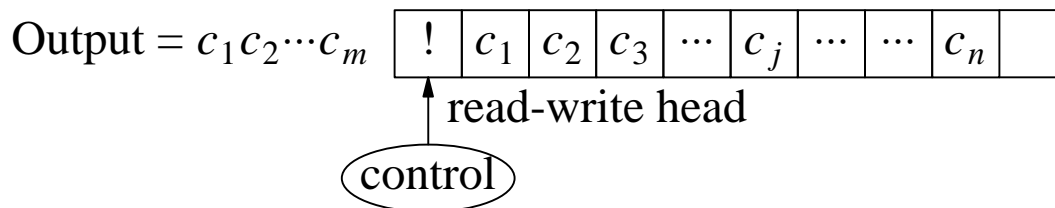
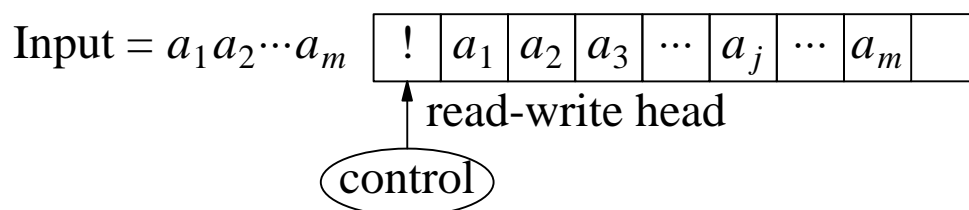


## BASIC STRUCTURE OF TURING MACHINE

### Turing machine:

- The input tape is also used as the output tape.
- The read/write head can move left and right, and thus a tape symbol may be read/written more than once.
- The output string is left justified, and the read/write head is placed on first square.



- it is possible that the tape was used beyond the the last ouput symbol  $c_n$ .
- $n$  may be  $< m$ ,  $= m$ , or  $> m$ .

## A TM FOR ERASING THE INPUT

- Input alphabet =  $\{a, b\}$
- The tape symbol  $\beta$  is just like 'B' = blank except that it is written by the TM.

Input =  $aaba$ : 

!	a	a	b	a	B	B	B	...
---	---	---	---	---	---	---	---	-----

  
↑ start

Output =  $\lambda$ : 

!	$\beta$	$\beta$	$\beta$	$\beta$	$\beta$	B	B	...
---	---------	---------	---------	---------	---------	---	---	-----

  
↑ halt

!	a	a	b	a	B	B	...
---	---	---	---	---	---	---	-----

  
↑ start

!	a	a	b	a	B	B	...
---	---	---	---	---	---	---	-----

  
↑ erase

after many erasing steps

!	$\beta$	$\beta$	$\beta$	$\beta$	B	B	...
---	---------	---------	---------	---------	---	---	-----

  
↑ erase

!	$\beta$	$\beta$	$\beta$	$\beta$	$\beta$	B	...
---	---------	---------	---------	---------	---------	---	-----

  
↑ go-left

!	$\beta$	$\beta$	$\beta$	$\beta$	$\beta$	B	...
---	---------	---------	---------	---------	---------	---	-----

  
↑ go-left

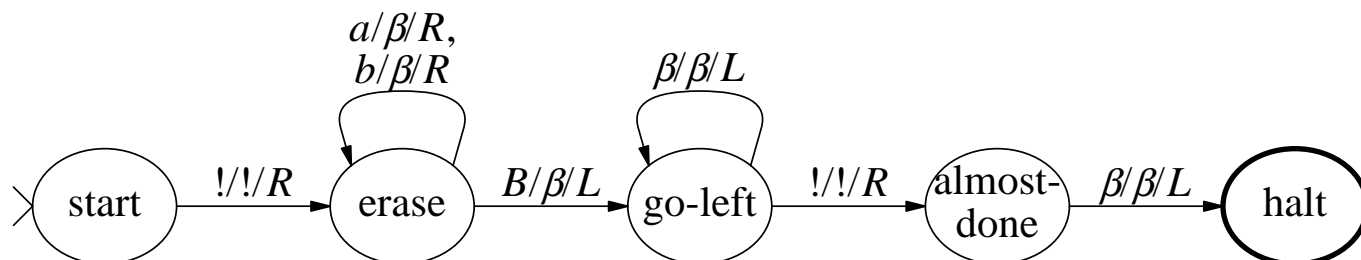
!	$\beta$	$\beta$	$\beta$	$\beta$	$\beta$	B	...
---	---------	---------	---------	---------	---------	---	-----

  
↑ almost-done

!	$\beta$	$\beta$	$\beta$	$\beta$	$\beta$	B	...
---	---------	---------	---------	---------	---------	---	-----

  
↑ halt

## A FINITE-STATE DIAGRAM FOR THE ERASING-TM



### Transition-label:

Tape-symbol-read/Tape-symbol-written/Head-move-direction

## A TM FOR RIGHT-SHIFTING A NON-EMPTY BINARY STRING

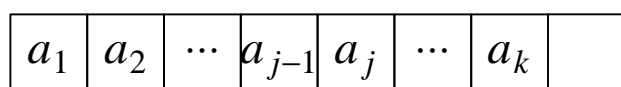
Input:    1011    (no starting '!')  
Output:   !1011   ('!' added to the left)

### Key ideas:

- (1) Read an input symbol  $a_j$ , move to the right, write that  $a_j$  (in position  $j + 1$ ), and at the same time remember the original symbol  $a_{j+1}$  in that position. Repeat this for all symbols in the input. (Write '!' at the start-position in the start-state, when there is no previous input symbol.)
- (2) When 'blank' is read, drop the last symbol remembered ( $a_n$ ), and then keep moving to the left to the first position.

### Configuration: $(x. y : state)$

- $x$  = the tape content to the left of the read/write head
- $y$  = the remaining non-blank part of the tape
- The state  $m0$  ( $m1$ ) means the most recent bit remembered from the position on the left is 0 (resp., 1).
- The first symbol of  $y$  is the current symbol being read.

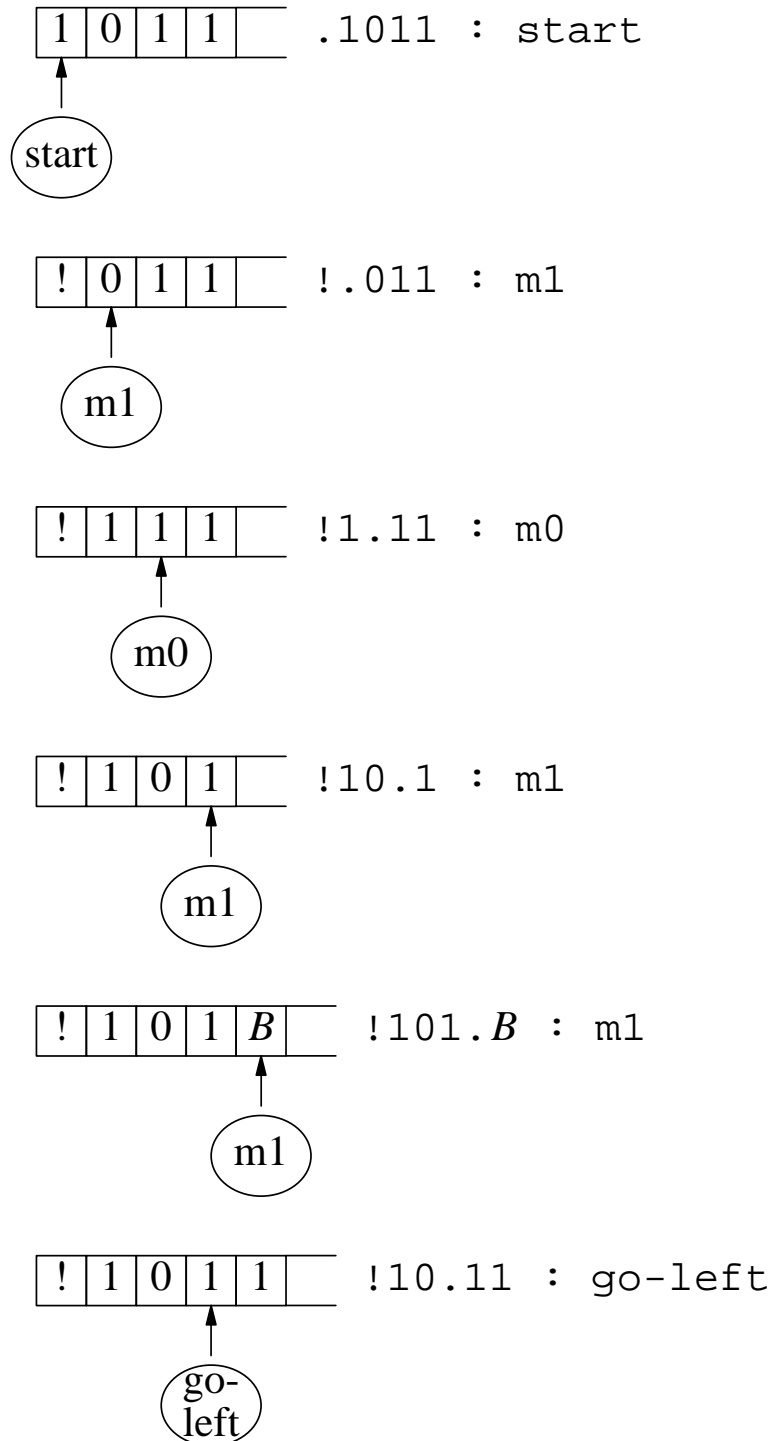


↑  
read/write head

$$x = a_1 a_2 \dots a_{j-1}$$

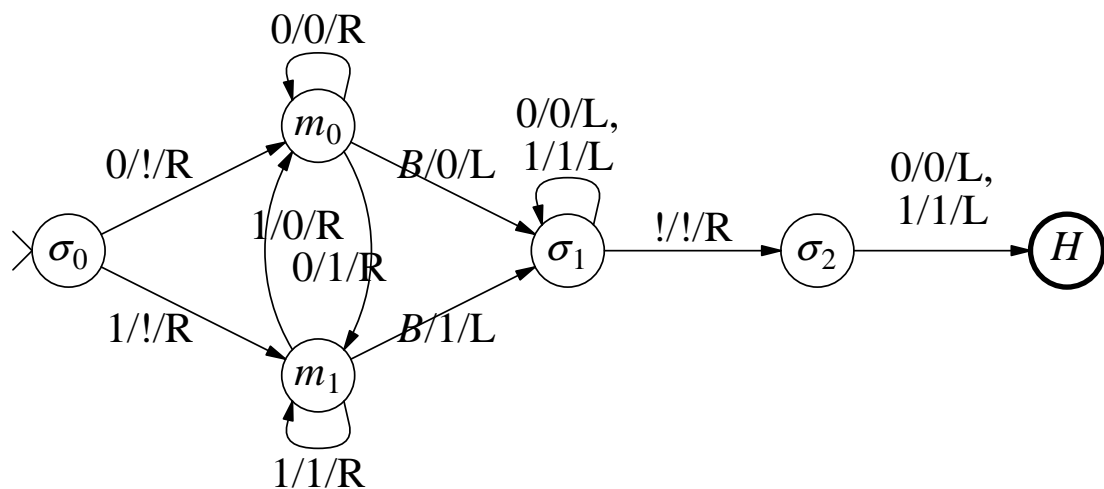
$$y = a_j a_{j+1} \dots a_k \text{ or } B$$

## PART OF THE MOVE SEQUENCE FOR INPUT 1011



"halt" = The final-state of a TM.

## THE STATE-DIAGRAM FOR $T_{right-shift}$



$\sigma_0$  = start

$\sigma_1$  = go-left

$\sigma_1$  = almost-done

$H$  = halt

### Question:

- ? Show the state diagram for a Turing Machine which makes a copy of an input binary string. The input !100 produces the output !100=100, where the copy is separated from the original by '='. (The next page shows a move sequence of such a TM.)

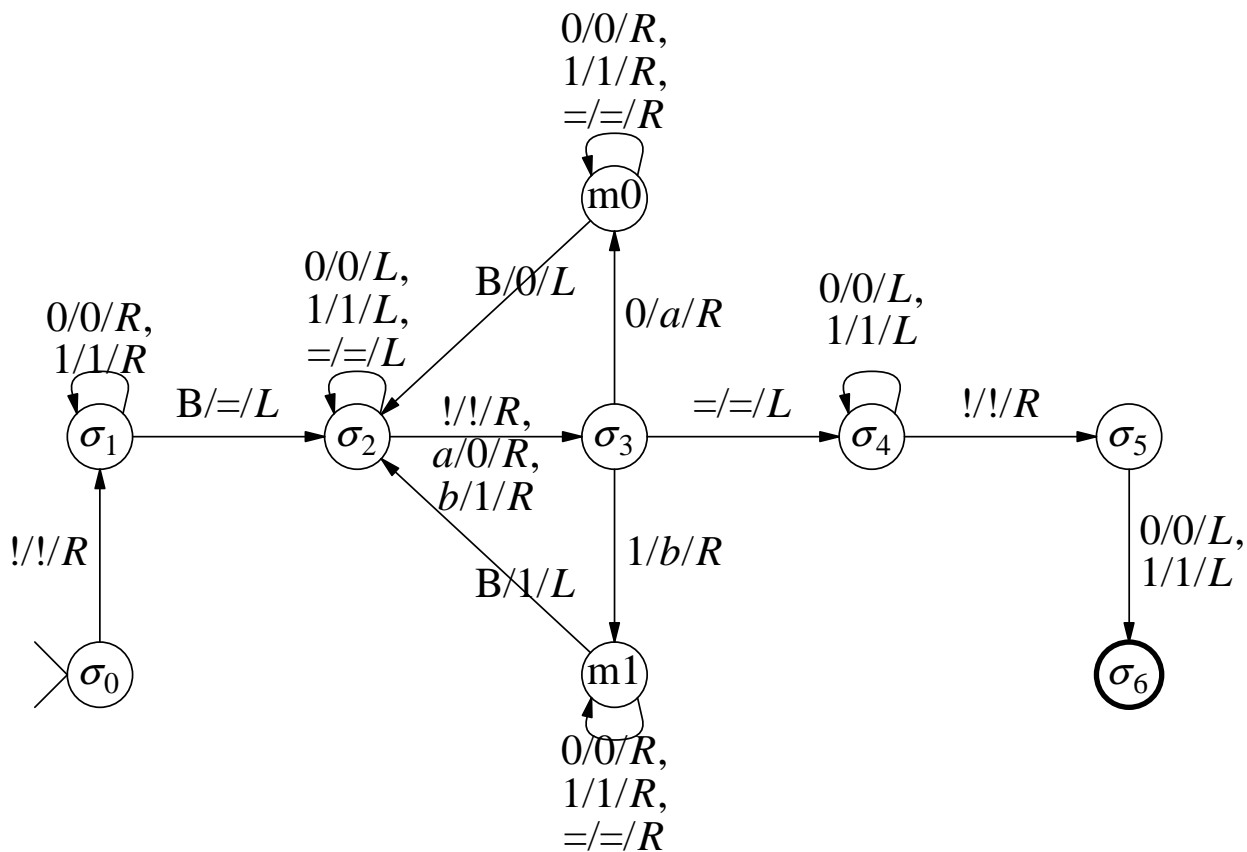
## A MOVE SEQUENCE FOR $T_{copy}$

```

.!100           : start
!.100           : findFirstB
!1.00          : findFirstB
!10.0          : findFirstB
!100.B         : findFirstB
!10.0=         : findLastSymbolCopiedOr!
!1.00=        : findLastSymbolCopiedOr!
!.100=        : findLastSymbolCopiedOr!
.!100=        : findLastSymbolCopiedOr!
!.100=        : foundSymbolToCopy
!b.00=        : m1 (b = marked '1')
!b0.0=        : m1
!b00.=        : m1
!b00=.B       : m1
!b00.=1       : findLastSymbolCopiedOr!
!b0.0=1       : findLastSymbolCopiedOr!
!b.00=1       : findLastSymbolCopiedOr!
!.b00=1       : findLastSymbolCopiedOr!
!1.00=1       : foundSymbolToCopy
!1a.0=1       : m0 (a = marked '0')
!1a0.=1       : m0
!1a0=.1       : m0
!1a0=1.B      : m0
!1a0=.10      : findLastSymbolCopiedOr!
!1a0.=10      : findLastSymbolCopiedOr!
!1a.0=10      : findLastSymbolCopiedOr!
!1.a0=10      : findLastSymbolCopiedOr!
!10.0=10      : foundSymbolToCopy
!10a.=10      : m0
!10a=.10      : m0
!10a=1.0      : m0
!10a=10.B     : m0
!10a=1.00     : findLastSymbolCopiedOr!
!10a=.100     : findLastSymbolCopiedOr!
!10a.=100     : findLastSymbolCopiedOr!
!10.a=100     : findLastSymbolCopiedOr!
!100.=100     : foundSymbolToCopy
!10.0=100     : find!AllCopyingDone
!1.00=100     : find!AllCopyingDone
!.100=100     : find!AllCopyingDone
.!100=100     : find!AllCopyingDone
!.100=100     : almostDone
.!100=100     : halt

```

## STATE DIAGRAM FOR $T_{copy}$



- $\sigma_0$  = start
- $\sigma_1$  = fi ndFirstB
- $\sigma_2$  = fi ndLastSymbolCopiedOr!
- $\sigma_3$  = FoundSymbolToCopy
- $m0$  = symbolToCopyIs0
- $m1$  = symbolToCopyIs1
- $\sigma_4$  = fi nd!AllCopyingDone
- $\sigma_5$  = almostDone
- $\sigma_6$  = halt

## FIND LEFTMOST OCCURRENCE OF A MATCH-STRING $x$ IN A MASTER-STRING $y$

**Input to  $TM_{find-substring}$ :**

- A match-string  $x$  and a master-string  $y$  (both over some alphabet  $\Sigma$ ), separated by a special symbol '?'. Input alphabet =  $\Sigma \cup \{?\}$ .

**Example Input Tape ( $\Sigma = \{c, d\}$ ):**

- If  $x = cdcd$  and  $y = ddcdddcdcdcd$ ,  
then input tape =  $!y?x = !ddcdddcdcdcd?cdcd$ .

**Output of  $TM_{find-substring}$ :**

- If  $x$  occurs in  $y$ , then mark the leftmost occurrence of  $x$ .
- Otherwise, it is same as the input string.

**Example Output Tape:**

- $!ddcdddCDCDcdcd?cdcd$ , with  $C = \text{marked-}c$  and  $D = \text{marked-}d$ .

**An Intermediate Tape Configuration:  $!ddCD.dcdcdcdcd?cdCd$**

- The initial part  $cd$  of  $x$  is already matched with the part  $CD$  in  $y$ .
- The next symbol  $c$  (shown as  $C$ ) in  $x$  has failed to match the next symbol  $d$  immediately to the right of the marked part  $CD$  in  $y$ .
- After abandoning the current matching process, cleaning  $y$  and choosing a new start position in  $y$  (shown as  $\clubsuit$  = the position of the first symbol  $C$  of the matched part  $CD$  in  $y$ ), and finally cleaning  $x$ , the tape configuration becomes:  $!dd\clubsuit d d d c d c d c d d . ? c d c d$ .
- The symbol  $\clubsuit$  allows the TM to regard it both: (1) as '?' and start a new matching process of  $x$  into the tail part of  $y$  after  $\clubsuit$ , and (2) as 'c' and restore it to 'c' before halting.

## RECOVERING FROM A FAILED-MATCH AFTER PARTIAL MATCHING

- Shown below is an example of tape-configurations and states in the recovery process after matching the initial part  $cd$  of  $x$  and then failing to match the next symbol  $c$  (shown as  $C$ ) in  $x$  with the next symbol  $d$  (after the ".") in  $y$ . See the first configuration below.

```

!ddCD.ddcdcdcd?cdCd      : mc (memory of current match-symbol c in x)
!ddC.Dddcdcdcd?cdCd      : clean-y
!dd.Cddcdcdcd?cdCd       : clean-y (cleaning and moving left)
!d.dcdcdcdcd?cdCd        : clean-y (found first unmarked symbol or '!')
!dd.cddcdcdcd?cdCd       : mark-next-match-start
!dd¢.ddcdcdcd?cdCd       : clean-x
!dd¢d.dcdcdcd?cdCd       : clean-x (moving right)
!dd¢dd.ccdcdcd?cdCd      : clean-x
...
!dd¢ddcdcdcd?cd.Cd       : clean-x (found marked symbol in x)
!dd¢ddcdcdcd?c.dcd       : find-first-symbol-in-x
!dd¢ddcdcdcd?.cdcd       : find-first-symbol-in-x
!dd¢ddcdcdcd.?cdcd       : find-first-symbol-in-x
!dd¢ddcdcdcd?.cdcd       : find-first-symbol-in-x

```

### Notes:

- State  $mc$  at the time of failing shows the TM was looking for  $c$  in  $y$ .
- The TM first moves left and changes the state to  $clean-y$ . In state  $clean-y$ , the TM cleans  $y$  (replacing each  $C/D$  by  $c/d$  as it moves left), and then marks the position after which the next attempt to match  $x$  in  $y$  can start and changes state to  $clean-x$ .  
If there is  $c$  ( $d$ ) in that marked position, it writes  $¢$  (resp.,  $ḍ$ ).
- In state  $clean-x$ , the TM finds in  $x$  the only  $C$  (or  $D$ ), replaces it by  $c$  (resp.  $d$ ), and moves left with the new state  $find-first-symbol-in-x$ .
- In state  $find-first-symbol-in-x$ , the TM moves left zero or more positions to left till it finds '?' and then finally moves right to positions itself to read the first symbol in  $x$ .
- Since  $x$  begins with  $c$  here, there is at most one  $¢$  (no  $ḍ$ ) at any time in the part  $y$  and that too only if there are no  $C$  and  $D$  in  $y$ .

## SIMULATION OF AN ARBITRARY TM BY THE UNIVERSAL TURING MACHINE UT

### Input to UT:

- Encoding of start-state of TM, encoding of transitions of TM, and encoding of an input to TM preceded by a special symbol, say, '·'.

!	encoding of start-state of TM	encoding of TM-transitions	:	encoding of input to TM	<i>B</i>	<i>B</i>	...
---	----------------------------------	-------------------------------	---	----------------------------	----------	----------	-----

### Encoding of TM for UT:

- For states of TM, use  $q$  and a fixed number of binary bits; for tape symbols of TM use  $a$  and a fixed number of binary bits (use  $B$  in place of binary bits; this string will be considered by UT as a match for its own blank= $B$  tape symbol).

States of $T_{copy}$			Tape symbols of $T_{copy}$	
$\sigma_0$ $q0000$	$\sigma_3$ $q0011$	$\sigma_6$ $q0110$	! $a000$	$a$ $a011$
$\sigma_1$ $q0001$	$\sigma_4$ $q0100$	$m0$ $q0111$	0 $a001$	$b$ $a100$
$\sigma_2$ $q0010$	$\sigma_5$ $q0101$	$m1$ $q1000$	1 $a010$	= $a101$

- Encode each transition as a 5-tuple:  
(current-state, symb-read, symb-written, dir-of-move, next-state).

Thus, the transition  $(\sigma_3, 1/b/R, m1)$  in  $T_{copy}$  is encoded as

$$(q0011, a010, a100, R, q1000).$$

Note that we actually do not need any separating '·'.

**Input symbols of UT:**  $\{q, a, 0, 1, (, ), :\}$ .

- UT uses other tape symbols such as  $Q$  and  $A$  for marking.

## KEY STEPS IN SIMULATION OF AN ARBITRARY TM BY UT

- (1) Track current state of TM. Initially, it is given right after '!'. Subsequently, UT copies the 5th item of the TM-transition that UT simulated most recently right after '!', replacing old TM-state.
- (2) Track current input symbol of TM. Initially, this is next to the special symbol ':'; subsequently it may be any place after ':'.  
UT marks the appropriate 'a' after ':' as 'A' for this purpose.
- (3) Identify the applicable transition of TM. UT uses  $T_{find-substring}$  to match current state of TM and current input symbol of TM with the first two components of the encoded transitions of TM.  
The applicable TM-transition is indicated by its first 'q' and 'a' marked as 'Q' and 'A'.
- (4) Make changes to the part after ':' based on the symbol-written by TM. UT does this by a copy-replace operation based on the 3rd part of the applied transition of TM (which also changes the current 'A' to 'a').  
UT then simulates L/R move of TM by marking as 'A' the preceding 'a' or the following 'a' of the position where UT carried out the copy-replace operation.
- (5) Copy-replace the last part of the applied TM-transition after '!' to reflect the current-state of TM.
- (6) Unmark the applied TM-transition and move left until ready to read '!'.

The steps (3)-(6) are applied repeatedly and UT halts if and only if TM halts.

## EXAMPLE OF SIMULATION of $T_{copy}$ BY UT

- The first few steps of  $T_{copy}$  for input 01 that we will simulate below in UT:

.!01:  $\sigma_0 = \text{start}$   
 !.01:  $\sigma_1 = \text{findFirstB}$   
 !0.1:  $\sigma_1$   
 !01.B:  $\sigma_1$   
 !0.1=:  $\sigma_2 = \text{find lastSymbolCopiedOr!}$   
 !.01=:  $\sigma_2$   
 .!01=:  $\sigma_2$

- We show below only just two encoded transitions of  $T_{copy}$  for want of space, including the one that UT is about to simulate. We throw away the least recently used  $T_{copy}$ -transition when we need to make room for a new one.
- We indicate below the UT's tape configuration after completion of each step (2)-(6) and also indicate the particular step used to get the current configuration from the previous one. (Details of substring-matching, copy-replace operations, etc are not shown.)

•!q0000(q0000a000a000Rq0001)(q0001a001a001Rq0001)⋯:a000a001a010  
 (2) !q0000(q0000a000a000Rq0001)(q0001a001a001Rq0001)⋯:•A000a001a010  
 (3) !q0000(Q0000A000•a000Rq0001)(q0001a001a001Rq0001)⋯:A000a001a010  
 (4) !q0000(Q0000A000a000Rq0001)(q0001a001a001Rq0001)⋯:a000•A001a010  
 (5) !q0001•(Q0000A000a000Rq0001)(q0001a001a001Rq0001)⋯:a000A001a010  
 (6) •!q0001(q0000a000a000Rq0001)(q0001a001a001Rq0001)⋯:a000A001a010  
 (3) !q0001(q0000a000a000Rq0001)(Q0001A001•a001Rq0001)⋯:a000A001a010  
 (4) !q0001(q0000a000a000Rq0001)(Q0001A001a001Rq0001)⋯:a000a001•A010  
 (5) !q0001•(q0000a000a000Rq0001)(Q0001A001a001Rq0001)⋯:a000a001A010  
 (6) •!q0001(q0000a000a000Rq0001)(q0001a001a001Rq0001)⋯:a000a001A010  
 we show below a diff. 1st trans. of  $T_{copy}$  in UT's tape to continue  
 (6) •!q0001(q0001a010a010Rq0001)(q0001a001a001Rq0001)⋯:a000a001A010  
 (3) !q0001(Q0001A010•a010Rq0001)(q0001a001a001Rq0001)⋯:a000a001A010  
 (4) !q0001(Q0001A010a010Rq0001)(q0001a001a001Rq0001)⋯:a000a001a010•B  
 (5) !q0001•(Q0001A010a010Rq0001)(q0001a001a001Rq0001)⋯:a000a001a010B

(6) •!q0001(q0001a010a010Rq0001)(q0001a001a001Rq0001)…:a000a001a010B  
we show below a diff. 2nd trans. of  $T_{copy}$  in UT's tape to continue

(6) •!q0001(q0001a010a010Rq0001)(q0001aBBBa101Lq0010)…:a000a001a010B  
(3) !q0001(q0001a010a010Rq0001)(Q0001ABBB•a101Lq0010)…:a000a001a010B  
(4) !q0001(q0001a010a010Rq0001)(Q0001ABBBa101Lq0010)…:a000a001•A010a101  
(5) !q0010•(q0001a010a010Rq0001)(Q0001ABBBa101Lq0010)…:a000a001A010a101  
(6) •!q0010(q0001a010a010Rq0001)(q0001aBBBa101Lq0010)…:a000a001A010a101  
we show below a diff. 1st trans. of  $T_{copy}$  in UT's tape to continue

(6) •!q0010(q0010a010a010Lq0010)(q0001aBBBa101Lq0010)…:a000a001A010a101  
(3) !q0010(Q0010A010•a010Lq0010)(q0001aBBBa101Lq0010)…:a000a001A010a101  
(4) !q0010(Q0010A010a010Lq0010)(q0001aBBBa101Lq0010)…:a000•A001a010a101  
(5) !q0010•(Q0010A010a010Lq0010)(q0001aBBBa101Lq0010)…:a000A001a010a101  
(6) •!q0010(q0010a010a010Lq0010)(q0001aBBBa101Lq0010)…:a000A001a010a101  
we show below a diff. 2nd trans. of  $T_{copy}$  in UT's tape to continue

(6) •!q0010(q0010a010a010Lq0010)(q0010a001a001Lq0010)…:a000A001a010a101  
(3) !q0010(q0010a010a010Lq0010)(Q0010A001•a001Lq0010)…:a000A001a010a101  
(4) !q0010(q0010a010a010Lq0010)(Q0010A001a001Lq0010)…:•A000a001a010a101  
(5) !q0010•(q0010a010a010Lq0010)(Q0010A001a001Lq0010)…:A000a001a010a101  
(6) •!q0010(q0010a010a010Lq0010)(q0010a001a001Lq0010)…:A000a001a010a101

## EXERCISE

1. Show the moves of UT as it simulates the next two moves of  $T_{copy}$ . (You need to show two cycles of steps (3)-(6) following where we stopped above; you will also need to show some different encoded transitions of  $T_{copy}$  replacing the least recently used one as we had done above. Write legibly.)