Homework #5. Show that a language can be recognized by a deterministic queue automaton iff the language is Turing-recognizable. A queue automaton is like a push-down automaton except that the stack is replaced by a queue.

Proof. We want to show that deterministic queue automata (DQAs) and turing machines (TMs) are equivalent. In order to show equivalency we must show:

1. Given a TM, we can generate a DQA that recognizes the same language as the TM.
2. Given a DQA, we can generate a TM that recognizes the same language as the DQA.

1. In our DQA, we can push (adding a symbol) to the right-hand end of the queue and pull (removing a symbol) from the left-hand end of the queue (illustration below). We chose this right-left convention for a more intuitive representation of the TM tape.

Queue layout

![Queue layout diagram]

First, we will read the entire input tape, pushing each symbol onto the queue. Then we push "$\$" to mark the beginning of the simulated TM tape.

Queue representation of the TM tape

![Queue representation of the TM tape]

For example, given the TM tape $abc$, we will generate the queue $abc\$$. The symbol currently at the head of the queue ($a$) is the symbol under the head of the TM. In general, everything to the right of the head of the TM is represented by the queue contents between $a$ and $\beta$

To prove that we can generate a DQA from a given TM, we must show that we can simulate all the operations of a given TM with a DQA. Specifically, given $a, b, x, y$ in the tape alphabet, we must show that we can simulate the TM transitions $a \rightarrow x, R$ and $b \rightarrow y, L$ (move right and move left, reading and writing an arbitrary symbol) with queue operations.
\[ a \rightarrow x, R \quad \text{This operation is easy to simulate.} \]

\[ \begin{array}{c}
\text{head} \\
\text{tail} \\
\downarrow \\
\] \\
\[ a \ b \ c \\
\] \\
\[ \Rightarrow \\
\] \\
\[ x \ b \ c \\
\] \\
\[ \text{Move right} \]

\[ \begin{array}{c}
\text{head} \\
\text{tail} \\
\downarrow \\
\] \\
\[ a \ b \ c \$ \\
\] \\
\[ \Rightarrow \\
\] \\
\[ \backarrow a \leftarrow b \ c \$ \leftarrow x \\
\] \\
\[ \Rightarrow \\
\] \\
\[ b \ c \$ \ x \\
\] \\
\[ \text{TM} \]

We pull \( a \) from the queue and push \( x \) to the queue. This effectively moves the head one character to the right and preserves the rest of the contents the simulated TM tape. If we read a “\$”, we can undo the operation using the “Move Left” operation we define below, push a blank, and move left again. This allows us to move as far right on the TM tape as necessary.

\[ b \rightarrow y, L \quad \text{This operation is more difficult, as we must effectively move the character at the tail of the queue to the head. In order to do this, we must be able to “read ahead”. The general idea is this (illustrations below):} \]

(1) Given tape contents \( bcdefg...\$a \), we push a “\#” to mark the current tape position, yielding \( bcdefg...\$a\# \). (2) We then encode each symbol, starting at the current head position, as a 2-tuple of the previous and current symbol. Each symbol may be “remembered” with states, since we know the tape alphabet and it is finite. The very first symbol (\( b \) in our example) is “remembered” as its newly-written value (*\( \# \) in our example). When we get to “\#”, instead of pushing (\( \#, a \)) (in our example), we push “\#”, then \( a \). (3) Then we pull 2-tuples and push single symbols (the first element of the tuples) until we read “\#” again. (4) We discard the “\#”. This provides the “move left” operation.

\[ \text{TM} \]

\[ a \ b \ c \\
\] \\
\[ \Rightarrow \\
\] \\
\[ a \ y \ c \\
\] \\
\[ \text{Move Left} \]

\[ \begin{array}{c}
(1) \\
(2) \\
(3) \\
(4) \\
\end{array} \]

\[ \begin{array}{c}
\] \\
\[ b \ c \ d \ e \ f \ g \ ... \$a \] \\
\[ \Rightarrow \\
\] \\
\[ b \ c \ d \ e \ f \ g \ ... \$a \# \\
\] \\
\[ \Rightarrow \\
\] \\
\[ b \ c \ (c,d) \ (d,e) \ (e,f) \ (f,a) \ (a,...) \ (#,\$) \ (s,a) \ # \ a \\
\] \\
\[ \Rightarrow \\
\] \\
\[ (b,c) \ (c,d) \ ... \ (\$,#) \ (s,a) \ # \ a \\
\] \\
\[ \Rightarrow \\
\] \\
\[ # \ a \ b \ c \ d \ e \ f \ g \$ \\
\] \\
\[ \Rightarrow \\
\] \\
\[ \# \ a \ b \ c \ d \ e \ f \ g \$ \\
\] \\
\[ \Rightarrow \\
\] \\
\[ a \ b \ c \ d \ e \ f \ g \$ \\
\] \\
\[ \text{Given this left-shift operation, we can simulate moving the TM head to the left by reading once (to determine if we’re reading the correct input symbol), then shifting left twice (once} \]
to undo the right-shift caused by reading the symbol, once to move to the left of the original position). Note that we must abort the second shift if we read a "\$" (indicating we are already at the left of the TM tape).

2. To prove that we can generate a TM from a given DQA, we must show that we can simulate the two queue operations (push and pull) with a two-tape TM. Let the first tape be the DQA input, and let the second tape be the (simulated) queue. To push \( x \), the head goes to the first blank and writes \( x \). To pull, the head moves to the "\$", reads the first non-"#" symbol, and writes a "#" (indicating the symbol has been pulled).

Simulating a DQA with a TM

\[
\begin{align*}
\text{TM:} & \quad \begin{array}{c} \$ \end{array} \ a \ b \ c \\
0 & \text{push } a \text{ : } \begin{array}{c} \$ \end{array} a \ b \ c \ d \\
1 & \text{pull } a \text{ : } \begin{array}{c} \# \end{array} b \ c \ d \\
2 & \text{push } e \text{ : } \begin{array}{c} \# \# \end{array} b \ c \ d \ e \\
3 & \text{pull } b \text{ : } \begin{array}{c} \# \# \$ \end{array} b \ c \ d \\
4 & \text{pull } b \text{ : } \begin{array}{c} \# \# \# \# \$ \end{array} b \ c \ d \ e \\
& \text{and so forth...}
\end{array}
\]

Conclusion  By showing that we have generate an equivalent DQA from any given TM, and an equivalent TM from any DQA, we have shown that DQAs and TMs are equivalent.