12/15

1/1

Exercise 6.1

We encounter the problem that we want to force an A to be on top of the stack but at the same time not pop A. To work around this, we can add a second transition controlling the A.
It would look like this:

1. transition $q_0 \to q_1$: $\epsilon, A \to A$
2. transition $q_1 \to q_2$: $c, \epsilon \to B$

In the first transition we force an A to be on top of the stack by popping it, but then pushing it right back. Then in the second transition we process the c from the input word and add the B

5.5/6

Exercise 6.2

(a) First we have to introduce two new characters to the alphabet. The # will be used as delimiter between the different tapes and the $\dot{z}$ where $z \in \Gamma$ is used to save the positions of the different tape-heads on our single tape. With those new characters we can then construct our single tape as follows:

1. Move head to the left and write #. (in case it's not allowed to go left, you should shift everything to the right as in 2c) so that you can then write # all the way to the left.
2. Move head to the right and mark the current character with a dot.
3. Move head to the right until a $\square$ is reached and write #,
4. Move head to the right and write a dotted square: $\dot{\square}$
5. Move head to the right and repeat steps 3 and 4 (k-2)-times.
6. Move head all the way to the left (to the first #)

(b) Here we're scanning from the left-most to the (k+1)st # to determine what symbols are under virtual heads. To do that:

1. Move head to the right until an arbitrary dotted symbol $\dot{z}$ is read.
2. "Go" to that symbols state: $q_{\langle z_1 \rangle}$
3. Repeat step 1 and 2 until the (k+1)st # is read (end of last tape is reached). The state is now: $q_{\langle z_1, z_2, \ldots, z_k \rangle}$
4. Move all the way back to the leftmost symbol.

(Note that it would be more efficient to already process the transitions on the way back instead of going back and then go over the tape a third time.)

Nico Bachmann
Ruben Hutter

(c) We're supposed to write a □ to the right of the current head position. However to the right there is a # marking the end of that "tape". We now have to right-shift everything from our current position to the end of the tape to make space for the □. To do this:

1. Move right (to the #) and replace with a new symbol $ to mark the location where the □ should be put later.
2. Move all the way to the right until the (k+1)st # is read. (end of tape)
3. Read current arbitrary symbol z move right and write z how do you store the read symbol? -0.5
4. Move left twice
5. Repeat step 3 and 4 until the $ is read.
6. Replace the $ with □ and move right.
7. ~~write #~~.
8. move left twice.

2/2
Exercise 6.3

(a) Since Turing-recognizable languages are exactly the type-0 languages (slide B12, page 11), we know that the language L recognized by our given DTM M must be of type-0. However type-0 languages are not closed under complement. Meaning that the complement of a type-0 language does not need to be of type-0.
Now since $\bar{L}$ doesn't have to be of type-0, the DTM M' would need to be able to recognize languages that are not type-0. This is not possible!

(b) Let state 1 be the accept state. Let M_1 have only a single transition that goes from state 0 to state 1 by consuming a 0 and going left.
Then let M_2 also have only a single transition that goes from state 0 to state 1 by consuming a 1 but now going right.

M_1 and M_2 both accept the word 0 but are not equal as they go in different directions and are therefore not a "pair".

Our given "product" $M_{1,2}$ then does not include this transition and does not accept 0. Which contradicts the statement in the question.

2.5/4

Exercise 6.4

(a) Encode the rules with the encoding:

states	characters	directions
$q_0 = bin(0) = 00$	$\square = bin(2) = 10$	$L = bin(0) = 00$
$q_{accept} = bin(1) = 01$	$0 = bin(0) = 00$	$R = bin(1) = 01$
$q_{reject} = bin(2) = 10$	$1 = bin(1) = 01$	

and we end up with the following:

encoding has to be of form: from read to write move -1

rules	## "from" # "to" # "read" # "write" # "move"
$(q_0 \to q_{accept}) = \square \to \square, L$	## 0 # 1 # 10 # 10 # 0
$(q_0 \to q_{reject}) = 0 \to 1, R$	## 0 # 10 # 0 # 1 # 1
$(q_0 \to q_{reject}) = 1 \to 0, L$	## 0 # 10 # 1 # 0 # 0

Now we transform our new words (rules) over {0, 1} with the mapping:
$0 \to 00$ $\quad 1 \to 01$ $\quad \# \to 11$

to:
1111 00 11 01 11 0100 11 0100 11 00
1111 00 11 0100 11 00 11 01 11 01
1111 00 11 0100 11 01 11 00 11 00 ✓

and then chain them together in an arbitrary order:
1111001101110100110100110 0 1111001101001100110111101 111100110100110111001100

and have successfully encoded the given Turing machine in a word over {0, 1}. ✓

(b) To decode w we have to reverse the steps from (a).

1. split at the ## and # separator encoded as 1111 or 11:

   1111 00 11 00 11 00 11 00 11 01
   1111 00 11 01 11 01 11 01 11 01
   1111 00 11 0100 11 0100 11 00 11 01

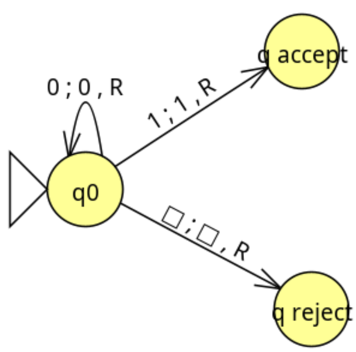
2. decode the character pairs back to their integer values:

   ## 0 # 0 # 0 # 0 # 1
   ## 0 # 1 # 1 # 1 # 1
   ## 0 # 10 # 10 # 0 # 1

3. decode the integer values back to their characters:

   $q_0 \to q_0 = 0 \to 0, R$
   $q_0 \to q_{accept} = 1 \to 1, R$
   $q_0 \to q_{reject} = \square \to \cancel{\square}, R$

   -0.5 should be 0



1/2 **Exercise 6.5**

True, see word problem for type-2

1. False because a language L is Turing-decidable, iff both L and $\bar{L}$ are Turing-recognizable. However type-2 languages are not closed under complement so $\bar{L}$ does not have to be of type-2 and therefore also doesn't have to be Turing-recognizable. And if that's the case, L is not Turing-decidable which contradicts the given statement.

2. True because if a language L is Turing-decidable, L and $\bar{L}$ both have to be Turing-recognizable and therefore be of type-0. And because they have to be type-0, they also need to have a grammar. ✓