

# A universal reversible Turing machine (URTM) that directly simulates reversible counter machines

## — Data set for the Golly simulator for visualizing the URTM —

Kenichi Morita\* 

August 2024

### 1 Using “Universal\_reversible\_Turing\_machine\_for\_RCMs.zip”

The file “Universal\_reversible\_Turing\_machine\_for\_RCMs.zip” contains a rule file and pattern files for simulating a 98-state 10-symbol universal reversible Turing machine (URTM(98,10)), which are executable on the general purpose cellular automaton (CA) simulator *Golly* [7]. *Golly* is an excellent CA simulator developed by A. Trevorrow, T. Rokicki, T. Hutton *et al.* 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.* [5]). Here, we use it for simulating RTMs and reversible counter machines (RCMs). In particular, we visualize computing processes of the URTM(98,10) that simulates RCMs. Readers can easily see them by the following procedure.

1. Download the Golly system from <https://golly.sourceforge.io/>
2. Install the system on your computer.
3. Put the file “Universal\_reversible\_Turing\_machine\_for\_RCMs.zip” in the “Patterns” folder of Golly.
4. Start the Golly simulator.
5. Select the zip file in the “Patterns” folder from Golly, and access any pattern file (a file with .rle) in the zip file.

Note that readers can know the usage of Golly by accessing its help menu.

Besides the above files for Golly, a simulator file for Turing machines that works on SWI-Prolog (<https://www.swi-prolog.org/>) is included. Giving a description file of the URTM(98,10) to the simulator, its computing process is observed.

### 2 Files in “Universal\_reversible\_Turing\_machine\_for\_RCMs.zip”

Various files are contained in the zip file. Files having .rle are *pattern files* for simulating RTMs and RCMs. They describe initial configurations of them. The file “URTM.rule” is a *rule file*. It describes rules for simulating arbitrary RTMs and RCMs. The rule file is automatically installed by Golly when users have selected the zip file. The files “Prolog\_tm\_simulator.txt” and “Prolog\_urtm.txt” are the ones for SWI-Prolog.

---

\*Currently Professor Emeritus of Hiroshima University, [morita.rcomp@gmail.com](mailto:morita.rcomp@gmail.com)

## The rule file “URTM.rule”

Using this rule file, *any* RTM (having at most 10 symbols) and *any* RCM can be constructed in Golly. Therefore, users can also design new patterns for RTMs and RCMs.

## Pattern files

There are seven pattern files for Golly. Short explanations for them are given below. Note that some of these patterns require millions (or even billions) of steps to obtain results. Thus the simulation speed of Golly must be accelerated by pressing the “+” key (slow-down is by the “-” key) several times. When viewing full computing processes of the URTM(98,10), speed setting  $8^5 - 8^7$  will be appropriate.

- 1\_RCM\_examples.rie  
It shows nine examples of RCMs. Some of them are simulated by the URTM(98,10) (see below).
- 2\_RTM\_examples.rie  
It shows two simple examples of RTMs. Though they are not URTMs, by these examples users can know how to design RTMs using URTM.rule.
- 3\_URTM\_that\_simulates\_RCM\_twice.rie  
We consider the RCM(2)  $M_{\text{twice}}$  that computes the function  $f(x) = 2x$ . Giving the description of  $M_{\text{twice}}$  on the tape, the URTM(98,10) simulates it.
- 4\_URTM\_that\_simulates\_RCM\_exp.rie  
We consider the RCM(3)  $M_{\text{exp}}$  that computes the function  $f(x) = 2^x$ . Giving the description of  $M_{\text{exp}}$  on the tape, the URTM(98,10) simulates it.
- 5\_URTM\_that\_simulates\_RCM\_mult.rie  
We consider the RCM(4)  $M_{\text{mult}}$  that performs multiplication. Giving the description of  $M_{\text{mult}}$  on the tape, the URTM(98,10) simulates it.
- 6\_URTM\_that\_simulates\_RCM\_divide.rie  
We consider the RCM(4)  $M_{\text{divide}}$  that performs division. Giving the description of  $M_{\text{divide}}$  on the tape, the URTM(98,10) simulates it.
- 7\_URTM\_that\_simulates\_RCM\_prime.rie  
We consider the RCM(5)  $M_{\text{prime}}$  that performs the primality test. Giving the description of  $M_{\text{prime}}$  on the tape, the URTM(98,10) simulates it (but it takes a huge number of steps).

## Files for SWI-Prolog

There are two files that are executed on SWI-Prolog (<https://www.swi-prolog.org/>).

- Prolog\_tm\_simulator.txt  
It is a program of SWI-Prolog, which can simulate any one-tape TM by giving its description file. It also tests if a given TM is deterministic and reversible. A detailed explanation on its usage is found in this file as comments. Note that, to run this program on SWI-Prolog, the file name “Prolog\_tm\_simulator.txt” should be renamed to “Prolog\_tm\_simulator.pl”.
- Prolog\_urtm.txt  
The complete description of the URTM(98,10) and examples of initial tapes for it are given. It can be read from the above program.

### 3 Frameworks of RTMs and RCMs

In this section, we explain the frameworks of RTMs and RCMs. For more detailed explanations on these models, see [4, 6].

#### 3.1 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,  $\delta$  is a move relation, which is a subset of  $(Q \times S \times S \times \{L, N, R\} \times Q)$ . 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  $\delta$  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  $\delta$ .

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  $\delta$ .

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

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  $\delta$ .

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

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  $\lambda$  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  $\beta$  (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  $\delta$  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 [4] for the precise definition of the transition relation.

Bennett [1] first showed that any (irreversible) TM can be simulated by a garbage-less RTM. Hence, the class of RTMs is computationally universal. See [4] for computational universality of some restricted classes of RTMs.

### 3.2 Reversible counter machines (RCMs)

Here, we choose RCMs as the target machines that a URTM simulates, since the class of RCMs is known to be Turing universal, and their structures are very simple. Furthermore, using the framework of RCMs in the program form (rather than the quadruple form), construction of a URTM is simplified.

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.

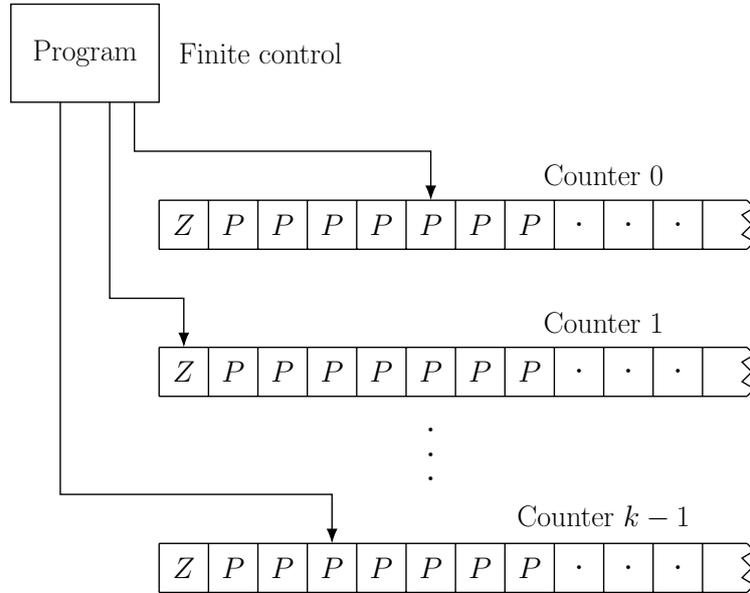


Figure 1:  $k$ -counter machine (CM( $k$ )).

Note that, in [3, 4], a CM is defined in the quadruple form, but here, we use the program form given in [6].

There are five kinds of *instructions* for a CM( $k$ ) shown below, where  $b_0, b_1, m_0$  and  $m_1$  are addresses of instructions, and  $i \in \{0, \dots, k-1\}$ . 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  $A^L, A_R^L, \mathbf{B}_k^L$  and  $\mathbf{M}_k^L$  as follows, where  $L (> 0)$  is the length of a program.

$$\begin{aligned} A^L &= \{0, 1, \dots, L-1\} \\ A_R^L &= \{1, \dots, L-1\} \cup \{-1, \dots, -L+1\} \\ \mathbf{B}_k^L &= \{B_i(b_0, b_1) \mid b_0, b_1 \in A_R^L \cup \{\#\}, i \in \{0, \dots, k-1\}\} \\ \mathbf{M}_k^L &= \{M_i(m_0, m_1) \mid m_0, m_1 \in A_R^L \cup \{\#\}, i \in \{0, \dots, k-1\}\} \end{aligned}$$

Here,  $A^L$  is the set of *addresses* of instructions, where the 0th instruction has the address 0, and the last has  $L-1$ .  $A_R^L$  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  $b_p \in A_R^L$  ( $m_p \in A_R^L$ , respectively) for  $p \in \{0, 1\}$ ,  $b_p$  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.

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

Note that, in [6] source and destination addresses are specified by *absolute addresses*, while they are specified by relative ones here.

**Definition 5** A *well-formed program* (WFP)  $P$  of length  $L$  for CM( $k$ ) is a mapping  $P : A^L \rightarrow \mathbf{S}_k^L$  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  $-b_p$ :

$$\begin{aligned} &\forall a \in A^L, \forall p \in \{0, 1\}, \forall i \in \{0, \dots, k-1\}, \\ &\forall b_0, b_1 \in A_R^L \cup \{\#\}, \exists m_0, m_1 \in A_R^L \cup \{\#\} \\ &((P(a) = B_i(b_0, b_1) \wedge b_p \neq \#)) \\ &\quad \Rightarrow (P(a + b_p) = M_i(m_0, m_1) \wedge m_p = -b_p) \end{aligned}$$

(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  $-m_p$ :

$$\begin{aligned} &\forall a \in A^L, \forall p \in \{0, 1\}, \forall i \in \{0, \dots, k-1\}, \\ &\forall m_0, m_1 \in A_R^L \cup \{\#\}, \exists b_0, b_1 \in A_R^L \cup \{\#\} \\ &((P(a) = M_i(m_0, m_1) \wedge m_p \neq \#)) \\ &\quad \Rightarrow (P(a + m_p) = B_i(b_0, b_1) \wedge b_p = -m_p) \end{aligned}$$

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  $P_{\text{twice}}$  be the following sequence of instructions.

$$\begin{array}{cccccccccc} B_1(1, \#) & M_1(-1, 6) & B_0(6, 1) & M_0(\#, -1) & D_0 & I_1 & I_1 & B_1(\#, -6) & M_0(-6, \#) & H \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array}$$

It is easy to see that  $P_{\text{twice}}$  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.

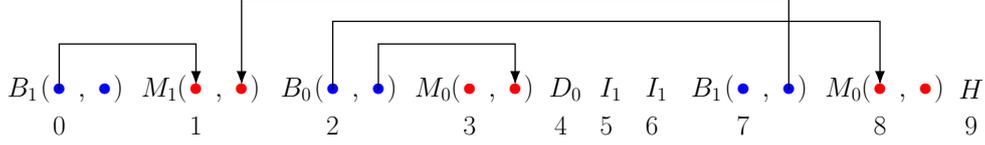


Figure 2: Graphical representation of the WFP  $P_{\text{twice}}$ .

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  $A^L = \{0, \dots, L-1\}$ .

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  $A^L = \{0, \dots, L-1\}$ . An *instantaneous description* (ID) of  $M$  is an expression  $(a, (n_0, n_1, \dots, n_{k-1})) \in A^L \times \mathbb{N}^k$ , where  $\mathbb{N} = \{0, 1, \dots\}$ . It represents that the  $i$ -th counter keeps  $n_i$  ( $i \in \{0, \dots, k-1\}$ ), 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*  $\xrightarrow{M}$  over IDs of  $M$  is defined as follows. For every  $i \in \{0, \dots, k-1\}$ ,  $a, a' \in A^L$  and  $n_0, \dots, n_{k-1}, n'_i \in \mathbb{N}$ ,

$$\begin{array}{c} (a, (n_0, \dots, n_{i-1}, n_i, n_{i+1}, \dots, n_{k-1})) \\ \xrightarrow{M} (a', (n_0, \dots, n_{i-1}, n'_i, n_{i+1}, \dots, n_{k-1})) \end{array}$$

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  $\xrightarrow{M}$  is denoted by  $\xrightarrow{M^*}$ , and  $n$ -step transition by  $\xrightarrow{M^n}$  ( $n = 0, 1, \dots$ ).

Let  $M = (P, k, A_F)$  be a CM. An ID  $(a, (n_1, \dots, n_k))$  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  $C \xrightarrow{M} C'$ . An ID  $(a, (n_1, \dots, n_k))$  of  $M$  is called a *final ID*, if  $a \in A_F$ . Every final ID  $(a, (n_1, \dots, n_k))$  is a halting ID, since  $P(a) = H$ . Let  $C_i$  ( $i \in \{0, 1, \dots, n\}$ ) be IDs. We say that  $C_0 \xrightarrow{M} C_1 \xrightarrow{M} \dots \xrightarrow{M} C_n$  (or  $C_0 \xrightarrow{M^*} C_n$ ) is a *complete computing process* of  $M$ , if  $C_0$  is an initial ID, and  $C_n$  is a final ID.

In [6], 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  $P_{\text{twice}}$  is the WFP in Example 1. If we start from the initial ID  $(0, (2, 0))$ , we have the following complete computing process of  $M_{\text{twice}}$ .

$$(0, (2, 0)) \xrightarrow{M_{\text{twice}}} (1, (2, 0)) \xrightarrow{M_{\text{twice}}} (2, (2, 0)) \xrightarrow{M_{\text{twice}}^*} (9, (0, 4))$$

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

Minsky [2] 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 [3] 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.

## 4 URTM(98,10) $T_U$ that simulates RCMs

We give a URTM  $T_U$  that simulates any  $RCM(k)$  in the program form. Note that the objective of this construction is not to minimize the numbers of states and tape symbols, but to give a URTM of a reasonable size whose simulating processes of RCMs are easily understood.  $T_U$  is defined as below.

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

It has 98 states and 10 symbols, and the move relation  $\delta$  is given in Figs. 3 and 4. A complete description of  $T_U$  is given in the file “Prolog\_urtm.txt”. Reversibility of  $T_U$  is verified by the TM simulator “Prolog\_tm\_simulator.txt” that runs on SWI-Prolog (see Sect 2). Although the set of final states  $\{halt(a), halt(b)\}$  is specified as above, these states are not included in the 98 states, since  $T_U$  works as a URTM correctly even if these states 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 different routines, and hence merging them spoils readability of the algorithm of  $T_U$ .

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

Figure 3: Move table of URTM(98,10)  $T_U$  (part 1).

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

Figure 4: Move table of URTM(98,10)  $T_U$  (part 2).

## 4.1 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.

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

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

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

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

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

Finally,  $H$  instruction is encoded as follows.

$$-H* \quad \text{or} \quad -H1*$$

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  $P$ .

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

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

**Example 3** Consider the WFP  $P_{\text{twice}}$  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**
```

## 4.2 Simulating RCMs in the URTM(98,10) $T_U$

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  $m(\cdot)$ . The states of the forms  $h(\cdot)$ ,  $i(\cdot)$ ,  $d(\cdot)$ ,  $b(\cdot)$  and  $m(\cdot)$  are for processing  $H$ ,  $I_i$ ,  $D_i$ ,  $B_i$  and  $M_i$  instructions, respectively. The 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,  $T_U$  goes to 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 marker  $@$  for the counters is shifted to the position immediately left of the  $i$ -th counter.

If the next instruction symbol 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 position.

Figure 5 shows an example of the above process. Here,  $T_U$  executes a WFP  $I_2H$ . 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 ( $t = 45$ ). Then, by the routine  $i(\cdot)$ , the content of the 2nd counter is incremented ( $t = 67$ ). At  $t = 91$  it 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  $I_2H$  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 URTM have to “jump” from  $B_i$  and  $M_i$  reversibly. We explain how it is done using a simple example. Figure 6 shows an example.  $T_U$  executes a WFP  $B_1(3,3)I_0HM_1(-3,-3)H$ . 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  $b(1)$ ,  $T_U$  tests 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  $t = 111$ . Therefore,  $T_U$  accesses the second argument of  $B_1(3,3)$ , which has the branching address  $b_1 = 3$ , at  $t = 149$ . It means that  $T_U$  must jump 3 instructions to the right. It is performed by shifting the position marker @ to the right as shown at  $t = 216, 229$  and  $247$  by the states  $b(r_1) - b(r_5)$ . By this, finally, the next instruction  $M(-3,-3)$  is marked by @ at  $t = 247$ . Note that when shifting the position marker @ to the left, the states  $b(l_1) - b(l_10)$  are used. At  $t = 400$   $T_U$  finishes the  $B_1(3,3)$  operation, and at  $t = 401$  it starts to simulate the  $M_1(-3,-3)$  operation. Since the position marker @ is left at the left-side of the B instruction ( $t = 401$ ) as a garbage information, the routine  $m(\cdot)$  reversibly erases it by referring the second argument of  $M_1(-3,-3)$ , which has the merge address  $m_1 = -3$ . Using the states  $m(l_1) - m(l_5)$ ,  $T_U$  shifts the position marker @ to the right as seen at  $t = 461, 502, 539$  and  $573$ . By this, the garbage information on the previous address is reversibly erased. After that,  $T_U$  executes the routine  $cb(\cdot)$  ( $t = 650$ ), and the counter marker is shifted back to the 0th counter ( $t = 698$ ). Finally it executes the  $H$  instruction, and halts ( $t = 701$ ).

In this way, any WFP is simulated by  $T_U$ .

$t$	IDs of URTM(98,10) $T_U$
0	<i>start</i> 0-B1*111*111*-I*-H1*-M1*----*----*-H*@1*11*0
4	0-B@ <u>ca(1)</u> *111*111*-I*-H1*-M1*----*----*-H*@1*11*0
75	0-B <u>b(1)</u> 1*111*111*-I*-H1*-M1*----*----*-H**1@11*0
111	0-B1@111*111*-I*-H1*-M1*----*----*-H**1 <u>b(p1)</u> 011*0
149	0-B1*111@ <u>b(p4)</u> 111*-I*-H1*-M1*----*----*-H**1011*0
216	0-B1*111*011 <u>b(r5)</u> *@I*-H1*-M1*----*----*-H**1011*0
229	0-B1*111*101*-I <u>b(r5)</u> *@H1*-M1*----*----*-H**1011*0
247	0-B1*111*110*-I*-H1 <u>b(r5)</u> *@M1*----*----*-H**1011*0
400	0@B1*111*111*-I*-H1* <u>b(l3)</u> @M1*----*----*-H**1011*0
401	0@B1*111*111*-I*-H1*- <u>m(1)</u> M1*----*----*-H**1011*0
461	0@B1*111*111*-I*-H1*-M1*---- <u>m(l1)</u> *0---*---H**1011*0
502	0-B1*111*111*@ <u>m(l5)</u> I*-H1*-M1*----*0---*---H**1011*0
539	0-B1*111*111*-I*@ <u>m(l5)</u> H1*-M1*----*0-*---H**1011*0
573	0-B1*111*111*-I*-H1*@ <u>m(l5)</u> M1*----*0-*---H**1011*0
650	0-B1*111*111*-I*-H1*-M <u>cb(1)</u> 1*----*----*-H**1@11*0
698	0-B1*111*111*-I*-H1*-M1*----*----* <i>start</i> -H*@1*11*0
701	0-B1*111*111*-I*-H1*-M1*----*----*-H <u>halt(a)</u> *@1*11*0

Figure 6: Execution process of a WFP  $B_1(3,3)I_0HM_1(-3,-3)H$  simulated by  $T_U$ . The initial values of two counters are (1,2).

**Acknowledgements:** I express my gratitude to the developing and support teams of *Golly* [7].

## References

- [1] Bennett, C.H.: Logical reversibility of computation. *IBM J. Res. Dev.* **17**, 525–532 (1973). doi:[10.1147/rd.176.0525](https://doi.org/10.1147/rd.176.0525)
- [2] Minsky, M.L.: *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ (1967)
- [3] Morita, K.: Universality of a reversible two-counter machine. *Theoret. Comput. Sci.* **168**, 303–320 (1996). doi:[10.1016/S0304-3975\(96\)00081-3](https://doi.org/10.1016/S0304-3975(96)00081-3)
- [4] Morita, K.: *Theory of Reversible Computing*. Springer, Tokyo (2017). doi:[10.1007/978-4-431-56606-9](https://doi.org/10.1007/978-4-431-56606-9)
- [5] Morita, K.: 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)
- [6] Morita, K.: *Reversible World of Cellular Automata – Fantastic Phenomena and Computing in Artificial Reversible Universe*. World Scientific Publishing, Singapore (2024). doi:[10.1142/13516](https://doi.org/10.1142/13516)
- [7] Trevorrow, A., Rokicki, T., Hutton *et al.*, T.: Golly: an open source, cross-platform application for exploring Conway’s Game of Life and other cellular automata. <https://golly.sourceforge.io/> (2005)