Middle East Technical University

Computer Engineering Department

CENG 280
Formal Languages and Abstract Machines

Homework 3

Devrim Çavuşoğlu
2010023

Date: June 13, 2022

Question 1

We have language L, where $L = \{0^N 1^N | N \geq 1\}$. Let M be a TM recognizing L, and $M = (K, \Sigma, \delta, q_0, H)$, where

- $K = \{q_0, q_1, q_2, q_3, q_4, h\}$
- $H = \{h\}$

The intuition is to design the machine in a way that it reads/writes from outer towards inner, this way it can easily follow the constraints that the machine is not allowed to write anything but Σ. This is essential because the machine is not allowed to write any symbol other than the alhapbet (including blank symbol), so that it cannot mark any place in any way in the tape. However, if we let $s \in L$, then the middle of s is naturally determined (or marked) because there is a jump to another symbol namely from symbol 0 to symbol 1 (or vice-versa in backward direction), and this information is valuable to be considered within the given constraint. The machine M is illustrated in Figure 1.

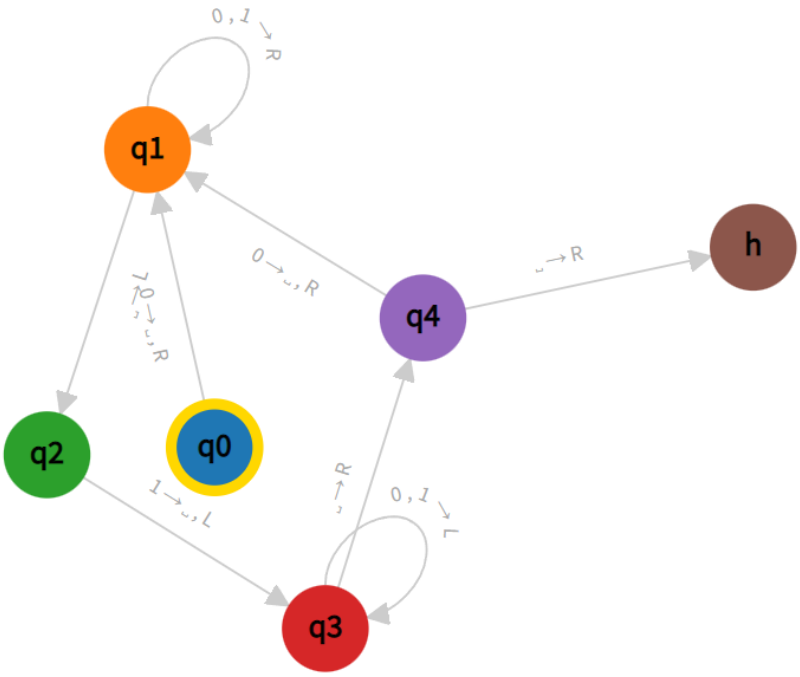

Figure 1: Model design for the machine M.

The procedure can be summarized as follows, the state associated with the statement is given at the beginning of the each statement:

1. (q_0) Start from the left most symbol.
2. (q_0, q_4) If encounter symbol 0, then erase (replace with blank symbol) the current symbol. Otherwise, **halt and reject**.
3. (q_1) Move the head to the right most symbol of the input, which is required to be 1 as the language L dictates.

4. (q_2) If encounter symbol 1, erase the symbol and move the head to the left. Otherwise, **halt and reject**.

5. (q_3) Move the head to the left, if encounter blank symbol (which means there is no more 0 to be replaced and we just erased the last 1, so the number of 0 and 1 are equal), then **halt and accept**.

6. (q_4) If not halted, move the head to the left most symbol of the remaining input (first non-blank symbol) and go back to step 2.

Practically q_0 and q_4 functions in a similar way, almost identical indeed. The need for a separate state q_4 is just to make sure that e (empty string) is not accepted as $e \notin L$. Thus, we use q_0 as a start state, and only at the beginning, and if we loop back to the left most-end we use q_4 for the remaining operations. The machine decision is given for some example inputs in Figure 2.

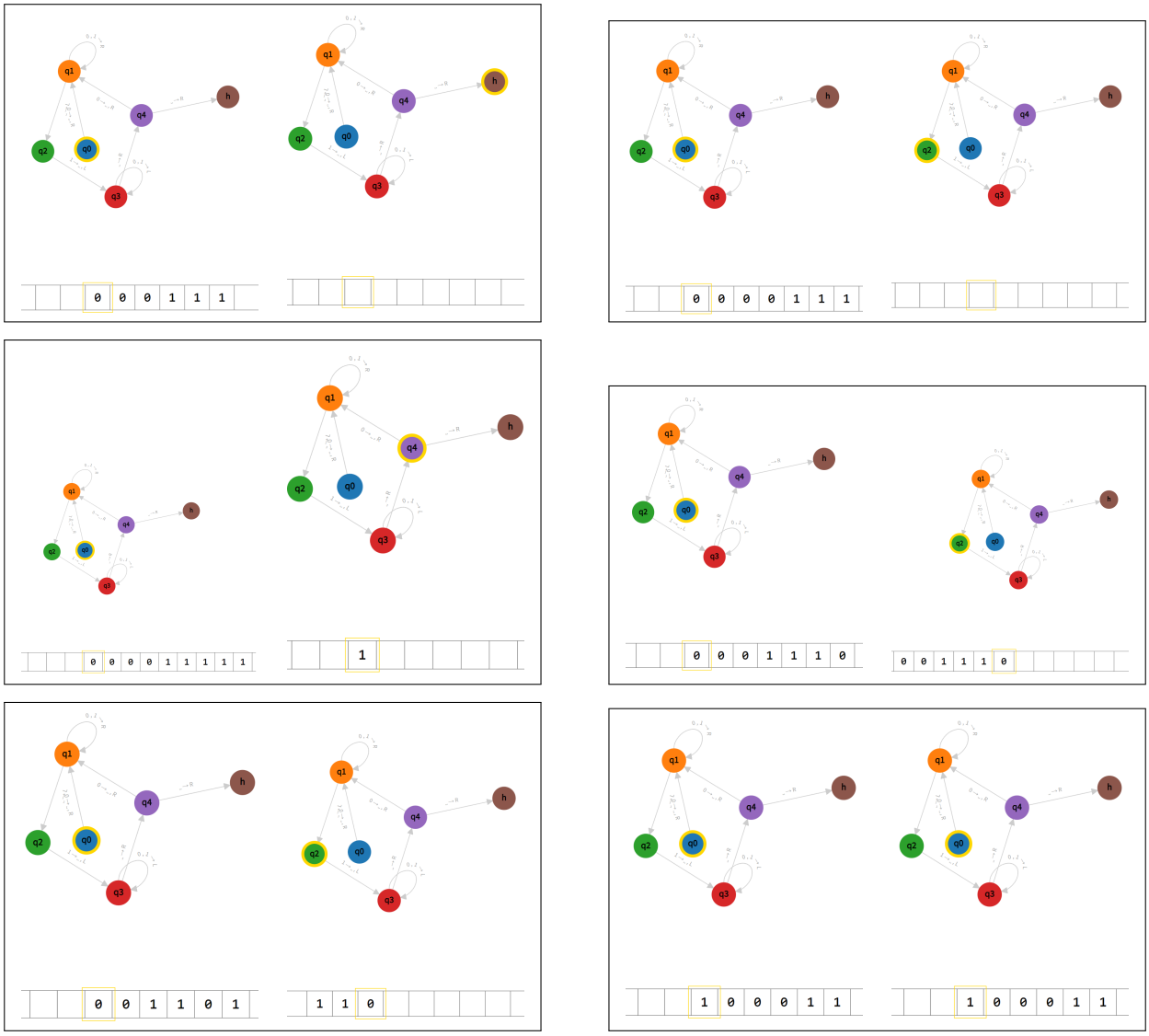

Figure 2: Some example inputs that are fed to the machine M. The current state is marked with yellow ring.

Question 2

Let M_2 be the machine computing $f(w) = ww^R$, $w \in \{0,1\}^+$. $M_2 = (K_2, \Sigma, \delta_2, q_0, H_2)$, where

- $K_2 = \{start, swipe\_right, zero\_writer, one\_writer, left\_swipe, fix, h\}$
- $\Sigma = \Sigma(w) \cup \{Z, O\}$
- $H_2 = \{h\}$

Note that, unlike the machine M in the Question 1, we do not have the constraint that limits the alphabet $\{0,1\}$, so we will use two additional symbols $\{Z, O\}$ indicating zero (Z) and one (O) respectively. The diagram of the machine M_2 is given in Figure 3.

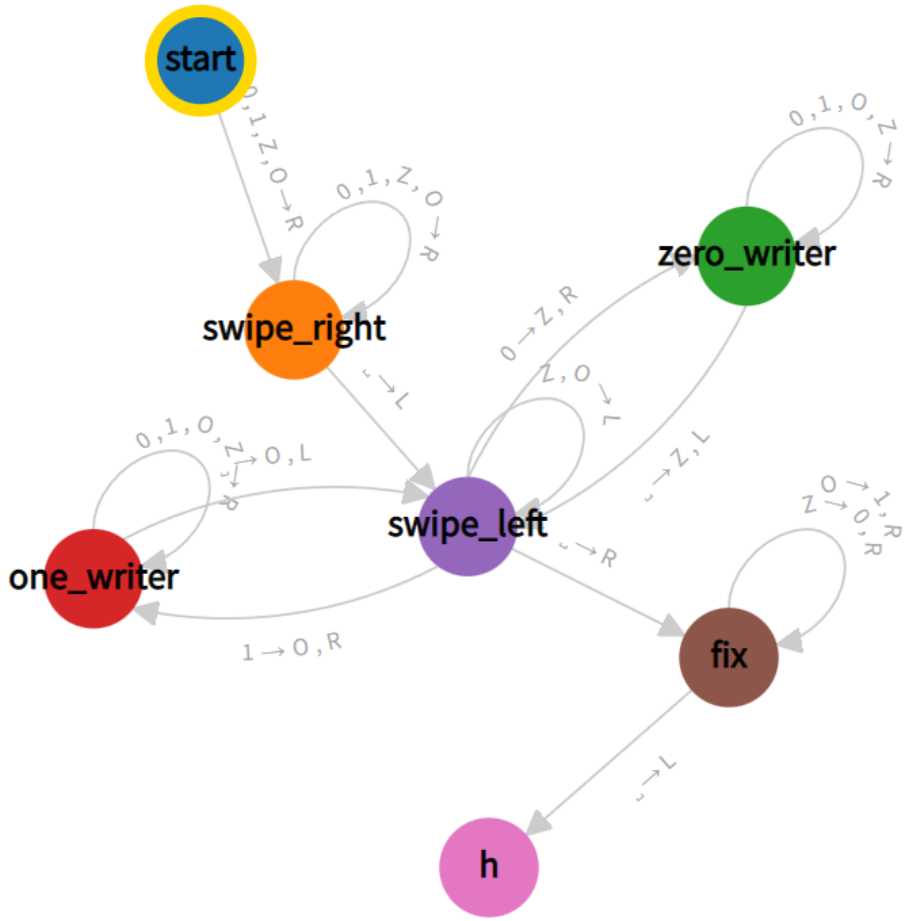

Figure 3: Model design for the machine M_2.

The procedure can be summarized as follows, the state associated with the statement is given at the beginning of the each statement:

1. (*start*) Start from the left most symbol, if encounter a blank symbol (which means input is an empty string) then **halt and reject**, otherwise go to step 2.
2. (*swipe_right*) Swipe the head the the right most symbol ignore any symbol except the blank symbol. If encounter a blank symbol, move the head left and go to step 3.

3. ($swipe\_left$) Read backwards starting with the current symbol. If the symbol $c \in \Sigma(w)$, then replace the symbol with an associated indicator (Z (0) or O (1)). Ignore symbols Z and O and move the head until hitting 0, 1, or blank symbol. If encounter 0 or 1 proceed with step 4. If encounter a blank symbol skip the step 4 and go to step 5.

4. ($zero\_writer, one\_writer$) Move the head towards right until hitting the first blank symbol, ignore any other symbol in Σ. Then, replace the blank symbol with Z ($zero\_writer$) or O (one_writer) depending on the state. Go back to step 3.

5. (fix) This step means that we have finished writing the indicators completely and we are at the left most symbol. Move the head towards the right. If encounter Z replace with 0, if encounter O replace with 1. If encounter a blank symbol, move the head to the left and go to step 6.

6. (h) We have finished computing f and the machine in the accept state h. Also, the head is pointing to the last symbol of computed $f(w)$.

The snapshots of the machine for some example inputs is given in Figure 4.

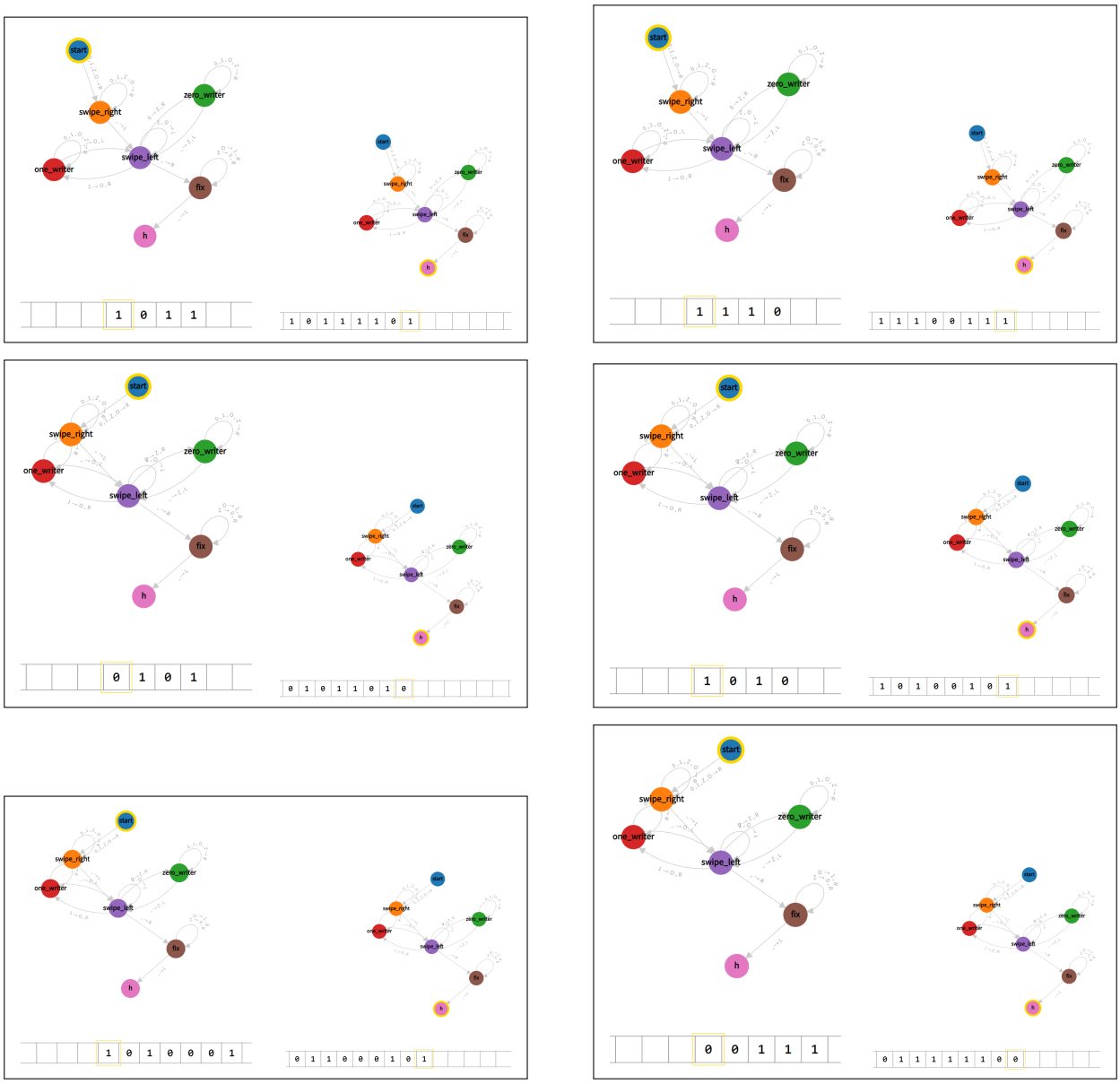

Figure 4: Some example inputs that are fed to the machine M_2. The current state is marked with yellow ring. Note that for some long input strings the whole output string that the machine computed may not be visible.

Question 3

For convenience we will assume that the machine and configurations set in Q1 is valid and further assume that we try to design a 2-tape TM for recognizing the language L as an example. Let M_3 be a Multitape TM ($m = 2$) where $M_3 = (K', \Sigma', \delta', s, H')$ and $\delta' : (K' - H') \times \Sigma'^2 \longrightarrow K' \times (\Sigma \cup \{\leftarrow, \rightarrow\})^2$. The machine has two heads for $tape_i^{(j)}$ where $i = 0, 1$ indicating the tape id/number, and $j \in N$ indicating the j^{th} position in the $tape_i$. We assume again for convenience that the input is given in the first tape, and that the second tape is initially blank. The model for M_3 is given in Figure 5 with some example input $s \in L$.

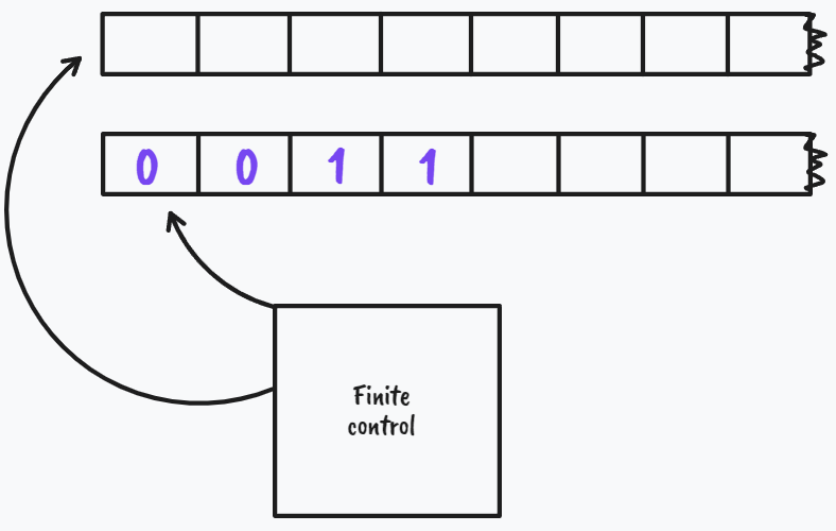

Figure 5: Model design and initial configuration for the machine M_3 with input $s = 0011$.

We here will not go into details and how the computations could be done in M_3 to recognize L, but a pseudo-ish procedure set (not the exact state definitions) can be as follows:

1. Read the input in the first tape until encountering 1 (basically read zeros towards right), meanwhile write the zeros in the second tape at the same time. In short, copy the first part of the input of zeros into the second tape. When encounter 1 go to step 2.

2. Move towards right in the first tape by reading 1s (crash otherwise), for each one we now erase zeros in the backward direction in the second tape. If the machine cannot erase a corresponding zero in the second tape, crash (it means more 1s than 0s). If you hit a blank symbol, go to step 3.

3. If the second tape is blank when the input is read in the first tape, then accept the input, otherwise reject.

Proof. Now, putting aside the example above, we argue that any procedure set that is followed by a multitape TM can be simulated by an ordinary TM (single tape). The first issue that we need to consider is how to simulate the multitape ops in a single tape. Generally, this is achieved by simulating the k-tape in a single tape where we need some kind of "virtualization". In a simple terms this can be achieved by dividing the single input tape into separate chunks by a delimiter. With the help of the delimiter symbol now the ordinary TM has the ability to chose between different chunks/virtual tapes, this can easily be achieved by adding a new state with the delimiter symbol, so we extend Σ as $\Sigma \cup \{\#\}$, where # is the delimiter symbol.

We have now ability of having different chunks/virtual tapes in a single tape, but we have yet another problem. The problem is a natural consequence of shifting to k-tape machine from m-tape machine where $k < m$, in our case $k = 1$. The obvious problem here is that we lost the multiple tape heads that are reading from distinct tapes. However, this problem can be eliminated with the help of the new symbols (this is a different approach than the one stated in [1]). We can simply introduce new symbols for each of the symbols in the alphabet where these new symbols have a corresponding symbol in alphabet and they tell us the head position in each separate chunk. The simple idea for these symbols for position information, or *pos_symbols* in short, is, for example, to extend the alphabet with the underlined symbols $\Sigma \cup \{\#, pos\_symbols\}$. For alphabet of L, $pos\_symbols = \{\underline{0}, \underline{1}\}$, and since $|pos\_symbol| = |\Sigma|$, we have information for both the head position and the associated alphabet symbol. A snapshot for an example simulation is given in Figure 6.

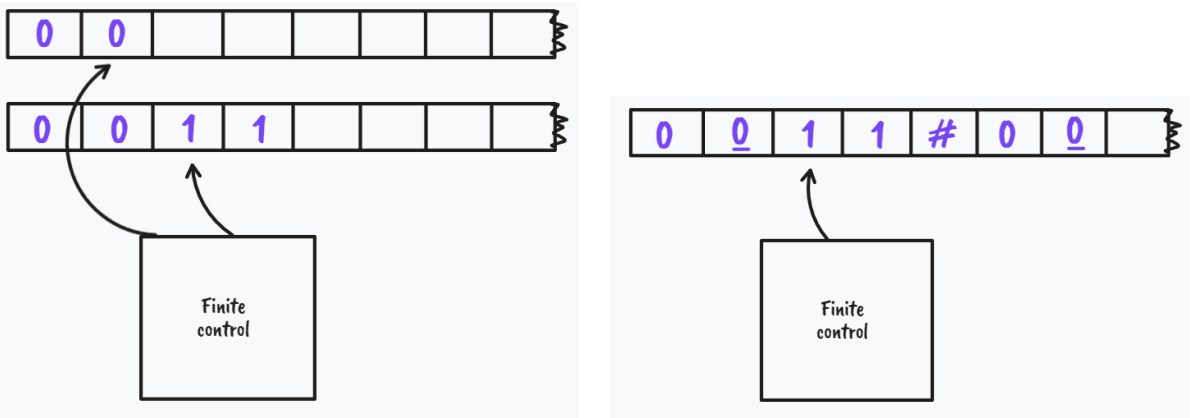

Figure 6: Illustration of the simulation of 2-tape machine with a single tape machine with input $s = 0011$.

A last but not least issue we need to consider is that the chunk sizes, or what will happen when machine needs to write in a chunk, e.g the first chunk which is initially filled with the input and the delimiter is set right after the input. For example if the machine needs to write immediately at the end of the input and within the first chunk, it basically cannot, it has to erase a symbol as it does not fit. This is a serious problem to be taken into account as the simulation is not complete without addressing this problem. For this problem, the immediate solution is when the machine is writing and there are currently no blank symbols in the chunk, the machine can shift the values to the right by 1 unit, and then it can append/write to the blank symbol. Hence, we need state or states to handle this problem, when the size is not fit for an append operation, provide new spot by shifting the input.

We have addressed all the issues so far and the simulation can now be generalized for any multitape machine. As the simulation is complete let's break down the time complexity. We assume the machine always halts and thus is deciding rather than recognizing.

For deciding language L, the 2-tape in the worst case halts after reading all of the input which is $O(tn)$. The single tape machine has additional operations to simulate the 2-tape machine one of which is to chunk-wise operations where the single-tape TM performs this in, say in pt steps. It also has the overhead for finding the head position in the chunk, say in qt steps. Lastly, it may need to shift symbols to make spot for writing, say in rt steps. This brought us $O(t(n + pt + qt + rt)) = O(t(n + (p + q + r)t))$ and which reduces to $O(tn + t^2)$, where $p, q, r \in R$ are constants. □

References

[1] Harry R Lewis and Christos H Papadimitriou. Elements of the theory of computation. *ACM SIGACT News*, 29(3):62–78, 1998.