Journal of Complexity 30 (2014) 634–646

Contents lists available at ScienceDirect

Journal of Complexity

journal homepage: www.elsevier.com/locate/jco

Wang's B machines are efficiently universal, as is Hasenjaeger's small universal electromechanical toy☆

Turlough Neary [a], Damien Woods [b], Niall Murphy [c,d,*], Rainer Glaschick [e]

[a] *Institute for Neuroinformatics, University of Zürich & ETH Zürich, Switzerland*
[b] *California Institute of Technology, Pasadena, CA 91125, USA*
[c] *Facultad de Informática, Universidad Politécnica de Madrid & CEI-Moncloa UPM-UCM, Spain*
[d] *Microsoft Research, Cambridge, CB1 2FB, UK*[1]
[e] *Paderborn, Germany*

ARTICLE INFO

Article history:
Received 29 September 2013
Accepted 31 January 2014
Available online 14 February 2014

Keywords:
Polynomial time
Computational complexity
Small universal Turing machines
Wang's B machine
Non-erasing Turing machines
Models of computation

ABSTRACT

In the 1960s Gisbert Hasenjaeger built Turing Machines from electromechanical relays and uniselectors. Recently, Glaschick reverse engineered the program of one of these machines and found that it is a universal Turing machine. In fact, its program uses only four states and two symbols, making it a very small universal Turing machine. (The machine has three tapes and a number of other features that are important to keep in mind when comparing it to other small universal machines.) Hasenjaeger's machine simulates Hao Wang's B machines, which were proved universal by Wang. Unfortunately, Wang's original simulation algorithm suffers from an exponential slowdown when simulating Turing machines. Hence, via this simulation, Hasenjaeger's machine also has an exponential slowdown when simulating Turing machines. In this work, we give a new efficient simulation algorithm for Wang's B machines by showing that they simulate Turing machines with only a polynomial slowdown. As a second result, we find that Hasenjaeger's

☆ This work is the outcome of a visit by the authors to the Isaac Newton Institute for Mathematical Sciences in Cambridge (UK) for the centenary of Alan Turing's birth, hosted by the 2012 program "Semantics and Syntax: A Legacy of Alan Turing'.

* Corresponding author at: Microsoft Research, Cambridge, CB1 2FB, UK.
E-mail addresses: tneary@ini.phys.ethz.ch (T. Neary), woods@caltech.edu (D. Woods), a-nimurp@microsoft.com (N. Murphy), rainer@glaschick-pb.de (R. Glaschick).

[1] Present affiliation.

http://dx.doi.org/10.1016/j.jco.2014.02.003

machine also efficiently simulates Turing machines in polynomial time. Thus, Hasenjaeger's machine is both small and fast. In another application of our result, we show that Hooper's small universal Turing machine simulates Turing machines in polynomial time, an exponential improvement.

1. Introduction

In the 1960s Gisbert Hasenjaeger built Turing Machines from electromechanical relays and uniselectors, but never published details of these machines. Recently, Hasenjaeger's family donated the machine shown in Fig. 1 to the Heinz Nixdorf MuseumsForum.[2] At the request of the MuseumsForum, Glaschick reverse engineered the table of behavior for this machine [3,1], and, using Hasenjaeger's notes [4], determined the machine's encoding and operation. It was found that Hasenjaeger's machine simulates Wang's B machines [15].

Wang used a unary encoding when proving his B machines universal and hence they suffer from an exponential slowdown when simulating Turing machines. As a result, Hasenjaeger's machine also suffers from an exponential slowdown. In this work, we show that Wang B machines and Hasenjaeger's machine simulate Turing machines with polynomial slowdown via the following chain of simulations:

$$\text{Turing Machine} \mapsto \text{non-erasing Turing Machine} \mapsto \text{Wang B machine} \mapsto \text{Hasenjaeger's universal Turing Machine}$$

where $A \mapsto B$ denotes that A is simulated by B. With the exception of the Wang B machine simulation of non-erasing machines, all of the simulations in the above chain are known to be efficient: non-erasing Turing machines simulate Turing machines with a polynomial slowdown in time [17], and Hasenjaeger's machine simulates Wang B machines in linear time. We complete the chain of efficient simulations by giving a new simulation algorithm that shows that Wang's B machines simulate Turing machines with only a polynomial slowdown in the simulated Turing machine's time. An immediate consequence of our new algorithm is that Hasenjaeger's machine also simulates Turing machines in polynomial time. This adds to the growing body of work [16,11,12] showing that the simplest known universal models of computation need not suffer from a exponential slowdown.

As mentioned above, the simulation of Turing machines by non-erasing Turing machines is already known to run with a polynomial slowdown [17]. However, to keep our paper self-contained, we give our own polynomial (cubic) time simulation in Section 2. This is followed by our main result in Section 3, where we show that Wang B machines simulate non-erasing Turing machines in linear time. So from Sections 2 and 3 we get Theorem 1.

Theorem 1. *Let $\mathcal{M}$ be a deterministic Turing machine with a single binary tape that runs in time t. Then there is a Wang B machine $\mathcal{W}_{\mathcal{M}}$ that simulates the computation of $\mathcal{M}$ in time $O(t^3)$.*

In Section 4 we give a formal description of Hasenjaeger's Turing machine and, for the sake of completeness, we show that Hasenjaeger's machine simulates Wang B machines in linear time. So from Theorem 1 and Section 4 we get that Hasenjaeger's machine is an efficient polynomial time simulator of Turing machines:

Theorem 2. *Let $\mathcal{M}$ be a deterministic Turing machine with a single binary tape that computes in time t. Hasenjaeger's universal Turing machine simulates the computation of $\mathcal{M}$ in time $O(t^3)$.*

[2] Heinz Nixdorf MuseumsForum, Paderborn, Germany. http://www.hnf.de/.

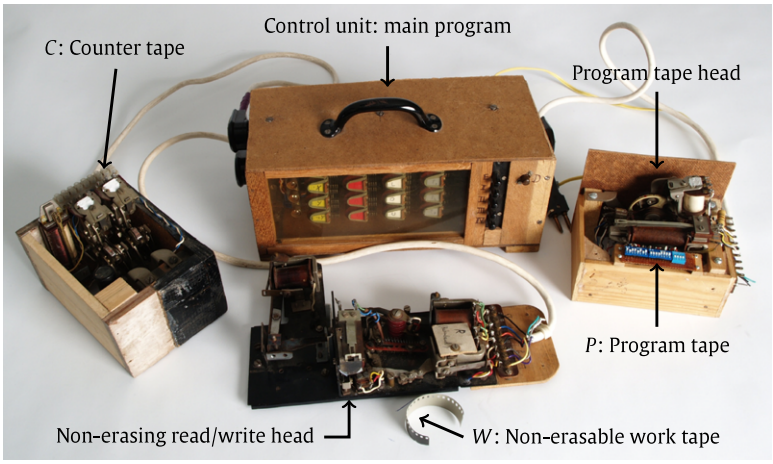

Fig. 1. Hasenjaeger's universal electromechanical Turing machine. The wiring in the control unit encodes a universal program, that uses only four states and two symbols, for simulating Wang B machines. The program of a Wang B machine may be stored on the program tape. There are two additional tapes which are used for the simulation, a counter tape and a work tape.

In Section 5 we apply Theorem 1 to show that a small universal Turing machine of Hooper's [5,6] is efficiently universal by showing that it simulates Turing machines (via Wang B machines) in polynomial time.

For the remainder of this section we discuss program-size in small universal Turing machines. Hasenjaeger's machine has 4 states and 2 symbols, making it a remarkably small universal program. However, it uses 3 non-erasable tapes, and so making direct comparisons with other Turing machine models that have small universal programs (but have more or less tapes, tape dimensions, etc.) is not a straightforward matter. The standard model in the small universal Turing machine world consists of a single one dimensional tape with one tape head, a deterministic program, and the usual notion of a blank symbol [12]. Other more general models use larger numbers of tapes, higher tape dimensions, infinitely repeated blank words instead of a repeated blank symbol, and so on, and these more general models often have smaller universal programs. In the absence of formal tools, namely tight program-size overheads for simulations between these models, comparisons between them is at best challenging. Glaschick is the most recent author to propose a formula to compare such models [2].

As an example of the difficulty of comparing different Turing machine models, consider one of Priese's [14] universal machines. Priese's universal machine has 2 states, 2 symbols, a single 2-dimensional tape with 2 tape heads, and uses an unconventional technique for ending its computation.[3] However, for standard 2-state, 2-symbol machines it is known that no universal machines exist as their halting problem is decidable [7,13]. So, by generalizing aspects of the model, Priese found a machine model whose smallest universal programs have *strictly* less states and symbols than those of the standard model. Returning our attention to Hasenjaeger's model, we note that while his machine has 3 tapes, the size of his program is still impressive when one considers that 2 tapes are read-only and the work tape is non-erasing.

For more recent results on small non-erasing Turing machines one can look to the work of Margenstern [8,9] where he constructs small non-erasing single-tape machines and gives boundaries between universality and non-universality for various parameters in the model. More on the topic of small universal Turing machines can be found in the surveys [10,12].

[3] Priese's machine does not end its computation using the standard method of halting on a state-symbol pair that has no transition rule: instead there is a choice, via the initial input encoding, of ending a computation either by entering a sequence of 6 repeating configurations or by halting when an attempt is made to move off the edge of the 2D tape.

2. Non-erasing Turing machines simulate Turing machines in time $O(t^3)$

Definition 3 (*Binary Turing Machine*)**.** A binary Turing machine is a tuple $M = (Q, \{0, 1\}, f, q_0, q_{|Q|-1})$. Here Q and $\{0, 1\}$ are the finite sets of states and tape symbols respectively. 0 is the blank symbol, $q_0 \in Q$ is the start state, and $q_{|Q|-1} \in Q$ is the halt state. The transition function f is of the form $f : Q \times \{0, 1\} \to \{0, 1\} \times \{L, R\} \times Q$ and is undefined on $\{q_{|Q|-1}\} \times \{0, 1\}$.

We write f as a list of transition rules. Each transition rule is a quintuple (q_i, x_1, x_2, D, q_j) with initial state $q_i \in Q$, read symbol $x_1 \in \{0, 1\}$, write symbol $x_2 \in \{0, 1\}$, move direction $D \in \{L, R\}$, and next state $q_j \in Q$.

Definition 4 (*Non-Erasing Turing Machine*)**.** A non-erasing Turing machine is a binary Turing machine where there are no transition rules that overwrite 1 with 0, that is, there is no transition rule of the form $(q_j, 1, 0, D, q_k)$, where $q_j, q_k \in Q$ and $D \in \{L, R\}$.

Lemma 5 (*Zykin [17]*)**.** *Let $\mathcal{M}$ be a deterministic single-tape binary Turing machine that runs in time t. Then there is a deterministic non-erasing Turing machine $\mathcal{N}_{\mathcal{M}}$ that simulates the computation of $\mathcal{M}$ in time $O(t^3)$.*

Proof. We give a brief overview of how $\mathcal{M}$ is simulated by a deterministic non-erasing Turing machine $\mathcal{N}_{\mathcal{M}}$ with a single tape in time $O(t^3)$. An arbitrary tape of $\mathcal{M}$ is encoded for $\mathcal{N}_{\mathcal{M}}$ as follows. Each symbol on the tape of $\mathcal{M}$ is encoded as three contiguous symbols on the tape of $\mathcal{N}_{\mathcal{M}}$. The two rightmost symbols of each triple encode 0 and 1 as 10 and 01 respectively. The leftmost symbol of the triple is 1 if and only if $\mathcal{N}_{\mathcal{M}}$ is simulating that $\mathcal{M}$'s tape head is currently reading the symbol encoded by the pair immediately to its right. To simulate a timestep of $\mathcal{M}$, $\mathcal{N}_{\mathcal{M}}$ simply makes a new copy of the encoded tape of $\mathcal{M}$ (to the right of the original), by scanning over and back repeatedly. During the copying process the encoded tape contents are appropriately modified to simulate the transition rule of $\mathcal{M}$. This involves simulating the tape head movement of $\mathcal{M}$ by copying the 1 that encodes the tape head position of $\mathcal{M}$ to the left of the pair of symbols encoding the new read symbol. If we are simulating a rule where $\mathcal{M}$ changes a bit under its tape head, then the encoded read symbol (i.e. the triple) is appropriately changed by $\mathcal{N}_{\mathcal{M}}$ as it is being copied. The state-changes of $\mathcal{M}$ can be simulated by state-changes of $\mathcal{N}_{\mathcal{M}}$ in a straight-forward manner.

Since $\mathcal{M}$ runs in time t, it uses at most t tape cells. Thus, $\mathcal{N}_{\mathcal{M}}$ takes $O(t^2)$ steps when copying the encoding of an arbitrary configuration of $\mathcal{M}$ to simulate a single step of $\mathcal{M}$. So t steps of $\mathcal{M}$ are simulated by $\mathcal{N}_{\mathcal{M}}$ in time $O(t^3)$. □

3. Wang B machines

A Wang B machine is a computing machine with a single non-erasing bi-infinite tape that has a binary alphabet [15]. Unlike a Turing machine, which performs three operations in a single timestep (write a 1 to its tape, move its tape head, and move program control to a arbitrary location in its program), a Wang B machine can perform only one operation at each timestep. Also, in a Turing machine, control flow can jump to an arbitrary program location when reading 0 or 1, but a Wang B machine performs a control flow jump only upon reading 1.

Definition 6 (*Wang B Machine*)**.** A Wang B machine is a finite list of instructions $\mathcal{W} = I_0, I_1, I_2, \ldots, I_{n-1}$ where each instruction is of one of the following four forms:

L:	move tape head left,
R:	move tape head right,
M:	mark the current tape cell by writing the symbol 1,
$J(x)$:	if the current cell contains the symbol 1 then jump to instruction I_x, otherwise move to the next instruction.

Instructions are executed by the machine one at a time, with the computation starting at instruction I_0. A left move or right move instruction ($I_k \in \{L, R\}$) moves the head one cell to the left or right on the tape. A mark instruction ($I_k = M$) marks the tape: if a cell is 0 (unmarked) it becomes 1 (marked), otherwise if a cell is 1 it stays as 1. For a jump instruction, $I_k = J(x)$, where $0 \leq x \leq n-1$, if the current tape cell is 1 then the machine jumps to instruction I_x. Alternatively, when $I_k = J(x)$ and the current cell is 0 the machine will either move to the next instruction I_{k+1} if $k < n-1$, or it will halt if $k = n-1$. After each move or mark instruction I_k, the machine either moves to the next instruction I_{k+1} if $k < n-1$, or halts if $k = n-1$.

3.1. Wang's B machines simulate non-erasing Turing machines in linear time

Theorem 7. *Let $\mathcal{N}$ be a deterministic non-erasing Turing machine with a single binary tape that runs in time t. Then there is a Wang B machine $\mathcal{W}_{\mathcal{N}}$ that simulates the computation of $\mathcal{N}$ in time $O(t)$.*

Proof. We begin by giving the program for the Wang B machine $\mathcal{W}_{\mathcal{N}}$ followed by the encoding it uses to simulate $\mathcal{N}$. We then show that $\mathcal{W}_{\mathcal{N}}$ simulates each transition rule in $\mathcal{N}$ in constant time, and so simulates the computation of $\mathcal{N}$ in time $O(t)$.

3.1.1. Encoding

Let $\langle TR_{q_i,\sigma_1}\rangle$ denote a sequence of Wang B machine instructions that encode the transition rule $TR_{q_i,\sigma_1} = (q_i, \sigma_1, \sigma_2, D, q_j)$ from $\mathcal{N}$ where $q_i, q_j \in Q$, $\sigma_1, \sigma_2 \in \{0, 1\}$ and $D \in \{R, L\}$. The sequence of instructions for $\mathcal{W}_{\mathcal{N}}$ is

$$\begin{aligned}\mathcal{W}_{\mathcal{N}} = {} & R, J(8), \langle TR_{q_0,0}\rangle, \langle TR_{q_0,1}\rangle,\\ & R, J(21), \langle TR_{q_1,0}\rangle, \langle TR_{q_1,1}\rangle,\\ & \qquad\vdots\\ & R, J(13i+8), \langle TR_{q_i,0}\rangle, \langle TR_{q_i,1}\rangle,\\ & \qquad\vdots\\ & R, J(13(|Q|-2)+8), \langle TR_{q_{|Q|-2},0}\rangle\langle TR_{q_{|Q|-2},1}\rangle, M\end{aligned} \tag{1}$$

where $|Q|$ is the number of states in $\mathcal{N}$, and $\langle TR_{q_i,0}\rangle$ and $\langle TR_{q_i,1}\rangle$ are the instruction sequences given by Eqs. (2) and (3).

We now define Eqs. (2) and (3) which give the sequence of instructions used to simulate each transition rule.

$$\langle TR_{q_i,0}\rangle = \begin{cases} R, M, M, M, M, J(13j) & \text{if } TR_{q_i,0} = (q_i, 0, 0, R, q_j)\\ L, L, L, M, M, J(13j) & \text{if } TR_{q_i,0} = (q_i, 0, 0, L, q_j)\\ M, R, M, M, M, J(13j) & \text{if } TR_{q_i,0} = (q_i, 0, 1, R, q_j)\\ M, L, L, L, M, J(13j) & \text{if } TR_{q_i,0} = (q_i, 0, 1, L, q_j)\end{cases} \tag{2}$$

$$\langle TR_{q_i,1}\rangle = \begin{cases} R, M, M, M, J(13j) & \text{if } TR_{q_i,1} = (q_i, 1, 1, R, q_j)\\ L, L, L, M, J(13j) & \text{if } TR_{q_i,1} = (q_i, 1, 1, L, q_j).\end{cases} \tag{3}$$

(There are only two cases for $\langle TR_{q_i,1}\rangle$ as non-erasing machines never overwrite 1 with 0.)

We encode the symbols 0 and 1 of $\mathcal{N}$ as $\langle 0\rangle = 10$ and $\langle 1\rangle = 11$ respectively. An arbitrary configuration of $\mathcal{N}$ is given by

$$q_i, \qquad w_0\, w_1\, \ldots\, w_{j-1}\, \underline{w_j}\, w_{j+1}\, \ldots\, w_{n-1} \tag{4}$$

where q_i is the current state, $w_0 \ldots w_{n-1}$ is the tape contents, $w_k \in \{0, 1\}$ and the tape head position is given by an underline. The configuration in Eq. (4) is encoded as the B machine tape

$$I_{13i}, \qquad \langle w_0\rangle\, \langle w_1\rangle\, \ldots\, \langle w_{j-1}\rangle\, \underline{\boldsymbol{w_{j_1}}\boldsymbol{w_{j_2}}}\, \langle w_{j+1}\rangle \ldots \langle w_{n-1}\rangle \tag{5}$$

where $\langle w_k \rangle \in \{\langle 0 \rangle, \langle 1 \rangle\}$, the encoded read symbol $w_{j_1} w_{j_2} = \langle w_j \rangle$ is given in bold, and I_{13i} is the next instruction to be executed and encodes that $\mathcal{N}$ is in state q_i. Note that I_{13i} is the first instruction in the sequence $I_{13i}, \ldots, I_{13i+12} = R, J(13i+8), \langle TR_{q_i,0} \rangle, \langle TR_{q_i,1} \rangle$ that encodes the pair of transition rules for state q_i.

The infinite number of blank tape cells of $\mathcal{N}$ each contain the symbol 0, as do the blank tape cells of $\mathcal{W}_{\mathcal{N}}$. Note that, during the simulation, $\mathcal{W}_{\mathcal{N}}$ may need to simulate the situation where the tape head of $\mathcal{N}$ moves to a blank tape cell. In this case, as described below, the simulator $\mathcal{W}_{\mathcal{N}}$ will move to the relevant blank portion of its own tape and convert the symbol pair 00 to $10 = \langle 0 \rangle$.

3.1.2. Simulating transition rules

At the start of each simulated timestep of machine $\mathcal{N}$, our Wang machine $\mathcal{W}_{\mathcal{N}}$ has a configuration of the form given in Eq. (5). Each simulated timestep begins with $\mathcal{W}_{\mathcal{N}}$ choosing which transition rule to simulate by reading the encoded read symbol and then choosing which sequence ($\langle TR_{q_i,0} \rangle$ or $\langle TR_{q_i,1} \rangle$) to execute.

From Eq. (5), each simulated timestep begins with the tape head over the leftmost symbol of the encoded read symbol ($\langle 0 \rangle = 10$ or $\langle 1 \rangle = 11$). So, immediately after we execute the first instruction (which is $I_{13i} = R$) the tape head is over the rightmost symbol of $\langle 0 \rangle$ or $\langle 1 \rangle$ and the program control is at instruction $I_{13i+1} = J(13i+8)$. If we are reading $\langle 0 \rangle = 10$, then the rightmost symbol is a 0 and so no jump occurs on $J(13i+8)$. This means that control moves to instruction I_{13i+2}, the leftmost instruction in $\langle TR_{q_i,0} \rangle$. Alternatively, if we are reading $\langle 1 \rangle = 11$, then the rightmost symbol is a 1 and $J(13i+8)$ will jump to instruction I_{13i+8}, sending control to the leftmost instruction of $\langle TR_{q_i,1} \rangle$. (To see this, use Eq. (1) to count the number of instructions that precede $\langle TR_{q_i,1} \rangle$, which gives $13i + 8$, specifically 13 instructions for each state q_j where $j < i$ and a further 8 instructions for the sequence $R, J(13i+8), \langle TR_{q_i,0} \rangle$.)

We now explain how the sequences in Eqs. (2) and (3) simulate the transition rules of $\mathcal{N}$.

Case 1. *Read symbol of* $\mathcal{N}$ *is* 1. As described at the beginning of Section 3.1.2, the simulation of each timestep begins with the execution of $R, J(13i+8)$. When the read symbol of $\mathcal{N}$ is 1, and the pair of instructions $R, J(13i+8)$ have executed, we have the following tape contents for $\mathcal{W}_{\mathcal{N}}$

$$\langle w_0 \rangle \langle w_1 \rangle \ \ldots \ \langle w_{j-2} \rangle \ 10 \ \underline{\mathbf{11}} \ \langle w_{j+1} \rangle \ldots \langle w_{n-1} \rangle. \tag{6}$$

(For illustration purposes, we assume that in $\mathcal{N}$ the symbol to the left of the read symbol is a 0, which is encoded as $\langle 0 \rangle = 10$ in Eq. (6).)

As described at the beginning of Section 3.1.2, when the read symbol of $\mathcal{N}$ is 1 the execution of $R, J(13i+8)$ is followed by the execution of the sequence $\langle TR_{q_i,1} \rangle$. If we are simulating $(q_i, 1, 1, L, q_j)$, then from Eq. (3) the instruction sequence $\langle TR_{q_i,1} \rangle = L, L, L, M, J(13j)$ is applied to the tape in Eq. (6) to give

$$\langle w_0 \rangle \langle w_1 \rangle \ \ldots \ \langle w_{j-2} \rangle \ \underline{\mathbf{10}} \ 11 \ \langle w_{j+1} \rangle \ldots \langle w_{n-1} \rangle. \tag{7}$$

The tape in Eq. (7) is of the form in Eq. (5), hence the tape configuration is ready for the simulation of the next timestep. The jump instruction $J(13j)$ sent the program control of $\mathcal{W}_{\mathcal{N}}$ to the first instruction of the sequence of instructions that encodes state q_j. This is verified by counting the number of instructions to the left of $R, J(13j+8), \langle TR_{q_j,0} \rangle, \langle TR_{q_j,1} \rangle$ using the same technique as above. So, the simulation of the transition rule $(q_i, 1, 1, L, q_j)$ is complete.

To generalize this example to all possible cases for simulating a rule of the form $(q_i, 1, 1, L, q_j)$ we need only consider the encoded symbol (from $\mathcal{N}$) immediately to the left of the encoded read symbol (in our analysis i, j are already arbitrary). If the encoded symbol to the left of the tape head in Eq. (6) was $\langle 1 \rangle = 11$ instead of $\langle 0 \rangle = 10$, then it is verified in the same straightforward manner. If we are simulating the situation where $\mathcal{N}$ is at the left end of its tape (the tape is blank to the left: all 0s) and so contains the pair 00 immediately to the left of the encoded read symbol in Eq. (6). This 00 pair is changed to $\langle 0 \rangle = 10$ by the M instruction that immediately proceeds the $J(13j)$ instruction, correctly providing the symbol pair 10 that encodes the 0 as $\langle 0 \rangle = 10$ for the next simulated timestep. Also, the 1 printed by this M instruction allows instruction $J(13j)$ to jump the program control to the encoding of the next state q_j.

The case of simulating $(q_i, 1, 1, R, q_j)$ is verified by applying the sequence $\langle TR_{q_i,1} \rangle = R, M, M, M, J(13j)$ from Eq. (3) to the tape in Eq. (6). This analysis is similar to the previous example and so we

omit the details. We simply note that after the first of the three M instructions is executed, the tape cell will always contain a 1 and so the second and third M instructions do not change the tape. (These extra M instructions are used for padding so that each encoded state has exactly 13 instructions.)

Case 2. *Read symbol of $\mathcal{N}$ is* 0. As described at the beginning of Section 3.1.2, the simulation of each timestep begins with the execution of $R, J(13i + 8)$. When the read symbol of $\mathcal{N}$ is 0, and the pair of instructions $R, J(13i + 8)$ have been executed, we have the following tape contents for $\mathcal{W}_{\mathcal{N}}$.

$$\langle w_0\rangle\,\langle w_1\rangle\,\ldots\,\langle w_{j-1}\rangle\,\mathbf{\underline{10}}\,11\,\langle w_{j+2}\rangle\ldots\,\langle w_{n-1}\rangle. \tag{8}$$

(For illustration purposes, we assume that in $\mathcal{N}$ the symbol to the right of the read symbol is a 1, which is encoded as $\langle 1\rangle = 11$ as in Eq. (8).)

As described at the beginning of Section 3.1.2, when the read symbol of $\mathcal{N}$ is 0 the execution of $R, J(13i+8)$ is followed by the execution of the sequence $\langle TR_{q_i,0}\rangle$. If we are simulating $(q_i, 0, 1, R, q_j)$, then from Eq. (2) the sequence $\langle TR_{q_i,0}\rangle = M, R, M, M, M, J(13j)$ is applied to the tape in Eq. (8) to give

$$\langle w_0\rangle\,\langle w_1\rangle\,\ldots\,\langle w_{j-1}\rangle\,11\,\mathbf{\underline{11}}\,\langle w_{j+2}\rangle\ldots\,\langle w_{n-1}\rangle. \tag{9}$$

The first M instruction changed $\langle 0\rangle = 10$ to $\langle 1\rangle = 11$ simulating the printing of the write symbol by $\mathcal{N}$. The tape in Eq. (9) is of the form found in Eq. (5) and is ready for the simulation of the next transition rule to begin. The jump instruction $J(13j)$ sends the program control of $\mathcal{W}_{\mathcal{N}}$ to the instruction sequence of the program that encodes state q_j. This is verified using the same technique as in the previous case. So, the simulation of $(q_i, 0, 1, R, q_j)$ is complete.

To generalize this example to all possible cases for simulating a rule of the form $(q_i, 0, 1, R, q_j)$, we need only consider the encoded symbol (from $\mathcal{N}$) immediately to the right of the encoded read symbol (in our analysis i, j are already arbitrary). If the encoded symbol to the right of the tape head in Eq. (8) was $\langle 1\rangle = 10$ instead of $\langle 1\rangle = 11$, then it is verified in the same straightforward manner. If we are simulating the situation where $\mathcal{N}$ is at the right end of its tape (the tape is blank to the right: all 0s) and so contains the pair 00 immediately to the right of the encoded read symbol in Eq. (8). This 00 pair is changed to $\langle 0\rangle = 10$ by the second M in the sequence $M, R, M, M, M, J(13j)$ which provides the symbol pair $10 = \langle 0\rangle$ that correctly encodes a 0 for the next simulated timestep. Also, the 1 printed by this M instruction allows instruction $J(13j)$ to jump the program control to the encoding of the next state q_j. As with the previous case, the extra M instructions are added for padding.

The other cases for simulating $\mathcal{N}$ reading a 0 are verified by applying the appropriate sequences from Eq. (2) to the tape in Eq. (8). The details are similar to the previous example and are omitted.

3.1.3. Halting and time complexity

When $\mathcal{N}$ enters its halt state, defined to be state $q_{|Q|-1}$ in Definition 3, then $\mathcal{W}_{\mathcal{N}}$ executes the jump instruction $J(13(|Q| - 1))$ and jumps to the rightmost instruction in Eq. (1), an M instruction. Note that in order to jump to this M instruction we must have read a 1 on the tape, and so this M instruction does not change the tape. After executing this M instruction, $\mathcal{W}_{\mathcal{N}}$ is at the end of its list of instructions and so it halts.

From Eqs. (1)–(3), exactly 13 instructions are used to encode the pair of transition rules for each state q_i of $\mathcal{N}$. Furthermore, from the above algorithm, the simulation of one of these transition rules involves the execution of at most 8 instructions (at most 8 timesteps). Thus $\mathcal{W}_{\mathcal{N}}$ simulates t steps of an arbitrary non-erasing Turing machine $\mathcal{N}$ in time $O(t)$. □

4. Hasenjaeger's electromechanical universal Turing machine

We begin this section by briefly describing the electromechanical device constructed by Hasenjaeger [4], which implements a multi-tape Turing machine. As mentioned in Section 1, Glaschick [2] reverse engineered the physical wiring of Hasenjaeger's electromechanical machine to find the Turing machine program left by the previous programmer, presumably Hasenjaeger, and with the help

of Hasenjaeger's notes saw that it simulates Wang B machines. For completeness we include a proof that this program (wiring) for Hasenjaeger's machine simulates Wang B machines in linear time.[4]

First we briefly describe Hasenjaeger's electromechanical machine, which is shown in Fig. 1.

- The *control unit* is constructed from 16 electromechanical relays which encode the *main program* (also called the state table) of the Hasenjaeger machine. This unit is limited to 4 states and operates on three tapes.
- The *program tape* (P) is a device consisting of 20 switches, 18 of which are connected, and together represent a cyclic, bi-directional, read-only binary tape with 18 cells. (This short tape can be used to store a simulated program.)
- The *counter tape* (C) consists of two selector switches that represent a bi-directional, cyclic, read-only tape with 18 cells. It represents a tape where all cells contain a 1 except for a single cell that contains a 0.
- The *work tape* (W) is a bi-directional non-erasable "infinite" tape.[5]

Hasenjaeger's electromechanical device, as wired, is an instance of a Turing machine. However, exactly what kind of Turing machine is a matter of opinion: there are a number of reasonable generalizations of this single device (machine instance) to get a general model of computation, here we give one. Formally, we write the tuple (Q, f, q_s) to denote an instance of a three-tape Turing machine of the following form. The three tapes are bi-directional and are denoted P, C and W. Each tape has alphabet $\{0, 1\}$ and blank symbol 0. Tapes P and C are read-only, while W is non-erasing (i.e. 1s cannot be overwritten with 0s). To give an instance of such a machine, we would assign values to the tuple (Q, f, q_s), where Q is a set of states, f is a transition function (or transition table), of the form $f : Q \times \{0, 1\} \times \{0, 1\} \times \{0, 1\} \to \{L, R, \_\} \times \{L, R, \_\} \times \{L, R, \_, 1\} \times Q$, and $q_s \in Q$ is the start state.

The machine works as follows. In state $q \in Q$, the machine reads a symbol from each of the tapes P, C, and W and, as dictated by f, for each tape does one of three things: move left (L), move right (R), do nothing ($\_$). However, for the tape W it has an additional fourth option of marking (M) the tape cell with the symbol 1.

Now we formally specify Hasenjaeger's machine $\mathcal{H} = (Q, f, q_s)$ as an instance of the above model. $\mathcal{H}$ has four states $Q = \{q_1, q_2, q_3, q_4\}$ and the start state is $q_s = q_1$. The function f is given as a list of transition rules in Table 1. This table of behavior is derived from the wiring of the electromagnetic relays of Hasenjaeger's device.

Lemma 8. *Let $\mathcal{W}$ be a Wang B machine that runs in time t. The multitape Turing machine $\mathcal{H}$, defined above, simulates the computation of $\mathcal{W}$ in time $O(t)$.*

Proof. We begin by giving the encoding used by the program $\mathcal{W}$, followed by a description of how the program simulates each of the four Wang B machine instructions as well as halting. We finish by giving the time analysis for this simulation.

Encoding. The four Wang B machine instructions M, R, L and $J(x)$ are encoded as binary words as follows: $\langle M\rangle = 1$, $\langle R\rangle = 01$, $\langle L\rangle = 001$, and $\langle J(x)\rangle = 0000^y1$ (the value $y \in \{0, 1, 2, \ldots\}$ will be defined later). The Wang B machine program $\mathcal{W} = I_0, I_1, \ldots, I_{n-1}$ is encoded as a single binary word via Eq. (10).

$$\langle \mathcal{W}\rangle = \langle I_0\rangle\langle I_1\rangle\langle I_2\rangle \ldots \langle I_{n-2}\rangle\langle I_{n-1}\rangle\langle J(n)\rangle. \tag{10}$$

The word $\langle \mathcal{W}\rangle \in \{0, 1\}^*$ is placed on $\mathcal{H}$'s circular program tape P. The C tape is defined to have length $n+2$, with $n+1$ of these cells containing the symbol 1, and the single remaining cell containing the symbol 0. The W tape has the same tape contents as that of the Wang B machine it simulates. At the beginning of a simulated computation step the tape head of P is over the leftmost symbol of the

4 Note that the machine can be re-programmed by re-wiring.

5 It is expected that the recent precipitous decline in the production of 35 mm film and paper punch tape will negatively impact the computing power of Hasenjaeger's machine.

Table 1
The program f for Hasenjaeger's universal machine $\mathcal{H}$ that simulates Wang's B machines. The $*$ symbol denotes that the read symbol can be 0 or 1. The rule numbers on the left are not part of the program.

Rule number	Q	P	C	W	P	C	W	Q'
1	q_1	1	$*$	0	R	–	1	q_1
2	q_1	1	$*$	1	R	–	–	q_1
3	q_1	0	$*$	$*$	R	–	–	q_2
4	q_2	1	0	$*$	R	–	R	q_1
5	q_2	0	0	$*$	R	R	–	q_2
6	q_2	1	1	$*$	R	L	L	q_1
7	q_2	0	1	$*$	R	L	–	q_3
8	q_3	0	$*$	0	R	–	–	q_3
9	q_3	1	$*$	0	R	–	–	q_1
10	q_3	0	$*$	1	R	R	–	q_3
11	q_3	1	$*$	1	L	R	–	q_4
12	q_4	0	1	$*$	L	–	–	q_4
13	q_4	1	1	$*$	L	L	–	q_4
14	q_4	$*$	0	$*$	R	–	–	q_1

encoded instruction it is simulating, C's tape head is over its single 0 symbol, and the tape head of W has the same location as the tape head of the Wang B machine it simulates.

To help simplify our explanation, we give partial configurations for $\mathcal{H}$ where we display a small part of each tape surrounding the tape head. For example, the following configuration occurs at the beginning of a simulated computation step

$$q_1 \quad P = \ldots 1\underline{0}01\ldots \quad C = \ldots 10\underline{1}\ldots \quad W = \ldots 1\underline{0}0\ldots.$$

Here, $\mathcal{H}$'s current state is q_1 and the position of each of the three tape heads is given by an underline. Also, in the above example the tape head of P is over the leftmost symbol of an encoded left move instruction $\langle L\rangle = 001$, and the C tape head is at cell C_0.

Simulate M instruction. The Wang B machine M instruction is encoded as $\langle M\rangle = 1$ on the P tape. If the tape head of W is reading a 0 then we have a configuration of the form

$$q_1 \quad P = \ldots 1\,\underline{1}\,001\ldots \quad C = \ldots 10\underline{1}\ldots \quad W = \ldots 1\underline{0}0\ldots.$$

(For the purposes of explanation, we have assumed that there is an encoded L instruction, given by $\langle L\rangle = 001$, to the right of $\langle M\rangle = 1$ on the P tape.) Rule 1 from Table 1 is applied to the above configuration to give

$$q_1 \quad P = \ldots 1\,1\,\underline{0}01\ldots \quad C = \ldots 10\underline{1}\ldots \quad W = \ldots 1\underline{1}0\ldots.$$

The M instruction was simulated by printing a 1 to the W tape. Note that the tape head on the P tape has moved to the leftmost symbol of the next encoded instruction ($\langle L\rangle = 001$), and the current state of $\mathcal{H}$ is once again q_1. So the simulation of the M instruction is complete and $\mathcal{H}$ is configured to begin simulation of the next Wang machine instruction.

In the case where the tape head of W is reading a 1, we simulate the M instruction by executing rule 2 from Table 1. This is very similar to the previous case above and so we omit the detail.

Simulate R instruction. The Wang B machine *right move* instruction is encoded as $\langle R\rangle = 01$ on the P tape. If the tape head of W is reading a 0 then we have a configuration of the form

$$q_1 \quad P = \ldots 1\,\underline{0}\,1\,001\ldots \quad C = \ldots 10\underline{1}\ldots \quad W = \ldots 1\underline{0}0\ldots.$$

Rules 3 and 4 from Table 1 are applied to the above configuration to give

$$q_1 \quad P = \ldots 1\,01\,\underline{0}01\ldots \quad C = \ldots 10\underline{1}\ldots \quad W = \ldots 10\underline{0}\ldots.$$

The tape head of W was moved one place to the right to simulate the R instruction. Also, the tape head on the P tape has moved right 2 places to the leftmost symbol of the next encoded instruction ($\langle L\rangle = 001$), and the current state of $\mathcal{H}$ is once again q_1. So the simulation of the R instruction is

complete and $\mathcal{H}$ is configured to begin simulation of the next Wang machine instruction. In the case where the tape head of W is reading a 1, the computation proceeds in the same manner as above by executing rules 3 and 4 from Table 1.

Simulate L instruction. The Wang B machine *left move* instruction is encoded as $\langle L\rangle = 001$ on the P tape. If the tape head of W is reading a 0 then we have a configuration of the form

$$q_1 \quad P = \ldots 1\underline{0}01\,01\ldots \quad C = \ldots 1\underline{0}1\ldots \quad W = \ldots 1\underline{0}0\ldots.$$

Rules 3, 5 and 6 from Table 1 are applied to the above configuration to give

$$q_1 \quad P = \ldots 1\,001\,\underline{0}1\ldots \quad C = \ldots 1\underline{0}1\ldots \quad W = \ldots \underline{1}00\ldots.$$

The tape head of W was moved one place to the left to simulate the L instruction. Also, the tape head on the P tape has moved right 3 places to the leftmost symbol of the next encoded instruction ($\langle R\rangle = 01$), and the current state of $\mathcal{H}$ is once again q_1. So the simulation of the L instruction is complete and $\mathcal{H}$ is configured to begin simulation of the next Wang machine instruction. In the case where the tape head of W is reading a 1, the computation proceeds in the same manner as above by executing rules 3, 5 and 6 from Table 1.

Simulate $I_k = J(x)$ instruction. There are two cases to consider here, which are determined by the value of read symbol of the simulated Wang B machine.

Case 1. *Wang B machine's read symbol is* 0. In this case, $\mathcal{H}$ simulates program control for $\mathcal{W}$ moving from instruction I_k to instruction I_{k+1}. This is simulated by moving the tape head to the leftmost symbol of $\langle I_{k+1}\rangle$. Instruction $I_k = J(x)$ is encoded as $\langle I_k\rangle = \langle J(x)\rangle = \underline{0}000^y1$ for some $y \in \{0, 1, 2, \ldots\}$ (y is defined below), and for the purposes of explanation we assume that $I_{k+1} = L$. This gives the configuration

$$q_1 \quad P = \ldots 1\,\underline{0}000^y1\,001\ldots \quad C = \ldots \underline{0}1\ldots \quad W = \ldots \underline{0}\ldots.$$

After applying rules 3, 5 and 7 from Table 1 we get the following

$$q_3 \quad P = \ldots 1\,000\underline{0}0^{y-1}1\,001\ldots \quad C = \ldots \underline{0}1\ldots \quad W = \ldots \underline{0}\ldots.$$

Next, rule 8 is applied y times followed by a single application of rule 9 to give

$$q_1 \quad P = \ldots 1\,0000^y1\,\underline{0}01\ldots \quad C = \ldots \underline{0}1\ldots \quad W = \ldots \underline{0}\ldots.$$

In the configuration immediately above, the simulation of $J(x)$ when the Wang machine read symbol is 0 is complete. Note that $\mathcal{H}$ has returned to state q_1 and the tape head of P is over the leftmost symbol of the encoded instruction $\langle I_{k+1}\rangle = 001$.

Case 2. *Wang B machine read symbol is* 1. In this case, simulating the instruction $I_k = J(x)$ involves moving the P tape head to the leftmost symbol of $\langle I_x\rangle$.

We begin with an overview, which includes specifying the encoding of jump instructions. Each encoded instruction contains a single 1 symbol, and so as we move through the P tape we can count the number of encoded instructions by counting the number of 1 symbols. If $x \leqslant k$, then, from Eq. (10), we can move from $\langle I_k\rangle$ to $\langle I_x\rangle$ on P by moving left until we have read the symbol 1 exactly $k - x + 1$ times, and then moving right. Recall that the P tape is circular, and so if $x > k$, using Eq. (10), we move from $\langle I_k\rangle$ to $\langle I_x\rangle$ on P by moving left until we have read the 1 symbol exactly $(n + 2 + k - x)$ times, and then moving right. We are now ready to give the encoding for jump instructions.

$$\langle I_k\rangle = \langle J(x)\rangle = \underline{0}000^y1$$

where

$$y = \begin{cases} k - x & \text{if } x \leqslant k \\ n + 1 + k - x & \text{if } x > k. \end{cases} \tag{11}$$

In the simulation, moving from $\langle I_k\rangle$ to $\langle I_x\rangle$ is done in 2 stages. In the first stage the word $\langle J(x)\rangle = \underline{0}000^y1$ is read and the value $y + 1$ is recorded by the tape head position on the C tape. In the second stage, using the value stored on the C tape, the tape head of P moves left until we have read the symbol 1 exactly $y + 1$ times. So, the tape head of P finishes its scan left immediately to the left of the 1 in $\langle I_{x-1}\rangle$; from there it moves right two cells to the leftmost symbol of $\langle I_x\rangle$.

Now we give the details of how $\mathcal{H}$ simulates a jump from instruction I_k to instruction I_x. For the purposes of illustration we assume the instruction to the left of I_k is $I_{k-1} = L$. This gives the configuration

$$q_1 \quad P = \ldots 001\,\underline{0}000^y 1\ldots \quad C = \ldots \underline{0}1^{y+1}\ldots \quad W = \ldots \underline{1}\ldots.$$

First, rules 3, 5 and 7 from Table 1 are applied, and then rule 10 is applied y times, followed by a single application of rule 11, to give

$$q_4 \quad \ldots P = 001\,0000^{y-1}\underline{0}1\ldots \quad C = \ldots 01^y\underline{1}\ldots \quad W = \ldots \underline{1}\ldots.$$

In the configuration immediately above the value $y+1$ is recorded by the position of the tape head of C, which is over $y+1$th symbol to the right of the single 0 symbol. Rule 12 is applied $y+3$ times to give

$$q_4 \quad \ldots P = 00\underline{1}\,0000^y 1\ldots \quad C = \ldots 01^y\underline{1}\ldots \quad W = \ldots \underline{1}\ldots.$$

When 1 is read on tape P the value stored on tape C is decremented by moving left once on C using rule 13. This gives

$$q_4 \quad \ldots P = 0\underline{0}1\,0000^y 1\ldots \quad C = \ldots 01^{y-1}\underline{1}1\ldots \quad W = \ldots \underline{1}\ldots.$$

The above process of decrementing the value stored in C by applying rules 12 and 13 continues until the tape head of C reads a 0, indicating that the scan left is finished (during this process Rule 13 is applied a total of $y+1$ times). At this point we have a configuration that is of one of the following two forms

$$q_4 \quad P = \ldots \underline{0}1\ldots \quad C = \ldots \underline{0}1\ldots \quad W = \ldots \underline{1}\ldots$$
$$q_4 \quad P = \ldots \underline{1}1\ldots \quad C = \ldots \underline{0}1\ldots \quad W = \ldots \underline{1}\ldots.$$

Rule 13 was applied $y+1$ times reading a 1 each time. Rules 14 and 2 are applied to move the tape head of P right twice, placing it over the leftmost symbol of instruction $\langle I_x\rangle$ to complete the simulation of $I_k = J(x)$.

Simulation of halting. Recall from Section 3, that a Wang B machine halts when it attempts to move to the non-existent instruction I_n after executing instruction I_{n-1}. Since $\mathcal{H}$ does not have a distinguished halt state, it instead simulates halting by entering a repeating sequence of configurations. Note that in Eq. (10), as part of the Wang B machine encoding, there is an extra instruction ($I_n = J(n)$) that jumps to itself. So when program $\mathcal{H}$ simulates a Wang B machine that halts by attempting to move to instruction I_n, the program simulates the instruction $J(n)$ which results in an infinite loop and signals the end of the simulation. (The jump instruction works as intended only if we have the assumption that the cell under W's tape head reads 1; it is easy to modify any Wang B program so that this is the case by having the program end with a single mark instruction, i.e. $I_{n-1} = M$.) This jump works as follows. From Eq. (11), a jump instruction of the form $I_n = J(n)$ is encoded as $\langle I_n\rangle = \langle J(n)\rangle = 0001$. This gives the configuration

$$q_1 \quad P = \ldots \underline{0}001\ldots \quad C = \ldots \underline{0}1\ldots \quad W = \ldots \underline{1}\ldots. \tag{12}$$

From here, $\mathcal{H}$ simulates the jump instruction $I_n = J(n)$, as described above. In this simple case, simulating the jump instruction involves executing exactly 10 rules (see Table 1) after which $\mathcal{H}$ returns to configuration (12). Hence we get an infinite loop where the tape contents are unchanged.

Complexity analysis. The Wang B machine instructions M, R, and L are each simulated by Hasenjaeger's machine in 1, 2 and 3 timesteps, respectively. The $J(x)$ instruction is simulated in $O(n^2)$ timesteps, where n is the number of instructions in the Wang B machine program. Note that we consider n to be a constant, independent of the input length. Therefore, Hasenjaeger's program $\mathcal{H}$ simulates t steps of the Wang B machine $\mathcal{W}$ in time $O(t)$. □

5. Hooper's small universal Turing machine simulates Turing machines in polynomial time

Hooper [5,6] gave a small universal Turing machine with 1 state, 2 symbols and 4 tapes. Using similar techniques to Hasenjaeger, Hooper proved his machine universal by simulating a restricted

class of Wang B machines. In Hooper's machine, a non-erasing work tape contains exactly the same contents as the tape of the Wang B machine it simulates, and a read-only unidirectional circular program tape stores the encoded Wang machine program. Hooper used a relative addressing technique like Hasenjaeger, but unlike Hasenjaeger, Hooper used two read–write counter tapes (instead of one read-only tape). Hooper's simulation of Wang B machines runs in linear time, and so from Lemma 5 and by suitably modifying the proof of Theorem 7 we get the following result.

Theorem 9. *Let $\mathcal{M}$ be a deterministic Turing machine with a single binary tape that runs in time t. Then Hooper's small universal Turing machine with* 1 *state,* 2 *symbols, and* 4 *tapes* [5,6] *simulates the computation of $\mathcal{M}$ in time $O(t^3)$.*

Proof. Hooper's machine simulates Wang B machines with the following restrictions:

1. In the program list if $I_k = J(x)$, then $I_{k+1} \in \{L, R\}$.
2. Each jump instruction jumps to $\{L, R\}$.
3. M instructions are executed only on tape cells that contain 0.

Our proof of Theorem 7 is easily modified to include the above restrictions. For restriction 1, we add the instruction sequence L, R after each jump instruction in the program. This has no effect on the program as a move left followed by a move right has the same effect as no move. Our proof already satisfies restriction 2, as we either jump to the beginning of the sequence encoding a state q_i (that is: $R, J(13i + 8), \langle TR_{q_i,0}\rangle, \langle TR_{q_i,1}\rangle$) or we jump to the beginning of a sequence of the form $\langle TR_{q_i,1}\rangle$ (given in Eq. (3)). To satisfy restriction 3, each $I_k = M$ instruction is replaced with the sequence $J(k + 4), R, L, M, R, L$. The $J(k + 4)$ will jump over the M instruction if the cell already contains a 1, and the extra R, L instructions are introduced to satisfy restrictions 1 and 2.

In addition to the above changes, we wish to maintain the property from the proof of Theorem 7 that the number of instructions used to encode each Turing machine state is the same for all states. Recall from Theorem 7 that each state q_i is encoded as the sequence of 13 instructions $R, J(13i + 8), \langle TR_{q_i,0}\rangle, \langle TR_{q_i,1}\rangle$. This sequence has 3 jump instructions and to satisfy restriction 1 we added the extra instruction pair L, R for each jump. For restriction 3, we replaced each M instruction with $J(k + 4), R, L, M, R, L$. In Eq. (3) this gives an extra 15 instructions for the case $(q_i, 1, 1, R, q_j)$ and an extra 5 for the case $(q_i, 1, 1, L, q_j)$. To ensure that the instruction sequence is the same length for each case we append the length-10 sequence $(L, R,)^5$ to the sequence for case $(q_i, 1, 1, L, q_j)$. Satisfying restriction 3 in Eq. (2) gives an extra 20 instructions for the cases $(q_i, 0, 0, R, q_j)$ and $(q_i, 0, 1, R, q_j)$, and an extra 10 for cases $(q_i, 0, 0, L, q_j)$ and $q_i, 0, 1, L, q_j$. To ensure that the instruction sequence is the same length for each case we append the length-10 sequence $(L, R,)^5$ to the sequences for case $(q_i, 0, 0, L, q_j)$ and case $(q_i, 0, 1, L, q_j)$. Now the length of the sequence that encodes each state is 54 (instead of 13), and so we replace jumps of the from $J(13i)$ with jumps of the from $J(54i)$. The sequence $R, J(13i+8), \langle TR_{q_i,0}\rangle$ of length 8 has been replaced by a sequence of length 32, and so we replace jumps of the from $J(13i + 8)$ with jumps of the form $J(54i + 32)$. This completes our conversion to a Wang B machine with the 3 restrictions mentioned above. □

Acknowledgments

We thank Benedikt Löwe for inviting us to the Isaac Newton Institute for Mathematical Sciences in Cambridge (UK) to participate in the 2012 program "Semantics and Syntax: A Legacy of Alan Turing", and for facilitating this fun project. Finally, we thank David Soloveichik for translating the work of Zykin [17].

Turlough Neary was supported by Swiss National Science Foundation grant 200021-141029. Damien Woods was supported by the USA National Science Foundation under grants 0832824 (The Molecular Programming Project), CCF-1219274, and CCF-1162589. Niall Murphy was supported by a PICATA Young Doctors fellowship from CEI Campus Moncloa, UCM-UPM, Madrid, Spain. Rainer Glaschick was supported by the Heinz Nixdorf MuseumsForum, Germany.

References

[1] R. Glaschick, Alan Turings Wirkung in Münster, Mitt. DMV 20 (2012) 42–49 (in German).
[2] R. Glaschick, A Size Index for Multitape Turing Machines, Technical Report NI12061-SAS, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK, 2012.
[3] R. Glaschick, Turing machines in Münster, in: Alan Turing—His Work and Impact, Elsevier, ISBN: 9780123869807, 2013, pp. 71–77.
[4] G. Hasenjaeger, On the early history of register machines, in: E. Börger (Ed.), Computation Theory and Logic, Springer-Verlag, London, UK, 1987, pp. 181–188.
[5] P. Hooper, Some Small, Multitape Universal Turing Machines, Tech. Rep., Computation Laboratory, Harvard University, Cambridge, Massachusetts, 1963.
[6] P. Hooper, Some small, multitape universal Turing machines, Inform. Sci. 1 (2) (1969) 205–215.
[7] M. Kudlek, Small deterministic Turing machines, Theoret. Comput. Sci. 168 (2) (1996) 241–255.
[8] M. Margenstern, Non-erasing Turing machines: a frontier between a decidable halting problem and universality, in: Z. Ésik (Ed.), FCT, in: LNCS, vol. 710, Springer, Heidelberg, Szeged, Hungry, 1993, pp. 375–385.
[9] M. Margenstern, The laterality problem for non-erasing Turing machines on {0, 1} is completely solved, RAIRO Theor. Inform. Appl. 31 (2) (1997) 159–204.
[10] M. Margenstern, Frontier between decidability and undecidability: a survey, Theoret. Comput. Sci. 231 (2) (2000) 217–251.
[11] T. Neary, D. Woods, *P*-completeness of cellular automaton rule 110, in: ICALP 2006, Part I, in: LNCS, vol. 4051, Springer, Venice, Italy, 2006, pp. 132–143.
[12] T. Neary, D. Woods, The complexity of small universal Turing machines: a survey, in: SOFSEM 2012: Theory and Practice of Computer Science, in: LNCS, vol. 7147, Springer, 2012, pp. 385–405.
[13] L. Pavlotskaya, Solvability of the halting problem for certain classes of Turing machines, Math. Notes (Springer) 13 (6) (1973) 537–541; Translated from Mat. Zametki 13 (6) (1973) 899–909.
[14] L. Priese, Towards a precise characterization of the complexity of universal and non-universal Turing machines, SIAM J. Comput. 8 (4) (1979) 508–523.
[15] H. Wang, A variant to Turing's theory of computing machines, J. ACM 4 (1) (1957) 63–92.
[16] D. Woods, 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), IEEE, Berkeley, California, 2006, pp. 439–446.
[17] G.P. Zykin, Remarks about a theorem of Hao Wang, Algebra Logika 2 (1963) 33–35 (in Russian).