u_2
g	bLu_1	gRu_1
b	gLu_1	gRu_2
δ	cRu_2	cRu_2
c	δLu_1	

Table 13: Cycle 1 of $U_{6,4}$.

	u_2	u_3	u_4	u_5	u_6
g		bLu_3	bRu_2	bLu_6	bLu_4
b	gRu_2	bLu_5	gRu_4		
δ	cRu_2	δLu_5	cRu_4	δRu_5	
c	gRu_5	δLu_3	cRu_5	bLu_3	

Table 14: Cycle 2 of $U_{6,4}$.

	u_5	u_6
b	gRu_6	gRu_5
δ	δRu_5	gRu_1

Table 15: Cycle 3 of $U_{6,4}$.

right, and prints bb at the right end of the encoded dataword. If $U_{6,4}$ reads the subword δggb it scans right in state u_4 and prints δb at the right end of the encoded dataword. This process is repeated until the end of the encoded production is detected by reading δ in state u_5, which causes $U_{6,4}$ to enter Cycle 3.

Cycle 3 (Restore tape). In Cycle 3 (Table 15) the tape head of $U_{6,4}$ scans right switching between states u_5 and u_6, changing b symbols to g symbols. This continues until $U_{6,4}$ reads the δ marking the leftmost end of the encoded dataword in u_6. Note from Tables 2 and 3 and Equation (4) that there is an even number of g symbols between each pair of δ symbols in $\langle B \rangle$, hence each δ symbol in $\langle B \rangle$ is read in state u_5. Each a_i symbol in the dataword is encoded by an odd number of symbols ($\langle a_i \rangle = b^{8i-5}$) and hence the first δ symbol in the dataword is read in state u_6. This causes $U_{6,4}$ to enter state u_1, thus completing Cycle 3 and the production simulation.

Special case for $U_{6,4}$. If we are simulating a production of the form $P(e, a) = a_v a_k e_m$ we have a special case. Note from Table 3 and Cycle 2 that the simulation of $P(e, a) = a_v a_k e_m$ for $U_{6,4}$ results in the word $b^{8v-5}\delta b^{8hq+8k-3}\delta b^{8mq} b$ being printed to the right of the dataword. From Table 1 it is clear that a_k is not encoded in this word in its usual from. However when $U_{6,4}$ reads the subword $b^{8hq+8k-3}\delta$ it indexes $\langle P(a_k) \rangle$ in H which results in $\langle a_k \rangle$ being printed to the dataword. To see this, note that the value of H from Equation (4) for $U_{6,4}$ is as follows:

$$H = cgbV^2\langle P(a_q)\rangle V^2\langle P(a_{q-1})\rangle \dots V^2\langle P(a_1)\rangle V^3 \tag{5}$$

The halting condition for $U_{6,4}$ occurs in a similar manner to that of $U_{5,5}$. Halting occurs during the first scan left in Cycle 2 when $U_{6,4}$ reads c in state u_6 at the left end of $\langle B \rangle$ (note from Table 12 that there is no transition rule for state-symbol pair (u_6, c)).

3.5. $U_{15,2}$

Example 3.2. ($U_{15,2}$ simulating the execution of the production $P(a_1)$)
The example dataword $a_1 e_j a_i$ is encoded via Equation (3) and Table (1) as $cbcbcbbb(cb)^{8jq}(cb)^{8i-5}bb$ and $P(a_1)$ is encoded via Table 2 as $\langle P(a_1) \rangle = (cb)^4(cccb)^2(cc)^3$. Thus from Equation (3) we get the following initial configuration

$$\boldsymbol{u_1}, \dots \langle P(a_2)\rangle (cb)^6 (cccb)^2 (cc)^3 \; cb\; cb\; cb\; b\underline{c}\; cb\; cb\; cb\; bb\; (cb)^{8jq}(cb)^{8i-5}\; bb\; cc \dots$$

In this example we explain how $U_{15,2}$ operates by considering how it treats pairs of symbols during each cycle. Thus, the extra whitespace between each pair of symbols is to improve readability and help illustrate our explanation of $U_{15,2}$'s operation.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
c	cRu_2	bRu_3	cLu_7	cLu_6	bRu_1	bLu_4	cLu_8	bLu_9
b	bRu_1	bRu_1	cLu_5	bLu_5	bLu_4	bLu_4	bLu_7	bLu_7

	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}
c	cRu_1	bLu_{11}	cRu_{12}	cRu_{13}	cLu_2	cLu_3	cRu_{14}
b	bLu_{10}		bRu_{14}	bRu_{12}	bRu_{12}	cRu_{15}	bRu_{14}

Table 16: Table of behaviour for $U_{15,2}$.

Cycle 1 (Index next production). In Cycle 1 (Table 17) $U_{15,2}$ scans right in states u_1, u_2 and u_3 until it reads the subword ccb which it changes to cbc. Following this, it scans left in states u_4, u_5 and u_6 until it reads the subword cb. This cb is changed to bb and $U_{15,2}$ re-enters state u_1 and scans right. This process is repeated until $U_{15,2}$ has finished reading the encoded read symbol $\langle a_i \rangle$ or symbols $\langle a_i \rangle$ and $\langle e_j \rangle$. This occurs when the subword ccb no longer appears to the right of the tape head and signals the end of Cycle 1 and the beginning of Cycle 2.

	u_1	u_2	u_3	u_4	u_5	u_6
c	cRu_2	bRu_3	cLu_7	cLu_6	bRu_1	bLu_4
b	bRu_1	bRu_1	cLu_5	bLu_5	bLu_4	bLu_4

Table 17: Cycle 1 of $U_{15,2}$.

$$
\begin{aligned}
\vdash^{3} \quad & \boldsymbol{u_5}, \ldots \langle P(a_2) \rangle (cb)^6 (cccb)^2 (cc)^3 \, cb \, cb \, cb \, bc \, \underline{b}c \, cb \, cb \, bb \, (cb)^{8jq} (cb)^{8i-5} \, bb \, cc \ldots \\
\vdash^{4} \quad & \boldsymbol{u_5}, \ldots \langle P(a_2) \rangle (cb)^6 (cccb)^2 (cc)^3 \, cb \, cb \, \underline{c}b \, bc \, bc \, cb \, cb \, bb \, (cb)^{8jq} (cb)^{8i-5} \, bb \, cc \ldots \\
\vdash^{4} \quad & \boldsymbol{u_2}, \ldots \langle P(a_2) \rangle (cb)^6 (cccb)^2 (cc)^3 \, cb \, cb \, bb \, bc \, \underline{b}c \, cb \, cb \, bb \, (cb)^{8jq} (cb)^{8i-5} \, bb \, cc \ldots \\
\vdash^{20} \quad & \boldsymbol{u_2}, \ldots \langle P(a_2) \rangle (cb)^6 (cccb)^2 (cc)^3 \, cb \, bb \, bb \, bc \, bc \, \underline{b}c \, cb \, bb \, (cb)^{8jq} (cb)^{8i-5} \, bb \, cc \ldots \\
\vdash^{28} \quad & \boldsymbol{u_2}, \ldots \langle P(a_2) \rangle (cb)^6 (cccb)^2 (cc)^3 \, bb \, bb \, bb \, bc \, bc \, bc \, \underline{b}c \, bb \, (cb)^{8jq} (cb)^{8i-5} \, bb \, cc \ldots
\end{aligned}
$$

Note that in the configuration immediately above each cb subword in the encoded read symbol $\langle a_1 \rangle = cbcbcb$ has been changed to the subword bc. Note also that the substring ccb which causes a scan to the left in u_4, u_5, and u_6 no longer appears in the configuration to the right of the tape head. This causes $U_{15,2}$ to enter Cycle 2.

Cycle 2 (Print production). Cycle 2 (Table 18) begins with $U_{15,2}$ scanning right and printing cb to the right of the encoded dataword. Following this, $U_{15,2}$ scans left in states u_7, u_8, u_9, u_{10} and u_{11} and records the next symbol of the encoded production to be printed. If, during a scan left, $U_{15,2}$ reads the subword ccc then it scans right in states u_1 and u_2 and changes the cc immediately to the right of the encoded dataword to cb. If, during a scan left, $U_{15,2}$ reads the subword $ccbcc$ it scans right in states u_{12} and u_{13} and changes the first c to the right of the encoded dataword to b. This process is repeated until the end of the encoded production is detected by reading the subword $bcbcc$ during the scan left. This causes $U_{15,2}$ to enter Cycle 3.

	u_1	u_2	u_3	u_7	u_8	u_9	u_{10}	u_{11}	u_{12}	u_{13}
c	cRu_2	bRu_3	cLu_7	cLu_8	bLu_9	cRu_1	bLu_{11}	cRu_{12}	cRu_{13}	cLu_2
b	bRu_1	bRu_1	cLu_5	bLu_7	bLu_7	bLu_{10}		bRu_{14}	bRu_{12}	bRu_{12}

Table 18: Cycle 2 of $U_{15,2}$.

$$\begin{aligned}
\vdash^* &\quad \boldsymbol{u_1}, \ldots \langle P(a_2)\rangle (cb)^6 (cccb)^2\, cc\, cc\, cc\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, \underline{c}c\, cc\, cc \ldots \\
\vdash^3 &\quad \boldsymbol{u_7}, \ldots \langle P(a_2)\rangle (cb)^6 (cccb)^2\, cc\, cc\, cc\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, c\underline{b}\, cc\, cc \ldots \\
\vdash^* &\quad \boldsymbol{u_7}, \ldots \langle P(a_2)\rangle (cb)^6 (cccb)^2\, cc\, cc\, c\underline{c}\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, cb\, cc\, cc \ldots \\
\vdash^3 &\quad \boldsymbol{u_1}, \ldots \langle P(a_2)\rangle (cb)^6 (cccb)^2\, cc\, cc\, \underline{b}c\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, cb\, cc\, cc \ldots \\
\vdash^* &\quad \boldsymbol{u_7}, \ldots \langle P(a_2)\rangle (cb)^6 (cccb)^2\, cc\, c\underline{c}\, bc\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, cb\, cb\, cc \ldots
\end{aligned}$$

Each time the substring ccc is read during a scan left in states u_7, u_8, and u_9 $U_{15,2}$ scans right and prints cb to the right of the encoded dataword. Thus we get:

$$\begin{aligned}
\vdash^* &\quad \boldsymbol{u_7}, \ \ldots\ \langle P(a_2)\rangle (cb)^6\, cc\, cb\, cc\, cb\, c\underline{c}\, bc\, bc\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, (cb)^3\, cc\, cc \ldots \\
\vdash^5 &\quad \boldsymbol{u_{12}}, \ldots \langle P(a_2)\rangle (cb)^6\, cc\, cb\, cc\, \underline{b}b\, bc\, bc\, bc\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, (cb)^3\, cc\, cc \ldots \\
\vdash^* &\quad \boldsymbol{u_7}, \ \ldots\ \langle P(a_2)\rangle (cb)^6\, cc\, cb\, c\underline{c}\, bb\, bc\, bc\, bc\, (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, (cb)^3\, bc\, cc \ldots
\end{aligned}$$

Each time the substring $ccbcc$ is read during a scan left in states u_7, u_8, u_9, u_{10}, and u_{11} $U_{15,2}$ scans right and prints b to the right of the encoded dataword. Thus we get:

$$\begin{aligned}
\vdash^* &\quad \boldsymbol{u_7}, \ \ldots\ \langle P(a_2)\rangle (cb)^4\, cb\, cb\, c\underline{c}\, bb\, bc\, bb\, (bc)^3 (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, (cb)^3\, bb\, cc \ldots \\
\vdash^5 &\quad \boldsymbol{u_{14}}, \ldots \langle P(a_2)\rangle (cb)^4 cb\, \underline{b}b\, bc\, bb\, bc\, bb\, (bc)^3 (bb)^3 (bc)^4\, bb\, (cb)^{8jq} (cb)^{8i-5}\, bb\, (cb)^3\, bb\, cc \ldots
\end{aligned}$$

When the substring $bcbcc$ is read during a scan left in states u_7, u_8, u_9, u_{10}, and u_{11} Cycle 2 is complete and Cycle 3 is entered. Thus in the configuration immediately above $U_{15,2}$ has entered Cycle 3.

Cycle 3 (Restore tape). In Cycle 3 (Table 19) the tape head of $U_{15,2}$ scans right in states u_{14} and u_{15} changing each bc to cc and each bb to cb. This continues until $U_{15,2}$ reads c in state u_{14}. This c marks the leftmost end of the dataword. Note that during Cycles 1 and 2 each cc in $\langle B\rangle$ and each cb in the encoded read symbol are changed to the subwords bc. Also during Cycles 1 and 2, each cb subword in $\langle B\rangle$ is changed to the subword bb. Thus c will not be read in u_{14} until we encounter the subword cb at the left end of the next encoded symbol to be read in the dataword.

$$\begin{aligned}
\vdash^9 &\quad \boldsymbol{u_{15}}, \ldots \langle P(a_2)\rangle (cb)^4\, cb\, cb\, cc\, cb\, cc\, c\underline{b}\, (bc)^3 (bb)^3 (bc)^4\, bb\, cb\, (cb)^{8jq-1} (cb)^{8i-5}\, bb\, (cb)^3\, bb\, cc \ldots \\
\vdash^* &\quad \boldsymbol{u_{14}}, \ldots \langle P(a_2)\rangle (cb)^4\, cb\, cb\, cc\, cb\, cc\, cb\, (cc)^3 (cb)^3 (cc)^4\, cb\, \underline{c}b\, (cb)^{8jq-1} (cb)^{8i-5}\, bb\, (cb)^3\, bb\, cc \ldots \\
\vdash^3 &\quad \boldsymbol{u_1}, \ldots \langle P(a_2)\rangle (cb)^6 (cccb)^2 (cc)^3 (cb)^3 (cc)^4\, b\underline{c}\, cb\, (cb)^{8jq-1} (cb)^{8i-5}\, bb\, (cb)^3\, bb\, cc \ldots
\end{aligned}$$

In the configuration immediately above the example simulation of production $\langle P(a_1)\rangle$ is complete. The encoding of $\langle a_1\rangle = (cb)^3$ has been appended onto the right end of the dataword, the encoded tag system $\langle B\rangle$ has been restored to its original value and $U_{15,2}$ is ready to read the encoded symbols $\langle e_j\rangle$ and $\langle a_i\rangle$.

	u_3	u_5	u_{14}	u_{15}
c		bRu_1	cLu_3	cRu_{14}
b	cLu_5		cRu_{15}	bRu_{14}

Table 19: Cycle 3 of $U_{15,2}$.

Halting for $\mathbf{U_{15,2}}$. If the halt symbol e_h, encoded as $\langle e_h\rangle = (cb)^{8hq+3}bb$, is the leftmost symbol in the dataword then this is encoded via Definition 3.1 as follows:

$$\boldsymbol{u_1},\ bb\ cc\ cb\ \langle P(e_{h-1}, a_q)\rangle\ cb\ \ldots (cb)^2\langle P(a_1)\rangle (cb)^3((cc^2))^*\ b\underline{c}\ (cb)^{8hq+3}\ bb\ (\langle A\rangle\ bb)^*\ cc\ cc\ldots$$

The computation continues as before until $U_{15,2}$ enters Cycle 2 and scans left in u_7, u_8, and u_9. This scan ends with the following configuration:

$$\vdash \qquad \boldsymbol{u_{10}},\ \underline{b}b\ bc\ bb\ \langle P(e_{h-1}, a_q)\rangle'\ bb\ \ldots (bb)^2\langle P(a_1)\rangle'(bb)^3((bc)^2)^*\ (bc)^{8hq+4}\ bb\ (\langle A\rangle\ bb)^*\ cb\ cc\ldots$$

In the configuration immediately above, $\langle P\rangle'$ denotes the word in which each cc and cb subword in $\langle P\rangle$ is changed to the subword bc and bb, respectively. There is no transition rule in Table 16 for the case 'when in u_{10} read c' hence the computation halts.

4. Conclusion

In order to determine the minimum size for universal Turing machines we must identify the largest possible non-universal Turing machine. Traditionally this has been done by proving the halting problem decidable for a given state-symbol pair. For example the decidability results given in Figure 1 imply that a universal Turing machine, that simulates any Turing machine M and halts if and only if M halts, is not possible for these state-symbol pairs. Hence these results give lower bounds on the size of universal machines of this type. The decidable halting problem curve in Figure 1 could be considered a non-universality curve in this sense. Thus, following the new universal machines presented in this work there are 39 state-symbol pairs that remain open.

There has been no improvement on universal Turing machine lower bound results since 1978, when Pavlotskaya [19] proved that the halting problem is decidable for 3-state, 2-symbol machines. The proof is quite long and complex, and improving on this result may well be difficult. As the state-symbol product increases, the number of possible machines increases exponentially. Thus it seems that a new approach needs to be taken. To find new lower bounds one possible method is to prove that some non-universal system simulates all of the Turing machines for a given state-symbol pair.

It has been noted in the literature that Minsky's 7-state, 4 symbol machine [11], which simulates 2-tag systems, mutilates the final output. While it is true that Minsky's machine changes the final output by making one extra pass over the dataword before halting, this does not prevent his machine from

simulating all Turing machines. This is due to the following fact. When Cocke and Minsky's 2-tag algorithm [2] (or the algorithms given in [13] and [30]) is used to simulate Turing machines, this extra pass over the final dataword does not lose any information: in the sense that the simulated Turing machine's final output can be retrieved by a (simple) decoding function.

Since Minsky's universal machine all of the smallest universal Turing machines (including our bi-tag simulators) have used a similar algorithm. Rogozhin [22, 23] extended Minsky's technique to establish the universal curve. There have been incremental reductions in the size of many of Rogozhin's machines. However, the smallest of Rogozhin's machines, the 6-symbol machine, has not been improved upon since it was first presented almost 30 years ago. In order to significantly reduce the space between the decidable halting problem curve and the universality curve we suspect that a radically new approach must be taken. Below we give three methods to aid in the search for smaller universal Turing machines.

The first approach is to look for some universal systems, other than 2-tag or bi-tag systems, that would require less instructions to simulate. Cyclic tag systems [3] may be used to give smaller machines. However, the operation of cyclic tag systems is similar to that of 2-tag and bi-tag systems so this may not give much of an improvement. Perhaps a simple universal cellular automaton could be simulated. The cellular automaton Rule 110 has given rise to very small weakly universal Turing machines [3, 17, 27]. In the proof of universality of Rule 110, the initial condition contains a finite sequence of states that is repeated infinitely often to the left, and another finite sequence that is repeated infinitely often to the right. These small Turing machines that simulate it use a similar kind of initial condition and are thus said to be weakly universal. Perhaps a sufficiently simple universal cellular automaton could be found that allows us to construct small Turing machines that are universal, rather than only weakly universal.

Another approach is to simplify some existing universal model in order to make it easier to simulate. As an interesting example we will briefly consider small semi-weak machines (which are Turing machines with an infinitely repeated word on one side of the input and the usual repeated blank symbol on the other side). Watanabe [25] gave a small semi-weakly universal Turing machine with 5 symbols and 6 states that simulates Turing machines directly. Later, Watanabe [26] gave a small semi-weakly universal Turing machine with 4 symbols and 5 states that simulates restricted Turing machines. Watanabe noted that Turing machines with a binary $\{0, 1\}$ tape alphabet, where the tape head always moves right on 1 and left on 0, are universal. Because of this restriction, Watanabe's encoded table of behaviour for each Turing machine had no need to include information about the direction of movement of the tape head. This in turn simplified the problem of simulating Turing machines.

A third approach is to find an encoding that allows many different operations to be carried out by the same group of instructions. For example, this approach aided in the construction of the smallest known universal Turing machines [15] that simulate Turing machine directly. The encoding used by these machines allowed each set of transition rules to serve more than one purpose. A single set of transition rules reads both the encoded current state and the encoded read symbol. Another set of transition rules (1) prints the encoded write symbol (2) moves the simulated tape head and (3) establishes the new encoded current state. Combining steps in this way has reduced the number of transition rules needed for these machines.

The small universal machines we have given here conform to the classical model used by Minsky [12] and Shannon [24]. Generalising the Turing machine model often allows us to find universal Turing machines with smaller state-symbol products, such as the small weakly universal machines in [3, 17, 27], and the semi-weakly universal machines in [25, 26, 29]. It is important to note that the decidability results given in Figure 1 do not give lower bounds for such machines. In order to give relevant lower bounds

for these machines new decidability results must be given for these more general models. We note also that the weak machines in [3, 17] do not halt and thus proving the halting problem decidable does not immediately imply lower bounds relevant for these machines.

If we restrict the standard Turing machine model then the problem of finding machines with small state-symbol products often becomes more difficult. Some examples of restricted small universal Turing machines have been given by Margenstern [9]. Margenstern also gives a number of decidability results for restricted universal Turing machines.

By giving upper and lower bounds, some boundaries have been found for the smallest possible universal machines in terms of numbers of states and symbols with respect to more general Turing machine models [10, 20]. However, there are many open questions remaining in the world of small universal Turing machines. What are the trade-offs between number of states/symbols, space/time complexity, and encoding complexity in small universal Turing machines? How do the smallest possible universal machines for the different generalisations and restrictions compare? Do the upper and lower bounds meet? That is, do there exist machines in the space between the smallest possible universal Turing machines and the largest machines with a decidable halting problem? To this day Shannon's 50 year old question, regarding the smallest possible universal Turing machine, still remains unanswered.

References

[1] C. Baiocchi. Three small universal Turing machines. In M. Margenstern and Y. Rogozhin, editors, *Machines, Computations, and Universality (MCU)*, volume 2055 of *LNCS*, pages 1–10, Chişinău, Moldova, May 2001. Springer.

[2] J. Cocke and M. Minsky. Universality of tag systems with $P = 2$. *Journal of the ACM*, 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, Ninth Annual Symposium on Switching and Automata Theory*, 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, Nov. 1996.

[6] M. Kudlek and Y. Rogozhin. Small universal circular Post machines. *Computer Science Journal of Moldova*, 9(1):34–52, 2001.

[7] M. Kudlek and Y. Rogozhin. A universal Turing machine with 3 states and 9 symbols. In W. Kuich, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory*, volume 2295 of *LNCS*, pages 311–318, Vienna, May 2002. Springer.

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

[9] M. Margenstern. On quasi-unilateral universal Turing machines. *Theoretical Computer Science*, 257(1–2):153–166, Apr. 2001.

[10] 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.

[11] M. Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory, Symposium in Pure Mathematics*, volume 5, pages 229–238, Provelence, 1962. AMS.

[12] M. Minsky. *Computation, finite and infinite machines*. Prentice-Hall, New Jersey, 1967.

[13] T. Neary. *Small universal Turing machines*. PhD thesis, Department of Computer Science, National University of Ireland, Maynooth, 2008.

[14] T. Neary and D. Woods. A small fast universal Turing machine. Technical Report NUIM-CS-TR-2005-12, Department of Computer Science, National University of Ireland, Maynooth, Dec. 2005.

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

[16] 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.

[17] T. Neary and D. Woods. Small weakly universal Turing machines. Technical Report arXiv:0707.4489v1 [cs.CC], arXiv online report, July 2007.

[18] 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).

[19] 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).

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

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

[22] Y. Rogozhin. Sem' universal'nykh mashin T'juringa. In *Fifth all union conference on Mathematical Logic, Akad. Naul SSSR. Otdel. Inst. Mat., Novosibirsk*, page 27, 1979. (Seven universal Turing machines. In Russian).

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

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

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

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

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

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

[29] D. Woods and T. Neary. Small semi-weakly universal Turing machines. *Fundamenta Informaticae*, 91:161–177, 2009.

[30] 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.