Monash University
Faculty of Information Technology
2nd Semester 2022

FIT2014 Theory of Computation
Tutorial 6
Turing Machines

SOLUTIONS

Although you may not have time to do all the exercises in this Tutorial Sheet, it is still important that you attempt all the main questions and a selection of the Supplementary Exercises.

Even for those Supplementary Exercises that you do not attempt seriously, you should still give some thought to how to do them before reading the solutions.

1.

(i) $\underline{a}aa \rightarrow \#\underline{a}a \rightarrow \#a\underline{a} \rightarrow \#aa\underline{\Delta} \rightarrow \#a\underline{a} \rightarrow \#\underline{a}\# \rightarrow \underline{\#}a\# \rightarrow \#\underline{a}\# \rightarrow \#\#\underline{\#} \rightarrow \#\#\#\underline{\Delta}$ **Accept**

(ii) $\underline{a}ba \rightarrow \#\underline{b}a \rightarrow \#b\underline{a} \rightarrow \#ba\underline{\Delta} \rightarrow \#b\underline{a} \rightarrow \#\underline{b}\# \rightarrow \underline{\#}b\# \rightarrow \#\underline{b}\# \rightarrow \#\#\underline{\#}$ **CRASH**

(iii) $\underline{b}aaba \rightarrow \#\underline{a}aba \rightarrow \#a\underline{a}ba \rightarrow \#aa\underline{b}a \rightarrow \#aab\underline{a} \rightarrow \#aaba\underline{\Delta} \rightarrow \#aab\underline{a} \rightarrow \#aa\underline{b}\# \rightarrow \#a\underline{a}b\#$
$\rightarrow \#\underline{a}ab\# \rightarrow \underline{\#}aab\# \rightarrow \#\underline{a}aab\# \rightarrow \#\#\underline{a}b\# \rightarrow \#\#a\underline{b}\# \rightarrow \#\#ab\underline{\#} \rightarrow \#\#a\underline{b}\# \rightarrow \#\#\underline{a}\#\#$
$\rightarrow \#\underline{\#}a\#\# \rightarrow \#\#\underline{a}\#\# \rightarrow \#\#\#\underline{\#}\# \rightarrow \#\#\#\#\underline{\#}$ **Accept**

(iv) $\underline{a}babb \rightarrow \#\underline{b}abb \rightarrow \#b\underline{a}bb \rightarrow \#ba\underline{b}b \rightarrow \#bab\underline{b} \rightarrow \#babb\underline{\Delta} \rightarrow \#bab\underline{b} \rightarrow \#ba\underline{b}\# \rightarrow \#b\underline{a}b\#$
$\rightarrow \#\underline{b}ab\# \rightarrow \underline{\#}bab\# \rightarrow \#\underline{b}ab\# \rightarrow \#\#\underline{a}b\# \rightarrow \#\#a\underline{b}\# \rightarrow \#\#ab\underline{\#} \rightarrow \#\#a\underline{b}\# \rightarrow \#\#\underline{a}\#\#$
$\rightarrow \#\underline{\#}a\#\# \rightarrow \#\#\underline{a}\#\# \rightarrow \#\#\#\underline{\#}\# \rightarrow \#\#\#\#\underline{\#}$ **Accept**

2.

From	To	Read	Write	Move
1	3	a	#	R
3	3	a	a	R
3	4	b	b	R
4	4	b	b	R
4	5	Δ	Δ	L
5	6	b	#	L
6	7	b	#	L
7	7	a	a	L
7	7	b	b	L
7	8	#	#	R
8	2	#	#	R
8	9	a	#	R
9	9	a	a	R
9	9	b	b	R
9	10	#	#	L
10	7	b	#	L
1	11	b	b	R
11	2	Δ	Δ	R

3.

From	To	Read	Write	Move
1	2	Δ	Δ	R
1	3	a	#	R
3	4	Δ	Δ	L
4	2	#	Δ	R
3	5	a	a	R
5	5	a	a	R
5	6	Δ	Δ	L
6	7	a	Δ	L
7	7	a	a	L
7	1	#	a	R

4.

(i). $\underline{\mathbf{\Delta}} \to \mathbf{\Delta}\underline{\mathbf{\Delta}}$ **HALT**

(ii). $\underline{\mathbf{a}} \to \#\underline{\mathbf{\Delta}} \to \underline{\#}\mathbf{A} \to \mathbf{a}\underline{\mathbf{A}} \to \mathbf{aa}\underline{\mathbf{\Delta}} \to \mathbf{a}\underline{\mathbf{a}}$ **HALT**

(iii). $\underline{\mathbf{a}}\mathbf{a} \to \#\underline{\mathbf{a}}$ – $\mathbf{A} \to \mathbf{a}\underline{\mathbf{a}}\mathbf{A} \to \mathbf{a}\#\underline{\mathbf{A}} \to \mathbf{a}\#\mathbf{A}\underline{\mathbf{\Delta}} \to \mathbf{a}\#\underline{\mathbf{A}}\mathbf{A} \to \mathbf{a}\underline{\#}\mathbf{AA}$
$\to \mathbf{aa}\underline{\mathbf{A}}\mathbf{A} \to \mathbf{aa}$ **HALT**

5.

(i)

Such a TM decide reject based only on what's in the first tape cell, which might be blank (if the ty string) or the first letter of the input string. So the languages that can be are (using regular expression notation, since they are all regular languages):

$$\begin{array}{cc}
\emptyset & \varepsilon \\
\mathbf{a}(\mathbf{a}\cup\mathbf{b})^* & \mathbf{a}(\mathbf{a}\cup\mathbf{b})^*\cup\varepsilon \\
\mathbf{b}(\mathbf{a}\cup\mathbf{b})^* & \mathbf{b}(\mathbf{a}\cup\mathbf{b})^*\cup\varepsilon \\
(\mathbf{a}\cup\mathbf{b})(\mathbf{a}\cup\mathbf{b})^* & (\mathbf{a}\cup\mathbf{b})(\mathbf{a}\cup\mathbf{b})^*\cup\varepsilon,
\end{array}$$

which is the same as $(\mathbf{a}\cup\mathbf{b})^*$.

(ii)

These TMs read one character at a time, never re-reading them. Although they may overwrite a character they have read, they always go to the right so the new character is never seen again.

This TM is really just a FA. The class of languages recognised by them is therefore the class of regular languages.

(iii) (This excercise is Sipser ex. 3.13.)

Consider a transition from state p to state q labelled $x \to y, S$, where S denotes Stay Still. If this transition is taken, then there may be subsequent stay-still transitions, taking execution through some sequence of states and changing the character at the current tape cell, possibly several times. All the while, no more input characters are read. Since the TM is deterministic, this sequence of transitions is completely determined. If it leads to acceptance, then we can replace the transition from p to q labelled $x \to y, S$ by one from p to the Accept state labelled x. Otherwise, the sequence ends with a Right step transition, say $x' \to y', R$, to some state q'. Then we can replace the transitionfrom p to q labelled $x \to y, S$ by one from p to q' labelled $x \to y', R$.

Doing this for each transition with direction S gives an equivalent TM in which all transitions have direction R. But this is equivalent to a FA, as we saw in (ii).

(iv)

This type of TM is equivalent to that of part (iii). If you have a TM that crashes when attempting to move off the left-hand end of the tape, you can use it to simulate a TM of the other type as follows.

Create new TM states and transitions that do some preprocessing as follows. Shift the input string one cell to the right, marking the first cell (now vacated by the input string) with a new special symbol, say \$. Then position the Tape Head on the cell to the right of \$, which is the first letter of the input string. Then enter the original TM at its start state, and process the input just as it does. (Everything is happening one cell to the right, but the TM doesn't know that.) This new TM also needs new loop transitions, one at each state of the old TM and each of them labelled by $\$ \to \$, R$. If at any stage the Tape Head reaches the first cell, which means it has just moved Left from what *was* the first cell of the tape of the *original* TM, then it moves Right without changing the \$ and so returns to the second cell of the tape (i.e., the first cell of the original tape) without changing state. So, we have simulated standing still when trying to move off the left end of the tape, using a TM that would crash if we attempted to do so.

Conversely, if we have a TM that stands still when attempting to move off the left end of the tape, then we can use it to simulate one that would crash if it attempted to move left from there. The argument is similar to the above, except that if we are reading the $ character, then we need to go immediately into a Reject state (rather than just looping at our current state and moving Right, as we did above). Alternatively, if we want to avoid Reject states, we could simply not provide any transitions for the cha[illegible] TM finds itself reading $ — which will only happen if it has moved to the le[illegible] of the original tape — then it crashes.

So the class of lan[illegible] same as that for normal TMs.

6.

(a)

Each of the pure q[illegible] of the form

$$\mathtt{Rotation}(\theta,\hat{q}) \;=\; \cos(\theta/2)+\sin(\theta/2)\cdot\hat{q}$$

for appropriate angles θ and axis vectors $\hat{q}$ (see Assignment 2, p10). In each case, since the real part is 0, the angle θ must be 180°, in order to give real part $\cos(180°/2) = \cos 90° = 0$. The axis, in each case, is just the quaternion unit itself. So we have:

quaternion	angle	axis	
i	180°	i	$(1,0,0)$
j	180°	j	$(0,1,0)$
k	180°	k	$(0,0,1)$

(b)

Consider iv/i. This represents rotation of the point represented by the pure quaternion v according to the rotation represented by i: see Assignment 2, p11. Now, as we have just seen, i represents rotation by 180° around axis i. Now, rotation by 180° around axis i just negates the j- and k-coordinates, while fixing everything along the i-axis. (This is like rotating the entire two-dimensional plane by 180° around the origin, which has the effect of just negating the coordinates.) So it maps $xi + yj + yk$ to $xi - yj - zk$.

The other cases behave similarly. In summary,

expression	evaluates to
iv/i	$xi - yj - zk$
jv/j	$-xi + yj - zk$
kv/k	$-xi - yj + zk$

(c)

We prove the following claim by induction on n:

> For all n, a product of n fundamental unit quaternions represents either a rotation by 0° or a rotation by 180° about an axis given by some fundamental unit quaternion.

Inductive basis:

When $n = 0$, a product of n fundamental unit quaternions is an empty product and has value 1, which represents a rotation of 0°. So the claim is established in this case.

Inductive step:

Now suppose $n \geq 0$, and assume that the claim is true for n. (This is the Inductive Hypothesis.) Let $q_{n+1}q_n \cdots q_2 q_1$ be a product of $n+1$ fundamental unit quaternions. Now, $q_n \cdots q_2 q_1$ is a product of n fundamental unit quaternions, so it must represent a rotation by either 0° or 180° around an axis given by one of i, j, k, by the Inductive Hypothesis. Let $v = xi + yj + zk$ be the vector we start with, to be rotated. The rotation is done using

$$v' \;=\; q_n \cdots q_2 q_1\, v/q_1/q_2 \cdots /q_n.$$

In the 0° case, the rotation does nothing to v. In the 180° case, we saw in part (b) that it negates exactly two of the three coordinates of v. So, comparing the vector *after* the rotation so far (call it v') with the one *before* (which we called v), either zero or two of its coordinates are negated. In detail, the new vector v' must be one of $xi + yj + zk$, $xi - yj - zk$, $-xi + yj - zk$, $-xi - yj + zk$.

Now consider application of the rotation represented by q_{n+1}. This is done using the expression

$$[illegible]_{+1}\, v'/q_{n+1}.$$

Since q_{n+1} is also a fu[illegible]nion, this rotation must negate two of the coordinates (again using part (b)[illegible]e vectors v' that could have been obtained so far — i.e., the four listed at [illegible] paragraph — and negate two of the coordinates, you will get a vector that [illegible] or two of its coordinates negated, compared with v. Therefore it is obtain[illegible]n of 180° around an axis given by a fundamental unit quaternion. So the cl[illegible]

Conclusion:
Therefore, by Mathematical Induction, the claim is true for all $n \geq 1$.

Another approach to proving this claim is to prove that any product of fundamental unit quaternions is one of $\pm 1, \pm i, \pm j, \pm k$, so that the rotation represented by such a product must be represented by one of these eight quantities. We have already seen that i, j, k represent rotations by 180°. The negations of these quantities also represent rotations by 180°, by virtually the same reasoning given in (a).[1] Lastly, both 1 and -1 represent rotations by angle 0°, since $\cos(\theta/2) = \pm 1$ if and only if θ is a multiple of 360°, which as a rotation angle is equivalent to 0° and changes nothing.

7.

Interpretations of tape alphabet characters:

character	interpretation
`i`	fundamental unit quaternion i
`j`	fundamental unit quaternion j
`k`	fundamental unit quaternion k
`v`	a quaternion, to be rotated
`d`	division, /
`s`	marks first tape cell
`t`	marks the factors before `v` that have been examined, and have been (or are being) checked off against corresponding divisions after `v`.

[1] The axis *lines* are the same too. The axis *direction* is reversed, which has the effect of interchanging clockwise and anticlockwise senses of rotation around the axis line . . . but that actually doesn't make any difference when the angle is 180°.

	current state	current symbol	new state	new symbol	direction
/* Mark first cell (using next state to remember its contents), start moving rightwards. */					
Start	1	i	3	s	R
	1	j	4	s	R
	1	[illegible]	[illegible]	[illegible]	R
Accept	2				
/* Character to [illegible]. Move to the cell before the first blank. */					
	3	[illegible]	[illegible]	[illegible]	R
	3	[illegible]	[illegible]	[illegible]	R
	3	[illegible]	[illegible]	[illegible]	R
	3	[illegible]	[illegible]	[illegible]	R
	3	[illegible]	[illegible]	[illegible]	R
	3	Δ	6	Δ	R
/* Character to match, at other end, is j. Move to the cell before the first blank. */					
	4	i	4	i	R
	4	j	4	j	R
	4	k	4	k	R
	4	v	4	v	R
	4	d	4	d	R
	4	Δ	7	Δ	R
/* Character to match, at other end, is k. Move to the cell before the first blank. */					
	5	i	5	i	R
	5	j	5	j	R
	5	k	5	k	R
	5	v	5	v	R
	5	d	5	d	R
	5	Δ	8	Δ	R
/* If it's i, then it matches! Step left, to check for division symbol. */					
	6	i	9	Δ	L
/* If it's j, then it matches! Step left, to check for division symbol. */					
	7	j	9	Δ	L
/* If it's k, then it matches! Step left, to check for division symbol. */					
	8	k	9	Δ	L
/* If it's d, then we have division by i, matching earlier i. Step left. */					
	9	d	10	Δ	L
/* Move back to the left, to the rightmost s or t. */					
	10	i	10	i	L
	10	j	10	j	L
	10	k	10	k	L
	10	v	10	v	L
	10	d	10	d	L
	10	s	10	s	R
	10	t	11	t	R
/* Now at the leftmost unchecked letter. Mark cell (using next state to remember its contents), start moving rightwards. */					
	11	i	3	t	R
	11	j	4	t	R
	11	k	5	t	R
	11	v	12	Δ	L
/* Move leftwards to start of tape, overwriting each cell with blank. */					
	12	t	12	Δ	L
	12	s	14	Δ	R
/* Step leftwards to start of tape, overwriting cell with blank. */					
	14	Δ	2	Δ	L
/* String started with v, so there should be nothing after it. */					
	15	Δ	2	Δ	L

8.

The main idea is to use the two stacks to represent the Turing machine tape, as follows. The portion of the tape from the start up to, but not including, the position of the Tape Head is in the first stack. The portion of the tape from the Tape Head up to, and including, the last non-blank letter is in the second stack, with the last non-blank letter at the bottom of that stack and the Tape Head position corresp[illegible]e stack. So the letter at the Tape Head position is on the top of the second [illegible]

To simulate the T[illegible]A has a set-up phase where it moves the input string into the second stack, [illegible]nput at the bottom and the first letter of input at the top. The first stack is [illegible]ut tape of the 2PDA is now empty too.

Then, for each Tu[illegible]on, we alter the stacks in the 2PDA in a way that simulates the alteratio[illegible] Tape Head position in the TM.

Consider a genera[illegible]ansition from state p to state q, which has the form $x \to y, d$, where x and y are letters and $d \in \{\text{Left}, \text{Right}\}$ is the direction in which the Tape Head moves after reading x and writing y.

We will assume that a 2PDA transition $t, u, v \to w, z$ means: if the input letter is t, the top of the first stack is u, and the top of the second stack is v, then we pop u and v from their respective stacks, and push w onto the first stack and z onto the second stack. Any or all of these can be ε, with the same interpretation as in PDAs.

In the 2PDA, the way we translate the TM transition $x \to y, d$ depends on d.

If $d = \text{Right}$, then we have a transition from p to q labelled by $\varepsilon, \varepsilon, x \to y, \varepsilon$. This has the effect of popping x from the top of the second stack and pushing y onto the top of the first, which simulates moving the Tape Head to the Right after overwriting x by y. It is only done if the letter on top of the second stack is x, which is as it should be.

If $d = \text{Left}$, then a little more fiddling is needed. All we want to do is to move the letter at the top of the first stack to the top of the second stack, but we must do so only when the letter at the top of the second stack is x, and x must be replaced by y. We can't just write $\varepsilon, z, x \to \varepsilon, z$, since x was popped before z was put on the second stack, so now has become lost from that stack, whereas we want to change it to y and have it sitting just underneath z. Nor can we just write $\varepsilon, z, \varepsilon \to \varepsilon, z$: although this does correctly simulate the movement of the tape head one cell to the Left, it will do this whether or not x was sitting atop the second stack, and it will not change x to y. So this is not an exact translation of the TM rules that we are trying to simulate.

We can achieve our desired objective by two transitions.

We create a new state, just for this TM transition, and distinct from all other states. Call it s_{pq}. We first create a transition from p to s_{pq} labelled by $\varepsilon, \varepsilon, x \to \varepsilon, y$, which changes x to y on top of the second stack, and has no other effect on the stacks. But it ensures that we only get to s_{pq} if we have x on the top of the second stack at the beginning of the transition. We then create transitions from s_{pq} to q labelled by $\varepsilon, z, \varepsilon \to \varepsilon, z$, which moves z onto the top of the second stack; there is one such transition from s_{pq} to q for each letter z of the tape alphabet.

This completes the description of how to simulate the TM by a 2PDA.

9.

(a)

$$A_{t,i,\mathbf{a}} \vee A_{t,i,\mathbf{b}} \vee A_{t,i,\Delta}$$

(b)

$$(\neg A_{t,i,\mathbf{a}} \vee \neg A_{t,i,\mathbf{b}}) \wedge (\neg A_{t,i,\mathbf{a}} \vee \neg A_{t,i,\Delta}) \wedge (\neg A_{t,i,\mathbf{b}} \vee \neg A_{t,i,\Delta})$$

(c) This is just the conjunction of the expressions for (a) and (b):

$$(A_{t,i,\mathbf{a}} \vee A_{t,i,\mathbf{b}} \vee A_{t,i,\Delta}) \wedge (\neg A_{t,i,\mathbf{a}} \vee \neg A_{t,i,\mathbf{b}}) \wedge (\neg A_{t,i,\mathbf{a}} \vee \neg A_{t,i,\Delta}) \wedge (\neg A_{t,i,\mathbf{b}} \vee \neg A_{t,i,\Delta})$$

(d)

$$(B_{t,p} \wedge C_{t,i} \wedge A_{t,i,x}) \Rightarrow (B_{(t+1),q} \wedge A_{(t+1),i,y} \wedge C_{(t+1),i+d})$$

10.

If t is the time taken by T for some input, then we are given that the time taken by U to simulate this computation is $\leq t^3$. But we also know that $t \leq n^2$, where n is the input length. Putting these together, the time taken by U to simulate T on an input of length n is $\leq t^3 \leq (n^2)^3 = n^6$.

Extra exercise: what if we wrap big-O notation around the running times in this exercise?

More specifically:

Suppose that T al[illegible]ps, where n is the length of the input string, and that any computation by a [illegible]hat takes t steps can be simulated by U in $O(t^3)$ steps.

Derive an upper b[illegible], for the time taken by U to simulate T on an input of length n.

Supplementary exercises

11.

From	To	Read	Write	Move
1	3	b	b	R
1	1	a	a	R
3	4	b	b	R
4	2	b	b	R
3	1	a	a	R
4	1	a	a	R

12.

From	To	Read	Write	Move
1	1	a	a	R
1	2	Δ	Δ	R
1	3	b	b	R
3	1	a	a	R
3	4	b	b	R
3	2	Δ	Δ	R
4	1	a	a	R
4	2	Δ	Δ	R

13.

From	To	Read	Write	Move
1	3	a	#	R
3	3	a	a	R
3	2	Δ	Δ	R
3	5	b	$	L
1	4	b	#	R
4	4	b	b	R
4	5	a	$	L
5	5	a	a	L
5	5	b	b	L
5	5	$	$	L
5	6	#	#	R
6	6	b	b	R
6	6	$	$	R
6	7	a	$	L
7	7	b	b	L
7	7	$	$	L
7	8	#	#	R
8	8	a	a	R
8	8	$	$	R
8	5	b	$	L
8	2	Δ	Δ	R

14.

(i). $\mathbf{\underline{a}aaba \to A\underline{a}aba \to Aa\underline{a}ba \to Aaa\underline{b}a \to Aaab\underline{a} \to Aaaba\underline{\Delta} \to Aaab\underline{a} \to Aaa\underline{b} \to Aa\underline{a}b}$
$\mathbf{\to Aa\Delta\underline{b} \to Aa\Delta b\underline{\Delta} \to Aa\Delta\underline{b} \to Aa\underline{\Delta} \to A\underline{a} \to \underline{A}a \to a\underline{a}}$ **HALT** $(3-1=2)$

(ii). $\mathbf{\underline{b}aaa}$ **CRASH** (0 − 3 not defined)

(iii). $\mathbf{\underline{a}abaa \to A\underline{a}b}$ [illegible] $\mathbf{a \to Aaba\underline{a} \to Aabaa\underline{\Delta} \to Aaba\underline{a} \to Aab\underline{a} \to Aa\underline{b}a}$
$\mathbf{\to A\underline{a}ba \to A\Delta}$ [illegible] $\mathbf{ba\underline{\Delta} \to A\Delta b\underline{a} \to A\Delta\underline{b} \to A\underline{\Delta}b \to \underline{A}\Delta b \to \Delta\underline{\Delta}b}$
$\mathbf{\to \Delta\Delta\underline{b} \to \Delta\Delta}$ [illegible] **ALT** $(2-2=0)$

15.

16. (This exercise is Sipser ch. 3.14.)

Introduce two new characters, say \$ and #, into the tape alphabet. The former, \$, will be used to mark both the start and end of the portion of the input tape we have used so far. The latter, #, will be used to help keep track of the Tape Head position, in a manner to be explained below.

For set-up, move all letters of the input string into the queue, with first letter at the head and last letter at the tail, and append \$ to the queue.

Now consider how the Queue Automaton simulates the TM, after this set-up has been done.

Suppose we have a TM transition from state p to state q labelled $x \to y, d$.

If $d =$ Right, we remove (serve) x from the head of the queue and append y to the tail. Then the letter that was after x in the queue is at the head of the queue, which represents the fact that, in the TM, the Tape Head is now reading that letter.

If $d =$ Left, we would like to remove a letter from the tail and prepend it to the head. But these are not allowable queue operations. So, we achieve the same effect as follows.

We create a set of new states r_{pq}, s_{pqw}, t_{pqw} and u_{pq}, one of each of these for each state p, state q, and (for the middle two) each tape letter w. First, we create a transition from p to r_{pq} which just appends the special letter # to the queue. For now, this letter has the meaning, "the previous letter is the one we now want at the head of the queue". But when see # at the head, the previous character has already been served, so we need to remember it, which we do using the new states s_{pqw}. These states are used to repeatedly serve the queue and to remember what the last letter we served was.

For each tape letter w except #, we create a transition from r_{pq} to s_{pqw} which serves w from the queue precisely when the input letter is x and the head of the queue is w, and then appends w to the queue.[2]

At the state s_{pqw}, we have a transition to each $s_{pqw'}$, where w' is also in the tape alphabet (except for #). (It is allowed that $w = w'$, in which case we have a loop.) This transition applies when w' is at the head of the queue, and moves it to the tail.

So, as execution moves through the states s_{pqw}, letters are moved from head to tail, with the most recently moved letter always remembered by means of which of these states we are in.

From each state s_{pqw}, we have a transition from s_{pqw} to t_{pqw} which simply moves # from head to tail of queue. Then, for each tape letter w except for #, we have a transition from t_{pqw} to u_{pq} which appends w to the queue. Once we are in state u_{pq}, we know that # is now immediately *ahead* of the letter we want to be at the head of the queue. So we have loop transitions at u_{pq} for each tape alphabet letter except #, moving these letters from head to tail. Finally, we have a transition from u_{pq} to q which simply removes # from the head of the queue, without appending anything. Then, the letter after it becomes the head of the queue, and this is exactly the one we wanted there.

There is a big contrast here between the effort required to simulate a Right step by the TM (just move one letter from head to tail, by one serve and one append) and that required to simulate a

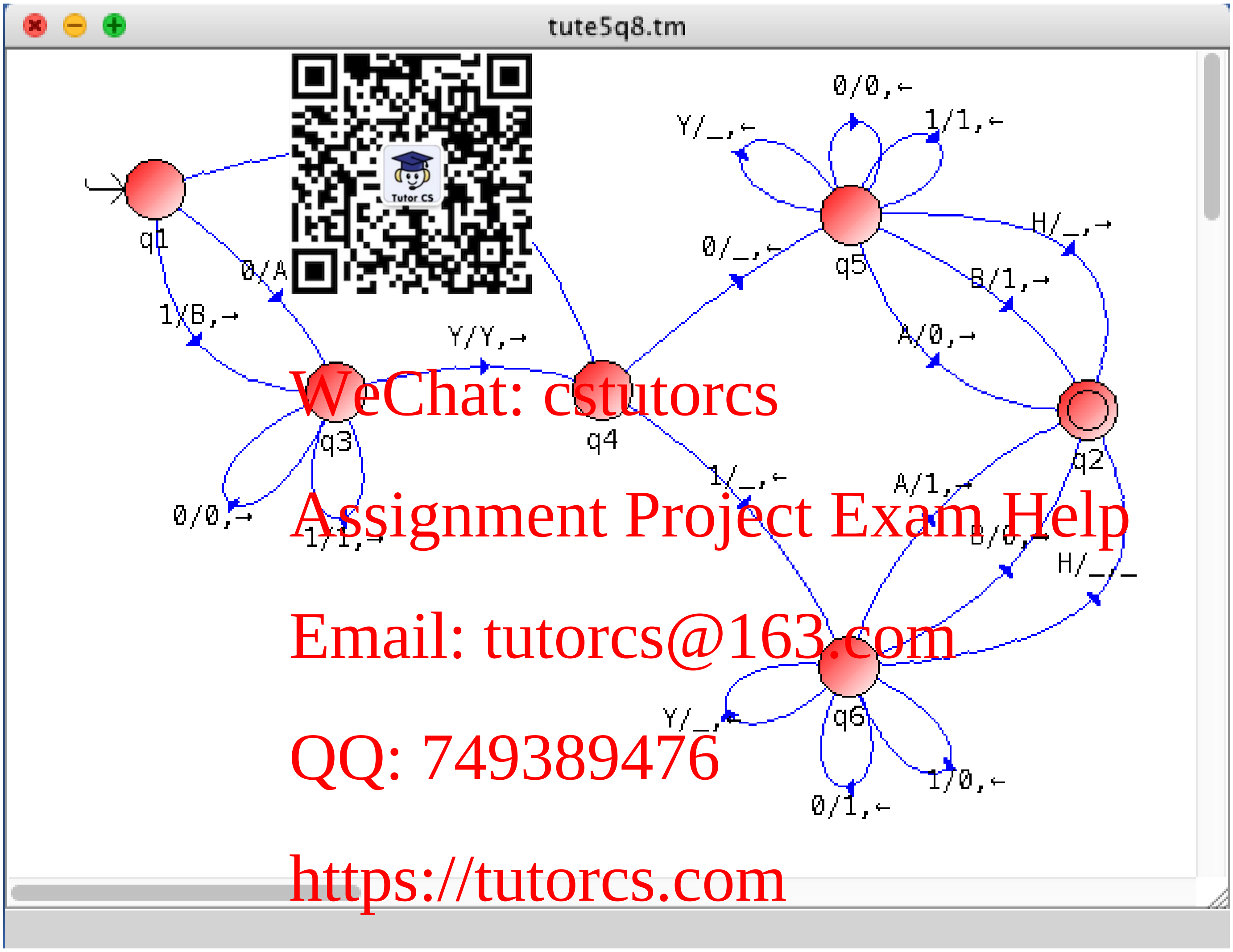
[2] Note that the state r_{pq} has no transition to deal with # at the head of the queue. But that's ok, as # will never be at the head of the queue when we are in this state.

Left step (cycling through virtually the whole queue twice). Nonetheless, each step of the TM takes at most about $2k$ steps of the Queue Automaton where k is the maximum length of TM tape used during the computation.

17.

	current state	[illegible]	[illegible]	[illegible]	direction
/* Mark first cell [illegible]mber its contents), start moving rightwards. */					
Start	1	[illegible]	[illegible]	[illegible]	R
	1	[illegible]	[illegible]	[illegible]	R
	1	[illegible]	[illegible]	[illegible]	R
Accept	2	[illegible]	[illegible]	[illegible]	
/* Move to the cell after the #. */					
	3	0	3	0	R
	3	1	3	1	R
	3	#	4	#	R
/* This bit indicates whether to flip or not. Remember, in next state. Start going left. */					
	4	0	5	Δ	L
	4	1	6	Δ	L
/* Delete **#**, go back to start, all bits unchanged; restore first bit. */					
	5	#	5	Δ	L
	5	0	5	0	L
	5	1	5	1	L
	5	A	2	0	R
	5	B	2	1	R
	5	H	2	Δ	R
/* Delete **#**, go back to start, flipping bits including first. */					
	6	#	6	Δ	L
	6	0	6	1	L
	6	1	6	0	L
	6	A	2	1	R
	6	B	2	0	R
	6	H	2	Δ	R

Here it is in Tuatara:


tute5q8.tm
q1
q2
q3
q4
q5
q6
0/A
1/B,→
Y/Y,→
0/0,→
1/1,→
0/_,←
1/_,←
0/0,←
1/1,←
Y/_,←
H/_,→
B/1,→
A/0,→
A/1,→
B/0,→
H/_,_
Y/_,←
0/1,←
1/0,←