A Universal Reversible Turing Machine that Directly Simulates Reversible Counter Machines

Kenichi Morita

Abstract

We construct a 1-tape 98-state 10-symbol universal reversible Turing machine (URTM(98,10)) that directly simulates reversible counter machines (RCMs). The objective of this construction is not to minimize the numbers of states and tape symbols, but to give a URTM a reasonable size whose simulating processes of RCMs are easily understood. Here, we choose RCMs as the target machines of simulation, since the class of RCMs is known to be Turing universal, and their operations are very simple. Furthermore, using the framework of RCMs in the program form (rather than the quadruple form), construction of a URTM is simplified. We also created a computer simulator for the URTM(98,10), by which simulation processes of RCMs are visualized.

Keywords: reversible computing, reversible Turing machine, universal Turing machine, reversible counter machine

MSC 2020: 68R04, 03D10

1 Introduction

A *universal Turing machine* (UTM) is a Turing machine (TM) that can simulate any TM (or Turing-universal computing system) by giving a description of the latter one on its tape. A UTM was first proposed by Turing himself [1]. Since then it has been attracting many researchers. In particular, UTMs of small sizes have been extensively studied (see a survey paper [2] on the history of the studies). Let UTM(m,n) denote an m-state n-symbol UTM. Then the problem is to find a UTM(m,n) having a small value of $m \times n$.

doi:10.56415/csjm.v32.22

In the early stage of the history, a direct simulation method of a TM was used to compose a UTM. Namely, if a description of a TM is given on the tape, the UTM simulates the former TM step by step. For example, Watanabe [3] composed a UTM(8,5) that directly simulates 2-symbol TMs. Later, to reduce the size of a UTM, an indirect simulation method was introduced. Cocke and Minsky [4] proposed a 2-tag system, a kind of a string rewriting system that can simulate a TM. Thus, it is a Turing-universal system. Then Minsky [5] designed a UTM(7,4) that can simulate 2-tag systems. Using 2-tag systems, Rogozhin [6] gave small UTM(m,n)'s for several pairs of m and n. They are UTM(24,2), UTM(10,3), UTM(7,4), UTM(5,5), UTM(4,6), UTM(3,10), and UTM(2,18). Some of these results were later improved by using a 2-tag system or its variant called a bi-tag system. They are UTM(3,9) by Kudlek and Rogozhin [7], and UTM(15,2), UTM(9,3), UTM(6,4), and UTM(5,5) by Neary and Woods [8].

A reversible Turing machine (RTM) is a standard model in reversible computing [9], [10]. Bennett [10] first showed that any (irreversible) TM can be converted into a 3-tape RTM that simulates the former, and produces no garbage information when it halts. Since a 3-tape RTM can be further converted into a 1-tape RTM [9], the class of 1-tape RTMs is Turing-universal.

A universal reversible Turing machine (URTM) is an RTM that can simulate any TM. Of course, an URTM can be obtained from an irreversible UTM by using the conversion method of Bennett [10]. However, if we do so, the numbers of states and symbols of a 1-tape URTM will become very large. Therefore, we should look for a better construction method of a URTM.

Let URTM(m,n) denote an m-state n-symbol URTM. A small URTM was first given by Morita and Yamaguchi [11]. They composed a URTM(17,5) by simulating a cyclic tag system (CTS). A CTS is a kind of a tag system proposed by Cook [12], where rewriting rules are applied cyclically. Cook proved that the class of CTSs is Turing-universal, and showed universality of the elementary cellular automaton with the rule number 110. Thus, the URTM(17,5) indirectly simulates TMs. Later, Morita gave URTM(15,6) in [13], URTM(24,4) and URTM(32,3) in [14], and URTM(13,7) and URTM(10,8) in [15] using CTSs.

The reason why CTSs were used in the above studies is that rewriting rules are applied one by one in the given order. Hence, it is easily performed by a URTM having *one 1-dimensional tape*. Even at the end of the rule sequence, the URTM can continue the cyclic application of rules by simply going back to the first rule. On the other hand, if a URTM simulates a 2-tag system, or a computing machine (like a TM) directly, it must perform some kind of "jump." Here, a jump means that a URTM goes from one address (or a state) to another address (another state) in the description of a simulated system. Assume a URTM jumps from a source address a_s to a destination address a_d. Since a simulated system can be arbitrarily large, the distance between a_s and a_d can also be arbitrarily large. Hence, the destination address a_d must be kept near the address a_s to jump correctly. Since the URTM is reversible, the source address a_s also must be kept near the destination address a_d. Otherwise, we cannot trace back the movement of the URTM. In fact, jumping from a_s to a_d must be carried out by using both the value a_d kept near the address a_s and the value a_s kept near a_d. Therefore, in the case of simulating 2-tag systems or computing machines directly, implementation of such a jumping in a URTM is one of the key points (though the above argument is informal). In Sect. 3.2, one solution for this *reversible jump problem* will be given.

Note that if the tape of a URTM is 2-dimensional, then the reversible jump problem is easily solved. On a 2-dimensional tape, we can write a *path* that connects the addresses a_s and a_d beforehand as a part of a code (like Fig. 2). Therefore, the URTM can easily perform a jump operation simply tracing the path by its head on a 2-dimensional tape. In fact, when constructing universal reversible 2-dimensional cellular automata (CAs), such a method of preparing signal paths was taken [16],[17]. As another possibility, Axelsen and Glück [18] proposed a method of using three 1-dimensional tapes for a URTM to simulate RTMs directly. However, since no complete move table is given, it is not clear how state transitions are correctly performed reversibly.

In this paper, we propose a 1-tape URTM(98,10) T_U that directly simulates any reversible counter machine (RCM) M. Here, "directly simulates" means that every configuration in M's computing process appears on the T_{U}'s tape in an encoded form, *i.e.*, T_{U} simulates M step

by step. The objective of this study is not to minimize the size of a URTM. We give a URTM having a reasonable size where descriptions of RCMs and their simulation processes are easily understood. Using T_{U}, we explain how the reversible jump problem is solved.

A counter machine (CM) is a computing model consisting of a finite control and a finite number of counters in which non-negative integers are stored [5]. Morita [19] showed that the class of reversible CMs (RCMs) is Turing-universal. Alhazov, Verlan, and Freund studied RCMs, and gave several universal RCMs (URCMs) [20]. Here, we use RCMs as "target machines" that a URTM simulates. The reasons why we choose RCMs rather than RTMs as target machines are as follows. First, the class of RCMs is Turing-universal. Second, an RCM is somewhat similar to an RTM, but its operations are simpler than those of an RTM. Third, using an RCM in the program form (rather than the quadruple form), construction of a URTM is simplified.

In this study, we created a simulator of the URTM T_{U} that works on the general purpose CA simulator *Golly* developed by Trevorrow, Rokicki, Hutton *et al.* [21]. Though Golly is mainly used for simulating CAs, it is also useful for simulating computing systems other than CAs. This is because it can deal with very large configurations, and its simulation speed is very high. The simulator of the URTM T_{U} implemented on Golly is available in [22], by which computing processes of T_{U} are visualized.

This paper is organized as follows. Section 1 is an introduction. In Sect. 2, definitions on RTMs and RCMs are given, and their basic properties are explained. In Sect. 3, the URTM T_{U} is proposed, and how it works is described. In Sect. 4, a simulator of T_{U} implemented on Golly is explained. Section 5 gives concluding remarks, where future research topics are given.

2 Reversible Turing Machines and Reversible Counter Machines

In the following, we construct a universal reversible Turing machine (URTM) that can simulate reversible counter machines. Below we give

their definitions and some known properties on them.

2.1 Reversible Turing Machines (RTMs)

A 1-tape Turing machine (TM) consists of a finite control, a read-write head, and a tape divided into squares in which symbols are written. Here we assume the tape is one-way (rightward) infinite.

Definition 1. A *1-tape Turing machine* (TM) is defined by

$$T = (Q, S, q_0, F, s_0, \delta),$$

where Q is a non-empty finite set of states, S is a non-empty finite set of tape symbols, q_0 is an *initial state* ($q_0 \in Q$), F is a set of *final states* ($F \subseteq Q$), and s_0 is a special *blank symbol* ($s_0 \in S$). Here, δ is a move relation, which is a subset of $(Q \times S \times S \times \{\mathrm{L}, \mathrm{N}, \mathrm{R}\} \times Q)$. The symbols L, N, and R are *shift directions* of the head, which stand for left-shift, no-shift, and right-shift, respectively. Each element of δ is a *quintuple* of the form $[p, s, s', d, q]$, which is called a *rule* of T. It means if T reads the symbol s in the state p, then write s', shift the head to the direction d, and go to the state q. We assume each state $q_f \in F$ is a *halting state*, *i.e.*, there is no quintuple of the form $[q_f, s, s', d, q]$ in δ.

Determinism and reversibility of a TM is defined as below.

Definition 2. Let $T = (Q, S, q_0, F, s_0, \delta)$ be a TM. We call T a *deterministic TM*, if the following holds for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ.

$$(p_1 = p_2) \ \Rightarrow \ (s_1 \neq s_2).$$

It means that for any pair of distinct rules, if the present states are the same, the read symbols are different.

In the following, we consider only deterministic TMs, and thus the term "deterministic" is omitted.

Definition 3. Let $T = (Q, S, q_0, F, s_0, \delta)$ be a TM. We call T a *reversible TM* (RTM), if the following holds for any pair of distinct quintuples $[p_1, s_1, t_1, d_1, q_1]$ and $[p_2, s_2, t_2, d_2, q_2]$ in δ.

$$(q_1 = q_2) \ \Rightarrow \ (d_1 = d_2 \ \wedge \ t_1 \neq t_2).$$

It means that for any pair of distinct rules, if the next states are the same, the shift directions are the same, and the written symbols are different. The above is called the *reversibility condition* for TMs.

An instantaneous description (ID) of a TM is an expression to describe its finite computational configuration such that the non-blank part of its tape is finite.

Definition 4. Let $T = (Q, S, q_0, F, s_0, \delta)$ be a TM. We assume $Q \cap S = \emptyset$. An *instantaneous description* (ID) of T is a string of the form $\alpha q \beta$, where $q \in Q$ and $\alpha, \beta \in S^*$. Let λ denote the *empty string*. The ID $\alpha q \beta$ describes the *finite computational configuration* of T where the content of the tape is $\alpha\beta$ (the remaining infinite part of the tape contains only blank symbols), and T is reading the leftmost symbol of β (if $\beta \neq \lambda$) or s_0 (if $\beta = \lambda$) in the state q. An ID $\alpha q_0 \beta$ is called an *initial ID*. An ID $\alpha q \beta$ is called a *final ID* if $q \in F$.

The *transition relation* among IDs of T is denoted by $\vdash_T$. Let $\alpha q \beta$ and $\alpha' q' \beta'$ be two IDs. If $\alpha' q' \beta'$ is obtained from $\alpha q \beta$ by applying a rule in δ of T, we write $\alpha q \beta \vdash_T \alpha' q' \beta'$. For example, if $[q, s, s', R, q'] \in \delta$ and $\alpha, \beta \in S^*$, then $\alpha q s \beta \vdash_T \alpha s' q' \beta$. See Sect. 5.1.1.3 of [9] for the precise definition of the transition relation.

Bennett [10] first showed that any (irreversible) TM can be simulated by a garbage-less RTM. Hence, the class of RTMs is computationally universal. See [9] for computational universality of some restricted classes of RTMs.

2.2 Reversible Counter Machines (RCMs)

A k-counter machine (CM(k)) is defined as a kind of multi-tape Turing machine as shown in Fig. 1. The tapes are read-only ones, and one-way infinite. The leftmost square of a tape contains the symbol Z, while all the other squares contain P. Therefore, if the machine reads the symbol Z (P, respectively), then it knows the content of the counter is zero (positive). The increment and decrement operations on a counter are performed by shifting the corresponding head. In [9], [19], a CM is defined in the quadruple form, but here we use the program form given in [16].

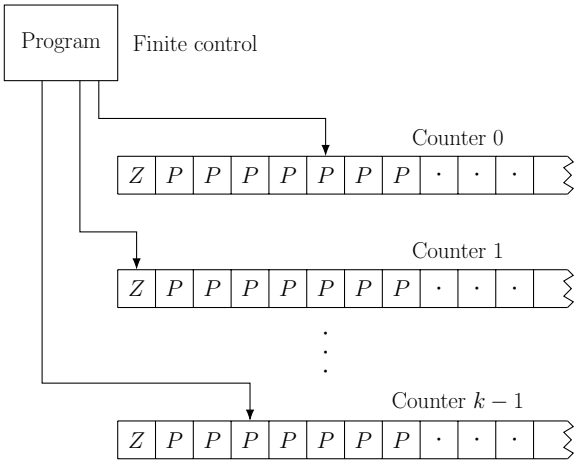


Figure 1. k-counter machine (CM(k))

There are five kinds of *instructions* for a CM(k), where b_0, b_1, m_0, and m_1 are addresses of instructions, and $i \in \{0, \ldots, k-1\}$. Intuitive meanings of the instructions are as follows.

I_i	Increment the i-th counter
D_i	Decrement the i-th counter
$B_i(b_0, b_1)$	Branch on the contents of the i-th counter, *i.e.*, if the i-th counter is 0, go to b_0, else go to b_1
$M_i(m_0, m_1)$	Merge on the contents of the i-th counter, *i.e.*, if the i-th counter is 0, merge from m_0, else from m_1
H	Halt

To define a program for a CM(k), we give the sets $A^L, A^L_{\mathrm{R}}, \mathbf{B}^L_k$, and $\mathbf{M}^L_k$ as follows, where L (> 0) is the length of a program.

$$
\begin{array}{lcl}
A^L & = & \{0, 1, \ldots, L-1\} \\
A^L_{\mathrm{R}} & = & \{1, \ldots, L-1\} \cup \{-1, \ldots, -L+1\} \\
\mathbf{B}^L_k & = & \{B_i(b_0, b_1) \mid b_0, b_1 \in A^L_{\mathrm{R}} \cup \{\#\},\ i \in \{0, \ldots, k-1\}\} \\
\mathbf{M}^L_k & = & \{M_i(m_0, m_1) \mid m_0, m_1 \in A^L_{\mathrm{R}} \cup \{\#\},\ i \in \{0, \ldots, k-1\}\}
\end{array}
$$

Here, A^L is the set of *addresses* of instructions, where the 0th instruction has the address 0, and the last has $L-1$. A^L_{R} is the set of *relative addresses*, by which destination and source addresses of B_i and

M_i instructions are specified. The set $\mathbf{B}_k^L$ ($\mathbf{M}_k^L$, respectively) contains all possible $B_i(b_0, b_1)$ instructions ($M_i(m_0, m_1)$ instructions), where # means no address is specified. If $b_p \in A_{\mathrm{R}}^L$ ($m_p \in A_{\mathrm{R}}^L$, respectively) for $p \in \{0, 1\}$, it is called a *destination address* (*source address*) of *port* p of the instruction. The set $\mathbf{S}_k^L$ of instructions, which is for a program of length L of CM(k), is as follows.

$$\mathbf{S}_k^L \;=\; \{I_i, D_i \mid i \in \{0, \ldots, k-1\}\} \cup \mathbf{B}_k^L \cup \mathbf{M}_k^L \cup \{H\}$$

Note that, in [16], source and destination addresses are specified by *absolute addresses*, while they are specified by *relative addresses* here.

Definition 5. A *well-formed program* (WFP) P of length L for CM(k) is a mapping $P : A^L \to \mathbf{S}_k^L$ that satisfies the following constraints.

(C1) The last instruction must be H or B_i instruction:

$$P(L-1) \in \{H\} \cup \mathbf{B}_k^L$$

(C2) The 0th instruction must not be M_i instruction, and the instruction just before M_i must be H or B_i instruction:

$$P(0) \notin \mathbf{M}_k^L \wedge \forall a \in A^L - \{0\} (P(a) \in \mathbf{M}_k^L \Rightarrow P(a-1) \in \{H\} \cup \mathbf{B}_k^L)$$

(C3) If the instruction of the address a is B_i, and its port p has a destination address $b_p (\neq \#)$, then the instruction at the address $a + b_p$ must be M_i, and its port p has the source address $-b_p$:

$$\begin{array}{l} \forall a \in A^L,\ \forall p \in \{0,1\},\ \forall i \in \{0, \ldots, k-1\}, \\ \forall b_0, b_1 \in A_{\mathrm{R}}^L \cup \{\#\},\ \exists m_0, m_1 \in A_{\mathrm{R}}^L \cup \{\#\} \\ \quad ((P(a) = B_i(b_0, b_1) \wedge b_p \neq \#) \\ \qquad\qquad \Rightarrow (P(a + b_p) = M_i(m_0, m_1) \wedge m_p = -b_p)) \end{array}$$

(C4) If the instruction of the address a is M_i, and its port p has a source address $m_p (\neq \#)$, then the instruction at the address $a + m_p$ must be B_i, and its port p has the destination address $-m_p$:

$$\begin{array}{l} \forall a \in A^L,\ \forall p \in \{0,1\},\ \forall i \in \{0, \ldots, k-1\}, \\ \forall m_0, m_1 \in A_{\mathrm{R}}^L \cup \{\#\},\ \exists b_0, b_1 \in A_{\mathrm{R}}^L \cup \{\#\} \\ \quad ((P(a) = M_i(m_0, m_1) \wedge m_p \neq \#) \\ \qquad\qquad \Rightarrow (P(a + m_p) = B_i(b_0, b_1) \wedge b_p = -m_p)) \end{array}$$

The constraint (C1) prevents the case of going to the address L. The constraint (C2) guarantees that each M_i instruction is activated only by B_i instructions. The constraints (C3) and (C4) say that the destination addresses of port p of B_i instructions, and the source addresses of port p of M_i instructions have one-to-one correspondence for each $p \in \{0, 1\}$.

Example 1. Let P_{twice} be the following sequence of instructions.

$B_1(1,\#)$	$M_1(-1,6)$	$B_0(6,1)$	$M_0(\#,-1)$	D_0	I_1	I_1	$B_1(\#,-6)$	$M_0(-6,\#)$	H
0	1	2	3	4	5	6	7	8	9

It is easy to see that P_{twice} satisfies the constraints (C1) – (C4) in Definition 5. Therefore, it is a well-formed program (WFP) of a CM(2). We often draw a WFP in a graphical form as in Fig. 2.

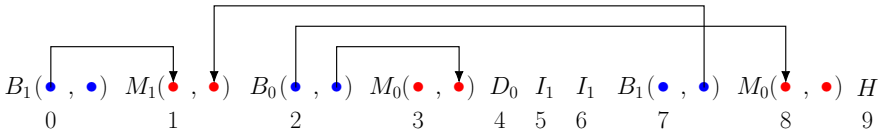


Figure 2. Graphical representation of the WFP P_{twice}

We now define a CM M in the program form, which has a WFP.

Definition 6. A *CM(k) in the program form* is defined by

$$M = (P, k, A_F),$$

where P is a WFP of length L, k is the number of counters, and A_F is a set of *final addresses* that satisfy the following: $A_F \subseteq \{a \mid a \in A^L \wedge P(a) = H\}$, where $A^L = \{0, \ldots, L-1\}$.

Next, an instantaneous description (ID) of a CM in the program form and a transition relation among IDs are defined.

Definition 7. Let $M = (P, k, A_F)$ be a CM in the program form. Let L be the length of P. Thus, the set of addresses of P is $A^L = \{0, \ldots, L-1\}$. An *instantaneous description* (ID) of M is an expression $(a, (n_0, n_1, \ldots, n_{k-1})) \in A^L \times \mathbb{N}^k$, where $\mathbb{N} = \{0, 1, \ldots\}$. It represents that the i-th counter keeps n_i ($i \in \{0, \ldots, k-1\}$), and the instruction $P(a)$ is going to be executed.

Definition 8. Let $M = (P, k, A_F)$ be a CM in the program form, and L be the length of P. The *transition relation* $\vdash_{\overline{M}}$ over IDs of M is defined as follows. For every $i \in \{0, \ldots, k-1\}$, $a, a' \in A^L$ and $n_0, \ldots, n_{k-1}, n_i' \in \mathbb{N}$,

$$(a, (n_0, \ldots, n_{i-1}, n_i, n_{i+1}, \ldots, n_{k-1})) \\ \vdash_{\overline{M}} (a', (n_0, \ldots, n_{i-1}, n_i', n_{i+1}, \ldots, n_{k-1}))$$

holds if and only if one of the following conditions (1) – (5) is satisfied.

1. $P(a) = I_i \ \wedge \ n_i' = n_i + 1 \ \wedge \ a' = a + 1$
2. $P(a) = D_i \ \wedge \ n_i' = n_i - 1 \geq 0 \ \wedge \ a' = a + 1$
3. $P(a) = B_i(b_0, b_1) \ \wedge \ n_i' = n_i = 0 \ \wedge \ a' = a + b_0$
4. $P(a) = B_i(b_0, b_1) \ \wedge \ n_i' = n_i > 0 \ \wedge \ a' = a + b_1$
5. $P(a) = M_i(m_0, m_1) \ \wedge \ n_i' = n_i \ \wedge \ a' = a + 1$

Reflexive and transitive closure of $\vdash_{\overline{M}}$ is denoted by $\vdash_{\overline{M}}^*$, and n-step transition by $\vdash_{\overline{M}}^n$ $(n = 0, 1, \ldots)$.

Let $M = (P, k, A_F)$ be a CM. An ID $(a, (n_1, \ldots, n_k))$ is called an *initial ID* of M, if $a = 0$. An ID C is a *halting ID*, if there is no ID C' such that $C \vdash_{\overline{M}} C'$. An ID $(a, (n_1, \ldots, n_k))$ of M is called a *final ID*, if $a \in A_F$. Every final ID $(a, (n_1, \ldots, n_k))$ is a halting ID, since $P(a) = H$. Let C_i $(i \in \{0, 1, \ldots, n\})$ be IDs. We say that $C_0 \vdash_{\overline{M}} C_1 \vdash_{\overline{M}} \cdots \vdash_{\overline{M}} C_n$ (or $C_0 \vdash_{\overline{M}}^* C_n$) is a *complete computing process* of M, if C_0 is an initial ID, and C_n is a final ID.

In [16], it is proved that any CM(k) in the program form can be expressed by an equivalent RCM(k) in the quadruple form. By this, we can see that CMs in the program form are actually *reversible* CMs. In fact, from Definitions 5 and 8, it is easy to see that any ID of a CM(k) in the program form has at most one previous ID.

Proposition 1. *Any CM(k) in the program form is reversible.*

Example 2. Consider an RCM $M_{\text{twice}} = (P_{\text{twice}}, 2, \{9\})$, where P_{twice} is the WFP in Example 1. If we start from the initial ID $(0, (2, 0))$, we have the following complete computing process of M_{twice}.

$$(0, (2, 0)) \vdash_{\overline{M}_{\text{twice}}} (1, (2, 0)) \vdash_{\overline{M}_{\text{twice}}} (2, (2, 0)) \vdash_{\overline{M}_{\text{twice}}}^* (9, (0, 4))$$

Generally, $(0,(x,0)) \mid\!\!\!\!\overset{*}{}_{M_{\text{twice}}} (9,(0,2x))$ holds for all $x \geq 0$, *i.e.*, M_{twice} computes the function $f(x) = 2x$ and stores it in the counter 1.

Minsky [5] proved the following result that any (irreversible) TM can be simulated by an irreversible CM having only two counters. There, to reduce the number of counters, a technique of using a Gödel number, which is for encoding several counters into one, is employed.

Proposition 2. *For any TM, there is a CM(2) that simulates the TM.*

In the case of RCMs, Morita [19] showed the following result.

Proposition 3. *For any (irreversible) TM, there is an RCM(2) that simulates the TM.*

Note that the RCM(2) that simulates the TM leaves no garbage information when it halts except the input information initially given to the TM. In this sense, the class of RCM(2)'s is Turing universal.

In the following, we consider RCMs having any number of counters. By this, we can design algorithms for RCMs more flexibly.

3 URTM(98,10) T_{U} that Simulates RCMs

We give a URTM T_{U} that simulates any RCM(k) in the program form:

$$T_{\text{U}} = (Q, \{0, 1, *, -, @, \text{I}, \text{D}, \text{B}, \text{M}, \text{H}\}, \mathit{start}, \{\mathit{halt}(a), \mathit{halt}(r)\}, 0, \delta)$$

It has 98 states and 10 symbols, and the move table of δ is given in Figs. 3 and 4. A file that contains δ is found in [22]. Reversibility of T_{U} was verified by a computer program. Although the set of final states $\{\mathit{halt}(a), \mathit{halt}(b)\}$ is specified as above, these states are not included in the 98 states, since T_{U} works correctly even if they are removed.

Also note that there are several pairs of states each of which can be merged into one state. By this, we can reduce the number of states of T_{U}. For example, the states $i(2)$ and $b(1)$ can be merged without violating the reversibility condition. However, here, we do not do so, since $i(2)$ and $b(1)$ belong to different routines, and hence merging them spoils the readability of the algorithm of T_{U}.

	0	1	∗	−	@	I	D	B	M	H
start	R	@,R, *ca*(1)	∗,L, *ca*(4)	R		R	R	R		H,R, *h*(1)
h(1)		1,N, *halt*(*r*)	∗,N, *halt*(*a*)							
ca(1)		R	R	R	∗,R, *ca*(2)	R	R	R	R	R
ca(2)		R	@,L, *ca*(3)							
ca(3)		L	L	L	1,R, *start*	L	L	L	L	L
ca(4)		L				0,R, *i*(1)	0,R, *d*(1)	B,R, *b*(1)		
cb(1)		@,R, *cb*(2)	∗,L, *cb*(5)							
cb(2)		R	R	R	∗,L, *cb*(3)	R	R	R	R	R
cb(3)		L	@,L, *cb*(4)							
cb(4)		L	L	L	1,R, *cb*(1)	L	L	L	L	L
cb(5)		L	∗,R, *start*	L		I,R, *cb*(6)	D,R, *cb*(6)		M,R, *cb*(6)	
cb(6)		R	R	R		I,L, *cb*(5)	D,L, *cb*(5)	B,L, *cb*(5)	M,L, *cb*(5)	H,L, *cb*(5)
i(1)		R	R	R	@,R, *i*(*s*1)	R	R	R	R	R
i(2)	0,L, *i*(3)									
i(3)	I,R, *cb*(1)	L	L	L	L	L	L	L	L	L
i(*s*1)		R	1,R, *i*(*ss*)							
i(*ss*)	∗,R, *i*(2)	∗,R, *i*(*s*1)	R							
d(1)	0,L, *d*(*s*0)	R	R	R	R	R	R	R	R	R
d(2)	D,R, *cb*(1)	L	L	L		L	L	L	L	L
d(*s*0)			0,L, *d*(*ss*)							
d(*s*1)		L	1,L, *d*(*ss*)		@,L, *d*(2)					
d(*ss*)		∗,L, *d*(*s*1)	L							
b(1)		R	@,R, *b*(2)							
b(2)		R	R	R	0,R, *b*(3)	R	R	R	R	R
b(3)		1,L, *b*(*p*1)	∗,L, *b*(4)							
b(4)	0,L, *b*(5)									
b(5)		L	L	L	∗,R, *b*(6)	L	L	L	L	L
b(6)		0,R, *b*(*r*3)		−,L, *b*(*l*1)						
b(7)	0,R, *b*(8)	1,L, *b*(*p*6)	∗,L, *b*(9)							
b(8)	0,R, *b*(7)	R	R	R	R	R	R	R	R	R
b(9)	0,L, *b*(10)									
b(10)	∗,L, *b*(11)	L	L	L	L	L	L	L	L	L
b(11)		L	∗,L, *b*(12)	L						
b(12)		L		@,R, *b*(13)				L		
b(13)	0,L, *b*(14)	R	R	R	−,R, *m*(1)	R	R	R	R	R
b(14)	0,R, *b*(13)	L	L	L	L	L	L	L	L	L
b(*p*1)	0,L, *b*(*p*2)									
b(*p*2)		L	L	L	∗,R, *b*(*p*3)	L	L	L	L	L
b(*p*3)		R	@,R, *b*(*p*4)	R						
b(*p*4)	0,R, *b*(*p*5)	R	R	R		R	R	R	R	R
b(*p*5)		1,L, *b*(4)								
b(*p*6)	0,L, *b*(*p*7)									
b(*p*7)	∗,L, *b*(*p*8)	L	L	L	L	L	L	L	L	L
b(*p*8)		L	0,R, *b*(*p*9)	L						
b(*p*9)	0,R, *b*(*p*10)	R	R	R	R	R	R	R	R	R
b(*p*10)		1,L, *b*(9)								
b(*r*1)		R	R	R	−,R, *b*(*r*2)	R	R	R	R	R
b(*r*2)		0,R, *b*(*r*1)	0,L, *b*(*r*4)			I,R, *b*(*r*3)	D,R, *b*(*r*3)	B,R, *b*(*r*3)	M,R, *b*(*r*3)	H,R, *b*(*r*3)
b(*r*3)		R	R	R		I,L, *b*(*r*4)	D,L, *b*(*r*4)	B,L, *b*(*r*4)	M,L, *b*(*r*4)	H,L, *b*(*r*4)
b(*r*4)		1,R, *b*(7)		@,L, *b*(*r*5)						
b(*r*5)	1,R, *b*(*r*2)	L	L	L		L	L	L	L	L
b(*l*1)			@,L, *b*(*l*2)	@,R, *b*(*l*3)						
b(*l*2)		L	L	L				B,L, *b*(*l*1)		
b(*l*3)		R	R	R	∗,R, *b*(*l*4)	R	R	R	R	R
b(*l*4)			0,L, *b*(*l*10)	0,L, *b*(*l*5)						
b(*l*5)		L	L	L	−,R, *b*(*l*6)	L	L	L	L	L
b(*l*6)						I,L, *b*(*l*7)	D,L, *b*(*l*7)	B,L, *b*(*l*7)	M,L, *b*(*l*7)	H,L, *b*(*l*7)
b(*l*7)		L	L	L		I,L, *b*(*l*8)	D,L, *b*(*l*8)	B,L, *b*(*l*8)	M,L, *b*(*l*8)	H,L, *b*(*l*8)
b(*l*8)				@,R, *b*(*l*9)						
b(*l*9)	−,R, *b*(*l*4)	R	R	R		R	R	R	R	R
b(*l*10)				−,R, *b*(7)						

Figure 3. Move table of URTM(98,10) T_{U} (Part 1). For a quintuple $[p, s, t, d, q] \in \delta$ such that $(p, s) = (q, t)$, only $d \in \{\mathrm{L},\mathrm{R}\}$ is indicated

	0	1	$*$	$-$	@	I	D	B	M	H
$m(1)$		R	@,R,$m(2)$						R	
$m(2)$	0,R,$m(3)$	R	R	R	0,R,$m(2)$	R	R	R	R	R
$m(3)$		1,L,$m(p1)$	$*$,L,$m(4)$							
$m(4)$	0,L,$m(5)$									
$m(5)$	@,L,$m(5)$	L	L	L	$*$,R,$m(6)$	L	L	L	L	L
$m(6)$		0,R,$m(r1)$	$*$,L,$m(7)$	0,L,$m(l1)$						
$m(7)$				$-$,R,$m(8)$						
$m(8)$			$*$,L,$m(9)$							
$m(9)$		L	@,R,$m(10)$	L						
$m(10)$	@,R,$m(11)$	R	R	R		R	R	R	R	R
$m(11)$		1,L,$m(p6)$	$*$,L,$m(12)$						M,R,$cb(1)$	
$m(12)$					@,L,$m(13)$					
$m(13)$		L	L	L	$*$,L,$m(14)$	L	L	L	L	L
$m(14)$		L			$-$,R,$m(11)$				L	
$m(p1)$	0,L,$m(p2)$									
$m(p2)$	@,L,$m(p2)$	L	L	L	$*$,R,$m(p3)$	L	L	L	L	L
$m(p3)$		R	@,R,$m(p4)$	R						
$m(p4)$	0,R,$m(p5)$	R	R	R	0,R,$m(p4)$	R	R	R	R	R
$m(p5)$		1,L,$m(4)$								
$m(p6)$					@,L,$m(p7)$					
$m(p7)$		L	L	L	$*$,L,$m(p8)$	L	L	L	L	L
$m(p8)$		L	@,R,$m(p9)$	L						
$m(p9)$		R	R	R	@,R,$m(p10)$	R	R	R	R	R
$m(p10)$		1,L,$m(12)$								
$m(r1)$		R	R	R	$-$,R,$m(r2)$	R	R	R	R	R
$m(r2)$						I,L,$m(r3)$	D,L,$m(r3)$	B,L,$m(r3)$	M,L,$m(r3)$	H,L,$m(r3)$
$m(r3)$	0,L,$m(r6)$	L	L	L		I,L,$m(r4)$	D,L,$m(r4)$	B,L,$m(r4)$	M,L,$m(r4)$	H,L,$m(r4)$
$m(r4)$				@,L,$m(r5)$						
$m(r5)$	1,R,$m(6)$	L	L	L		L	L	L	L	L
$m(r6)$		L	L	L					M,L,$m(r7)$	
$m(r7)$				@,R,$m(r8)$						
$m(r8)$	1,R,$m(8)$	R	R	R					R	
$m(l1)$		L	L	L	$-$,R,$m(l2)$	L	L	L	L	L
$m(l2)$						I,R,$m(l3)$	D,R,$m(l3)$	B,R,$m(l3)$	M,R,$m(l3)$	H,R,$m(l3)$
$m(l3)$		R	R	R		I,L,$m(l4)$	D,L,$m(l4)$	B,L,$m(l4)$	M,L,$m(l4)$	H,L,$m(l4)$
$m(l4)$				@,R,$m(l5)$						
$m(l5)$	$-$,R,$m(6)$	R	R	R		R	R	R	R	R

Figure 4. Move table of URTM(98,10) T_{U} (Part 2)

3.1 Representing a Program of an RCM

Instructions of an RCM are encoded by symbol sequences of T_{U}.

First, instructions I_i and D_i are encoded as follows.

$$\texttt{-I1}^i\texttt{*} \qquad \text{and} \qquad \texttt{-D1}^i\texttt{*}$$

To encode B_i and M_i instructions, we define $\varphi(x)$ for $x \in A_{\mathrm{R}}^L \cup \{\#\}$.

$$\varphi(x) = \begin{cases} \texttt{1}^x & \text{if } x \in \{1, \ldots, L-1\} \\ \texttt{-}^{-x} & \text{if } x \in \{-1, \ldots, -L+1\} \\ \lambda & \text{if } x = \# \end{cases}$$

Here λ is the empty string. Then $B_i(b_0, b_1)$ and $M_i(m_0, m_1)$ are encoded as follows.

$$\texttt{-B1}^i\texttt{*}\,\varphi(b_0)\,\texttt{*}\,\varphi(b_1)\,\texttt{*} \quad \text{and} \quad \texttt{-M1}^i\texttt{*}\,\varphi(m_0)\,\texttt{*}\,\varphi(m_1)\,\texttt{*}$$

Finally, H instruction is encoded as follows.

$$\texttt{-H*} \qquad \text{or} \qquad \texttt{-H1*}$$

The URTM T_{U} halts in the state *halt*(a) (*halt*(r), respectively) if the encoding of H instruction is `-H*` (`-H1*`). This feature is convenient when T_{U} simulates an RCM acceptor.

The *code* (*i.e.*, description) of a WFP P of an RCM is obtained by concatenating the codes of instructions contained in P.

A k-tuple of numbers $(n_0, n_1, \ldots, n_{k-1})$ stored in the k counters are encoded as follows. It is attached at the right end of the program code.

$$\texttt{@1}^{n_0}\texttt{*1}^{n_1}\texttt{*}\cdots\texttt{*1}^{n_{k-1}}\texttt{*}$$

Here the symbol `@` will be used as a *counter marker* that indicates the counter to be accessed. Initially, it is at the position immediately left of the string $\texttt{1}^{n_0}$. When the i-th counter is to be accessed, it is shifted to the corresponding position, *i.e.*, $\texttt{*1}^{n_0}\texttt{*1}^{n_1}\texttt{*}\cdots\texttt{@1}^{n_i}\texttt{*}\cdots\texttt{*1}^{n_{k-1}}\texttt{*}$.

Example 3. Consider the WFP P_{twice} in Example 1. Assume the numbers stored in the two counters are $(5, 0)$. Then the combined code of them is as follows.

```
-B1*1**-M1*-*111111*-B*111111*1*-M**-*-D*
-I1*-I1*-B1**------*-M*------**-H*@11111**
```

3.2 Simulating RCMs in the URTM T_{U}

There are eight kinds of states in T_{U} as it is seen in Figs. 3 and 4. They are *start*, *ca*($\cdot$), *cb*($\cdot$), *h*($\cdot$), *i*($\cdot$), *d*($\cdot$), *b*($\cdot$), and *m*($\cdot$), each of which is called a *routine*. The states of the forms *h*($\cdot$), *i*($\cdot$), *d*($\cdot$), *b*($\cdot$), and *m*($\cdot$) are routines for processing H, I_i, D_i, B_i, and M_i instructions, respectively. The routine consisting of a single state *start* is to start the processing of an instruction other than M_i.

If the next instruction read by the state *start* is `H`, then T_{U} goes to the state *h*(1) and halts in the state *halt*(a) or *halt*(r). Otherwise, T_{U} executes the routine *ca*($\cdot$). It is for accessing the i-th counter specified by I_i, D_i, or B_i. Namely, by the routine *ca*($\cdot$), the counter marker `@` is shifted to the position immediately left of the i-th counter.

If the next instruction is `I` (or `D`, respectively), then T_{U} goes to the routine $i(\cdot)$ ($d(\cdot)$). By this, the content of the i-th counter is incremented (decremented). This operation is easily performed reversibly. After that, T_{U} goes to the routine $cb(\cdot)$, which shifts the counter marker `@` back to the 0th counter.

Figure 5 shows an example of the above process. Here T_{U} executes a WFP $I_2\,H$. It starts from the state *start* at $t = 0$. At $t = 4$, T_{U} goes to the routine $ca(\cdot)$. By this, the counter marker `@` is shifted to the position of the 2nd counter ($t = 45$). In this process, to count the number i (in this case $i = 2$) for shifting the counter marker to the correct position, the symbol `@` is temporarily used in the string `I11`.

Then, by the routine $i(\cdot)$, the content of the 2nd counter is incremented (it starts at $t = 45$). In this process, the instruction symbol `I` is temporarily replaced by `0` for returning to this position later. Incrementation of the i-th counter is performed by the states $i(s0)$, $i(s1)$, and $i(ss)$, and then enters $i(2)$ for returning to the position of the `I` instruction ($t = 67$). The case of the `D` instruction is similar.

At $t = 91$, T_{U} executes the routine $cb(\cdot)$. By this, the counter marker is shifted back to the 0th counter ($t = 133$).

At $t = 140$, T_{U} starts to execute the next instruction. Since it is `H`, T_{U} halts in the state $halt(a)$ at $t = 143$.

t	IDs of URTM(98,10) T_{U}
0	*start*`0-I11*-H*@1*11*111*1111*00`
4	`0-I@` $ca(1)$`1*-H*@1*11*111*1111*00`
45	`0-0` $i(1)$`11*-H**1*11@111*1111*00`
67	`0-011*-H**1*11@1111*1111*` $i(2)$`0`
91	`0-I` $cb(1)$`11*-H**1*11@1111*1111*0`
133	`0-I` $cb(6)$`11*-H*@1*11*1111*1111*0`
140	`0-I11*` *start*`-H*@1*11*1111*1111*0`
143	`0-I11*-H` $halt(a)$`*@1*11*1111*1111*0`

Figure 5. Execution process of a WFP $I_2\,H$ by T_{U}. The initial values of four counters are $(1, 2, 3, 4)$, and their final values are $(1, 2, 4, 4)$

Execution of B_i and M_i instructions is more complex than the cases of I_i and D_i, since the *reversible jump problem* noted in Sect. 1 exists. Here we explain a method of solving it using a simple example.

t	IDs of URTM(98,10) T_{U}
0	*start*0-B1*111*111*-I*-H1*-M1*---*---*-H*@1*11*0
4	0-B@ *ca*(1)*111*111*-I*-H1*-M1*---*---*-H*@1*11*0
75	0-B *b*(1)1*111*111*-I*-H1*-M1*---*---*-H**1@11*0
111	0-B1@111*111*-I*-H1*-M1*---*---*-H**1 *b*(*p*1)011*0
149	0-B1*111@ *b*(*p*4)111*-I*-H1*-M1*---*---*-H**1011*0
216	0-B1*111*011 *b*(*r*5)*@I*-H1*-M1*---*---*-H**1011*0
229	0-B1*111*101*-I *b*(*r*5)*@H1*-M1*---*---*-H**1011*0
247	0-B1*111*110*-I*-H1 *b*(*r*5)*@M1*---*---*-H**1011*0
400	0@B1*111*111*-I*-H1* *b*(13)@M1*---*---*-H**1011*0
401	0@B1*111*111*-I*-H1*- *m*(1)M1*---*---*-H**1011*0
461	0@B1*111*111*-I*-H1*-M1*--- *m*(*l*1)*0--*-H**1011*0
502	0-B1*111*111*@ *m*(*l*5)I*-H1*-M1*---*0--*-H**1011*0
539	0-B1*111*111*-I*@ *m*(*l*5)H1*-M1*---*-0-*-H**1011*0
573	0-B1*111*111*-I*-H1*@ *m*(*l*5)M1*---*--0*-H**1011*0
650	0-B1*111*111*-I*-H1*-M *cb*(1)1*---*---*-H**1@11*0
698	0-B1*111*111*-I*-H1*-M1*---*---* *start*-H*@1*11*0
701	0-B1*111*111*-I*-H1*-M1*---*---*-H *halt*(*a*)*@1*11*0

Figure 6. Execution process of a WFP $B_1(3,3)\,I_0\,H\,M_1(-3,-3)\,H$ simulated by T_{U}. The initial values of two counters are $(1,2)$

Figure 6 shows how a WFP $B_1(3,3)\,I_0\,H\,M_1(-3,-3)\,H$ is executed by T_{U}. It starts from the state *start* at $t=0$. At $t=4$, T_{U} goes to the routine $ca(\cdot)$ as in the case of `I` or `D`. By this, the counter marker `@` is shifted to the position of the 1st counter ($t=75$).

From the state $b(1)$, T_{U} starts to test if the 1st counter is 0 or positive. In this case it is positive, and thus T_{U} goes to the state $b(p_1)$ at $t=111$, where the counter marker `@` is temporarily replaced by `0` (if it is 0, it goes to $b(4)$). Thus, T_{U} accesses the second argument of

$B_1(3,3)$, which has the branching address $b_1 = 3$ ($t = 149$). It means that $T_{\rm U}$ must jump 3 instructions to the right. To do this operation, @ is used as an *address marker*. It is performed by shifting the address marker @ to the right by three instructions as shown at $t = 216$, 229, and 247 by the states $b(r1)$–$b(r5)$. By this, finally, the next instruction $M(-3,-3)$ is marked by @ at $t = 247$. Note that, if $b_1 < 0$, the address marker is shifted left $|b_1|$ times by the states $b(l1)$–$b(l10)$.

At $t = 400$, $T_{\rm U}$ finishes the $B_1(3,3)$ operation, and at $t = 401$, it starts to simulate the $M_1(-3,-3)$ operation. Since the marker @ is remaining at the left-side of the $B_1(3,3)$ instruction as a garbage information ($t = 401$), the routine $m(\cdot)$ must reversibly erase it by referring the second argument of $M_1(-3,-3)$, which has the merge address $m_1 = -3$. Using the states $m(l1)$–$m(l5)$, $T_{\rm U}$ shifts the address marker @ to the right as seen at $t = 461$, 502, 539, and 573. The address marker finally reaches the position of the instruction $M_1(-3,-3)$ (at $t = 573$). Using the fact that $M_1(-3,-3)$ is now under execution, the address marker @ is easily erased reversibly. By this, the garbage information on the previous address is erased.

After that, $T_{\rm U}$ executes the routine $cb(\cdot)$ ($t = 650$), and the counter marker is shifted back to the 0th counter. Finally, it executes the H instruction and halts ($t = 701$).

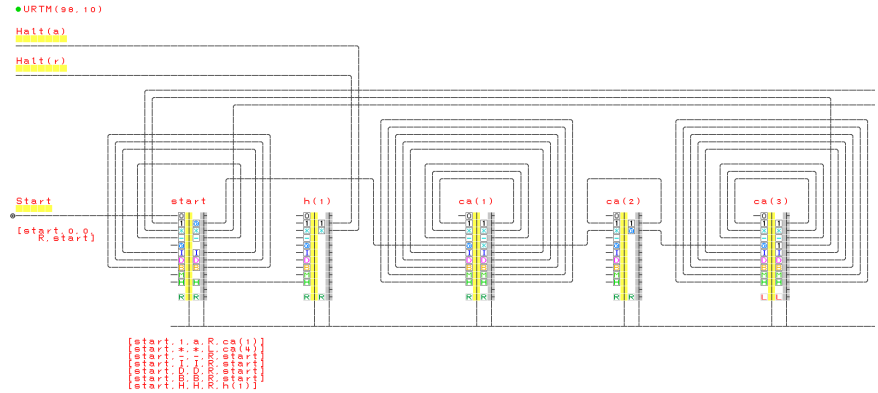
In this way, any WFP is simulated by $T_{\rm U}$ reversibly.

4 Visualizing URTM $T_{\rm U}$ Using Golly Simulator

Correctness of the algorithm (*i.e.,* δ) of $T_{\rm U}$ is roughly explained in Sect. 3.2. However, in this paper, we do not give its precise proof, since it will become very long and tedious to write and read. Instead, we created a simulator for the URTM(98,10) $T_{\rm U}$, which runs on Golly [21]. By this, readers can be convinced that $T_{\rm U}$ correctly simulates any RCM.

Golly is a general purpose simulator for cellular automata (CAs). It can deal with very large patterns of CAs, and its simulation speed is quite fast. By these features, it is also useful for simulating various machines other than CAs (see *e.g.* [17]). We created a system on Golly that can simulate any RTM having at most ten symbols and any RCM. We then constructed the URTM(98,10) $T_{\rm U}$ in it. Thus, by giving a code

of an RCM (Sect. 3.1) on its tape, any RCM can be simulated in it. Figure 7 shows a screenshot of Golly having the pattern of T_{U} with the code of P_{twice} in Example 3 on its tape.



(a)

(b)

Figure 7. Simulating URTM(98,10) T_{U} on Golly [22]. (a) A part of the finite control of T_{U}, and (b) its tape that contains P_{twice} in Example 3

It is easy to use this simulator that works on Golly. First, download the Golly system from [21], and install it. Second, put the zipped data file for T_{U}, which is available at [22], in the "Patterns" folder of Golly. Third, start the Golly simulator, and access any pattern file contained in the zip file. Then one can see full computing processes of the URTM(98,10) T_{U} that simulates various RCM examples, such as the ones that perform arithmetic operations, primality test, and others.

However, since the size of the pattern in Golly that realizes T_{U} is

very large, and millions (or even billions) of steps are required for T_{U} to simulate an RCM, the following method of viewing is recommended. First, select the simplest example of T_{U} that simulates M_{twice} in Example 2. Next, look at only the tape of T_{U}, which contains the code of M_{twice} and its counters. Then start the Golly simulator. Even in this case it takes more than 39 million steps to complete the simulation, and thus, simulation speed of Golly should be appropriately accelerated. By above, readers can see how the instructions of M_{twice} are processed on the tape of T_{U}. After that, readers may examine some details of the processing by slowing down the simulation speed. When viewing a computing process that simulates a larger RCM, it is appropriate to look at only the part of the tape that contains counters of the RCM.

5 Concluding Remarks

In this paper, we created a URTM(98,10) that directly simulates RCMs having arbitrary number of counters. Finding this kind of URTM that is much smaller in size is, of course, left for the future study. There remains a problem of finding a URTM that directly simulates RTMs. The method given in Sect. 3.2 for solving the reversible jump problem will be, in principle, applicable also to such a case. However, if we try to make a URTM that deals with RTMs having an arbitrary number of tape symbols, then its structure as well as codes of the RTMs will become complex. On the other hand, if we restrict the simulated RTMs to the ones having only two symbols, then the construction will become easier by using RTMs in the *program form* given in [23].

References

[1] A.-M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proc. London Math. Soc.,* Series 2, vol. 42, pp. 230–265, 1936.

[2] D. Woods and T. Neary, "The complexity of small universal Turing machines: a survey," *Theoret. Comput. Sci.*, vol. 410, pp. 443–450, 2009.

[3] S. Watanabe, "5-symbol 8-state and 5-symbol 6-state universal Turing machines," *J. ACM*, vol. 8, pp. 476–483, 1961.

[4] J. Cocke and M. Minsky, "Universality of tag systems with P = 2," *J. ACM*, vol. 11, pp. 15–20, 1964.

[5] M.-L. Minsky, *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ, 1967.

[6] Yu. Rogozhin, "Small universal Turing machines," *Theoret. Comput. Sci.*, vol. 168, pp. 215–240, 1996.

[7] M. Kudlek and Yu. Rogozhin, "A universal Turing machine with 3 states and 9 symbols," in *Proc. DLT 2001* (eds. W. Kuich, G. Rozenberg, A. Salomaa), LNCS 2295, 2002, pp. 311–318.

[8] T. Neary and D. Woods, "Four small universal Turing machines," *Fundam. Inform.*, vol. 91, pp. 123–144, 2009.

[9] K. Morita, *Theory of Reversible Computing.* Springer, Tokyo, 2017.

[10] C.-H. Bennett, "Logical reversibility of computation," *IBM J. Res. Dev.*, vol. 17, pp. 525–532, 1973.

[11] K. Morita and Y. Yamaguchi, "A universal reversible Turing machine," in *Proc. MCU 2007* (eds. J.O. Durand-Lose, M. Margenstern), LNCS 4664, 2007, pp. 90–98.

[12] M. Cook, "Universality in elementary cellular automata," *Complex Syst.*, vol. 15, pp. 1–40, 2004.

[13] K. Morita, "Reversible computing and cellular automata — A survey," *Theoret. Comput. Sci.*, vol. 395, pp. 101–131, 2008.

[14] K. Morita, "Universal reversible Turing machines with a small number of tape symbols," *Fundam. Inform.*, vol. 138, pp. 17–29, 2015.

[15] K. Morita, "Two small universal reversible Turing machines," in *Advances in Unconventional Computing* (ed. A. Adamatzky). Springer, 2017, pp. 221–237.

[16] K. Morita, *Reversible World of Cellular Automata – Fantastic Phenomena and Computing in Artificial Reversible Universe*. World Scientific Publishing, Singapore, 2024.

[17] K. Morita, "Reversible World of Cellular Automata – Data set for the Golly simulator, and solutions to selected exercises," Hiroshima University Institutional Repository, https://ir.lib.hiroshima-u.ac.jp/00055227, 2024.

[18] H.-B. Axelsen and R. Glück, "A simple and efficient universal reversible Turing machines," in *Proc. LATA 2011* (eds. A.H. Dediu, S. Inenaga, C. Martin-Vide), LNCS 6638, 2011, pp. 117–128.

[19] K. Morita, "Universality of a reversible two-counter machine," *Theoret. Comput. Sci.*, vol. 168, pp. 303–320, 1996.

[20] A. Alhazov, S. Verlan, and R. Freund, "Small universal reversible counter machines," in *Reversibility and Universality* (ed. A. Adamatzky). Springer, Cham, 2018, pp. 433–446.

[21] A. Trevorrow, T. Rokicki, and T. Hutton *et al.*, "Golly: an open source, cross-platform application for exploring Conway's Game of Life and other cellular automata," https://golly.sourceforge.io/, 2005.

[22] K. Morita, "A universal reversible Turing machine (URTM) that directly simulates reversible counter machines – Data set for the Golly simulator for visualizing the URTM," Hiroshima University Institutional Repository, https://ir.lib.hiroshima-u.ac.jp/00055598, 2024.

[23] K. Morita, "An instruction set for reversible Turing machines," *Acta Informatica*, vol. 58, pp. 377–396, 2021.

Kenichi Morita

Received August 22, 2024
Revised October 06, 2024
Accepted October 23, 2024

ORCID: https://orcid.org/0000-0002-9833-0539
Hiroshima University
Higashi-Hiroshima, Japan
E–mail: km@hiroshima-u.ac.jp