Efficient Universal Turing machine simulation in $\mathcal{O}(T \log T)$-time*

Kenenbek Arzymatov

April 27, 2018

Introduction

Turing Machine

It is possible to think about a Turing Machine (TM) as model of simple computer. Due to its simplicity computer scientists and mathematicians can use it to study the theoretical properties of computation. For instance, they managed to discover that no matter how powerful computer is, it is always achievable to translate its programs to work on the Turing Machine or that all programming languages can do the same things.

One of the central concepts in compution is "algorithm" because algorithm is a way for solving a particular problem. Let f be a function that takes a string of zeros and ones (element of the set $\{0,1\}^*$), here *star*-operation means:

$$A^* = \{x_0 x_1 \dots x_k | k \geq 0 \text{ and each } x_i \in A\}$$

and outputs either 0 or 1. An algorithm for computing f is a set of mechanical rules, such that by following them it is possible to compute $f(x)$ given any input $x \in \{0,1\}^*$. The set of rules being followed is fixed (the same rules must work for all possible inputs) though each rule in this set may be applied arbitrarily many times. Each rule involves one or more of the following "elementary" operations:

- Read a symbol on the inpute tape
- Read a symber on the work tape if TM has it
- Write a symbol to the work tape
- Eiter stop and output 0 or 1, or choose a new rule for the next step.

The *running time* of an algorithm is the number of these basic operations performed. It's measured in asymptotic terms. For example, if a machine runs in time $T(n)$ then it executes at most $T(n)$ basic operations time on inputs of length n.

Figure 1 is an illustration for 3-tape TM with an input, work and output tapes.

Definition. TM M is described by a tuple (Γ, Q, δ) containing:

[0]Presentation slides are accessible here

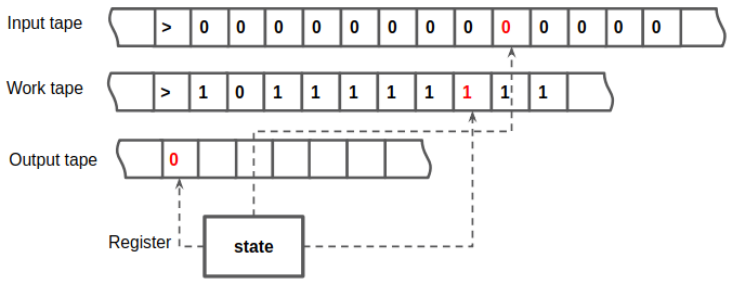


Figure 1: Example of Turing machine with three tapes: an input tape, a work tape and an output tape.

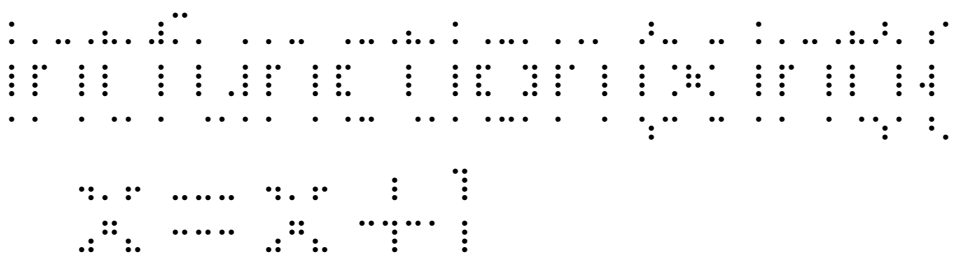
- A finite set of the symbols that M's tapes can contain. It contains a designated "blank" symbol, denoted $\square$; a designated "start" symbol, denoted $\triangleright$; and the numbers 0 and 1. This set is called the alphabet of TM M.
- A finite set Q of possible states M's register can be in. Q contains a designated start state, denoted q_{start} , and a designated halting state, denoted q_{halt}
- A function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$, where $k \geq 2$, describing the rules M use in performing each step; L, S, R stand for move Left, Stay in place or move Right accordingly. This function is called the transition function of M.

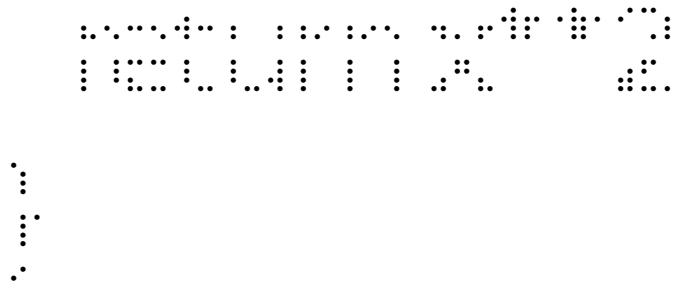
Useful statements:

1. Behavior of a Turing machine is determined by its transition function. For example, $\delta(q, (\sigma_1, \ldots, \sigma_k)) = (q', \sigma_2', \ldots, \sigma_k'), z)$ where $z \in \{L, S, R\}^k$ like $LS \ldots SR$ or $RR \ldots LS$. For example, if TM has 3 work tapes and output tape z can be like $SRRR$ which means stay in place for the first tape and move heads right for the other ones.
2. An algorithm (machine) can be represented as a bit string (consisting only of zeros and ones) by the means of encoding. For example, there is a function which takes an integer number and return an incremented one:

```
int function (int x){
    x = x + 1;
    return x**2;
}
```

The code above can be encoded with Braille font (just for illustrative purposes only):





The result of converting the dots to ones and the blanks to zeros is the bitstring.

$$\underbrace{1000010011001011111001000011111100010001111010\ldots101001001011010011}_{\text{length}}$$

3. Every string in $\{0,1\}^*$ represents some Turing machine.

4. An algorithm/machine can be used as a possible input to another algorithm — this makes the boundary between input, software, and hardware very thin.

5. Every TM is represented by infinitely many strings.

6. M_α is the TM whose representation as a bit string is α.

Definition. Let $f : \{0,1\}^* \to \{0,1\}^*$ and let $T : \mathbb{N} \to \mathbb{N}$ be some functions, and let M be a Turing machine. We say that M computes f if for every $x \in \{0,1\}^*$, whenever M is initialized to the start configuration on input x, then it halts with $f(x)$ written on its output tape. We say M computes f in $T(n)$-time if its computation on every input x requires at most $T(|x|)$ steps.

The universal Turing machine

Definition.There is a universal Turing machine U that can simulate any other Turing machine given its bit representation. Given a pair of bit strings (x, α) as input, this machine simulates the behavior of M_α on input x. This simulation is very efficient: If the running time of M_α was $T(|x|)$, then the running time of U is $O(T(|x|) \log T(|x|))$. Turing was the first to observe that general-purpose computers are possible, by showing a universal Turing machine that can simulate the execution of every other TM M given M's description as input.

Nowadays people who has a programming background easily can understand that the same function can be implemented on any programming language. But it is good to remember why it was once counterintuitive. The parameters of the universal TM are fixed—alphabet size, number of states, and number of tapes. The corresponding parameters for the machine being simulated could be much larger. The reason this is the ability to use encodings. Even if the universal TM has a very simple alphabet, this suffices to represent the other machine's state and transition table on its tapes and then follow along in the computation step by step.

Efficient universal Turing machine

Theorem.There exists a TM U such that for every $x, \alpha \in \{0,1\}^*, U(x, \alpha) = M_\alpha(x)$, where M_α denotes the TM represented by α. Moreover, if M_α halts on input x within T steps then $U(x, \alpha)$ halts within $CT \log T$ steps, where C is a number independent

of $|x|$ and depending only on M_α's alphabet size, number of tapes, and number of states.

Proof

The proof of the theorem is conducted in two steps:

1. Construction of TM that works for quadratic time
2. Using ideas from a previous step, do a modification for TM such way that it work for $T \log T$ time.

a universal TM U such that given an input x and a description of a TM M that halts on x within T steps, U outputs $M(x)$ within $\mathcal{O}(T \log T)$ time (where the constants hidden in the $\mathcal{O}$ notation may depend on the parameters of the TM M being simulated).

U will use its input and output tape in the same way M does and will also have extra work tapes to store M's transition table and current state and to encode the contents of M's work tapes.

Now we show a way to encode all of M's work tapes in a single tape of U, which we call the main work tape of U. Let k be the number of tapes that M uses (apart from its input and output tapes) and its alphabet. We may assume that U uses the alphabet Γ^k (as this can be simulated with a overhead depending only on $k, |\Gamma|$). Thus we can encode in each cell of U's main work tape k symbols of Γ, each corresponding to a symbol from one of M's tapes. This means that we can think of U's main work tape not as a single tape but rather as k parallel tapes.

Encoding M's tapes on U's tape

We encode the information (the content of M's work tapes) using "buffer zones": Rather than having each of U's parallel tapes correspond exactly to a tape of M, we add a special kind of blank symbol $\diamond$ to the alphabet of U's parallel tapes with the semantics that this symbol is ignored in the simulation.

For convenience, we think of U's parallel tapes as infinite in both the left and right directions (this can be easily simulated with minimal overhead: see Claim 1.8). Thus, we index their locations by $0, \pm 1, \pm 2, \ldots$. Normally we keep U's head on location 0 of these parallel tapes. We will only move it temporarily to perform a shift when, following our general approach, we simulate a left head movement by shifting the tape to the right and vice versa. At the end of the shift, we return the head to location 0.

We split each of U's parallel tapes into zones that we denote by $R_0, L_0, R_1, L_1, \ldots$. The cell at location 0 is not at any zone. Zone R_0 contains the two cells immediately to the right of location C (i.e., locations $+1 and + 2$), while Zone R_1 contains the four cells $+3, +4, +5, +6$. Generally, for every $i 1, \text{Zone} R_i \text{contains the} 2 \cdot 2^i$ cells that are to the right of Zone R_{i-1} (i.e., locations $[2i + 1 - 1, ..., 2i + 2 - 2]$). Similarly, Zone L_0 contains the two cells indexed by -1 and -2, and generally Zone L_i contains the cells $[-2^{i+2} + 2, \ldots, -2^{i+1} + 1]$. We shall always maintain the following invariants:

- Each of the zones is either empty, full, or half-full with non- $\diamond$ is either $0, 2^i, or 2 \cdot 2^i$ and the same holds number of symbols in zone R_i that are not for L_i .

(We treat the ordinary $\square$ symbol the same as any other symbol in Γ, and in particular a zone full of $\square$'s is considered full.) We assume that initially all the zones are half-full. We can ensure this by filling half of each zone with $\diamond$ symbols in the first time we encounter it.

- The total number of non-$\diamond$ symbols in $R_i \cup L_i$ is $2 \cdot 2i$. That is, either R_i is empty and L_i is full, or R_i is full and L_i is empty, or they are both half-full.
- Location 0 always contains a non-$\diamond$ symbol.

Performing a shift

The advantage in setting up these zones is that now when performing the shifts, we do not always have to move the entire tape, but we can restrict ourselves to only using some of the zones. We illustrate this by showing how U performs a left shift on the first of its parallel tapes (see also Figure 1.9):

1. U finds the smallest i_0 such that R_{i_0} is not empty. Note that this is also the smallest i_0 such that L_{i_0} is not full. We call this number i_0 the *index* of this particular shift.
2. U puts the leftmost non-$\diamond$ symbol of R_{i_0} in position 0 and shifts the remaining leftmost $2^{i_0}-1$ non-$\diamond$ symbols from R_{i_0} into the zones $R_0, \cdots, R_{i_0-1}$ filling up exactly half the symbols of each zone. Note that there is exactly room to perform this since all the zones $R_0, \cdots, R_{i_0-1}$ were empty and indeed $2^{i_0} - 1 = \sum_{j=0}^{i_0-1} 2^j$
3. U performs the symmetric operation to the left of position 0. That is, for j starting from i_0-1 down to 0, U iteratively moves the $2 \cdot 2j$ symbols from L_{j+1} to fill half the cells of L_{j+1} .Finally, U moves the symbol originally in position 0 (modified appropriately according to M's transition function) to L_0.
4. At the end of the shift, all of the zones $R_0, L_0, \cdot, R_{i_0-1}, L_{i_0-1}$ are half-full, R_{i_0} has 2^{i_0} fewer non-$\diamond$ symbols, and L_i has 2^i additional non-$\diamond$ symbols. Thus, our invariants are maintained.
5. The total cost of performing the shift is proportional to the total size of all the zones $R_0, L_0, R_{i_0}, L_{i_0}$. That is, $\mathcal{O}(\sum_{j=0}^{i_0} 2 \cdot 2^j) = \mathcal{O}(2^{i_0})$*operations*.

After performing a shift with index i the zones $L_0, R_0, L_{i-1}, R_{i-1}$ are half-full, which means that it will take at least 2^{i-1} left shifts before the zones $L_0, \cdots, L_{i-1}$ become empty or at least 2^{i-1} right shifts before the zones $R_0, \cdots, R_{i-1}$ become empty. In any case, once we perform a shift with index i, the next 2^{i-1} shifts of that particular parallel tape will all have index less than i. This means that for every one of the parallel tapes, at most a $1/2^i$ fraction of the total number of shifts have index i. Since we perform at most T shifts, and the highest possible index is $\log T$, the total work spent in shifting U's k parallel tapes in the course of simulating T steps of M is

$$\mathcal{O}(k \cdot \sum_{i=1}^{\log T} \frac{T}{2^{i-1}} 2^i) = \mathcal{O}(T \log T). \blacksquare$$

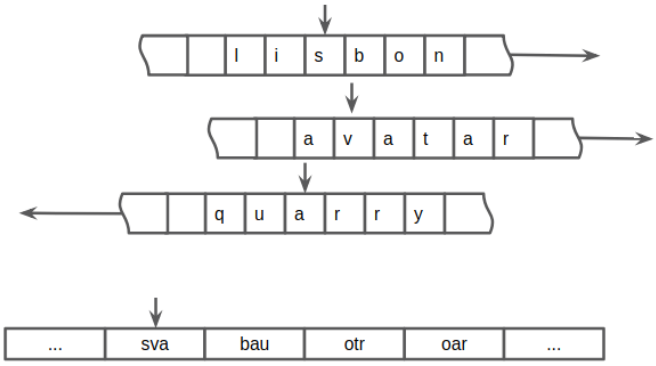


Figure 2: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

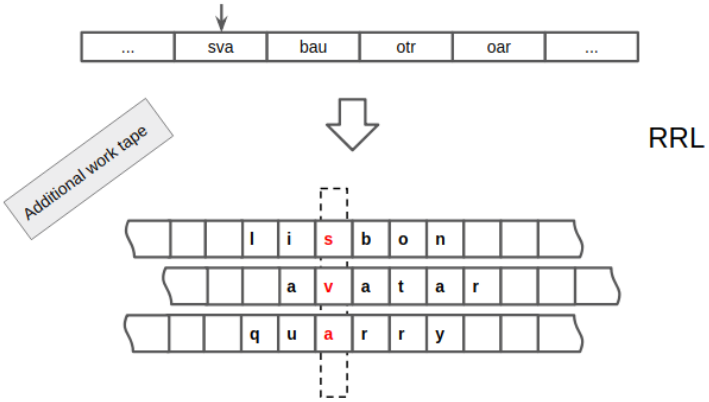


Figure 3: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

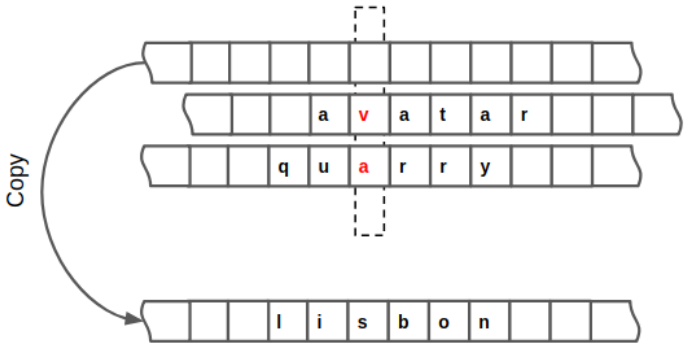


Figure 4: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

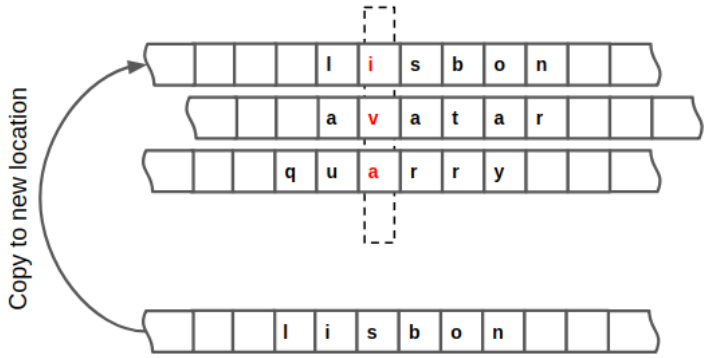


Figure 5: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

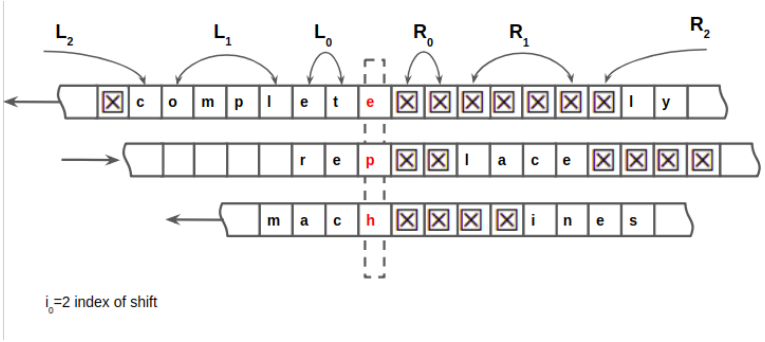


Figure 6: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

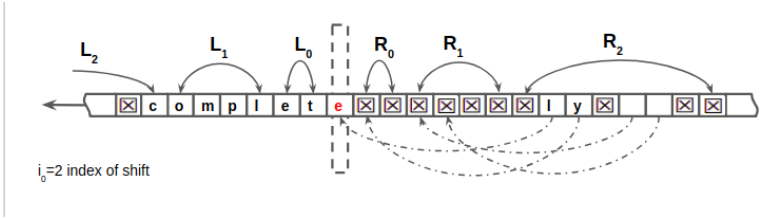


Figure 7: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

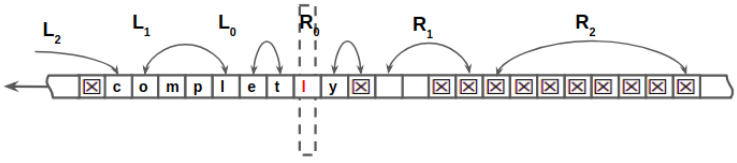


Figure 8: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

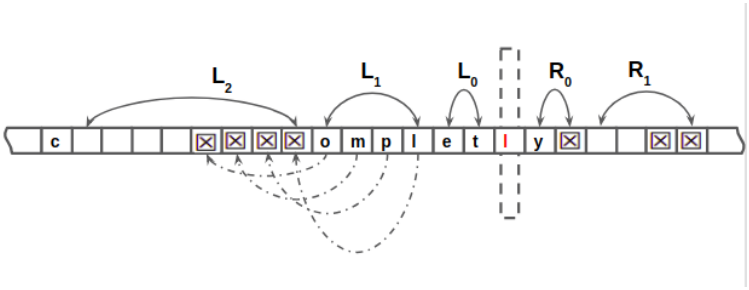


Figure 9: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

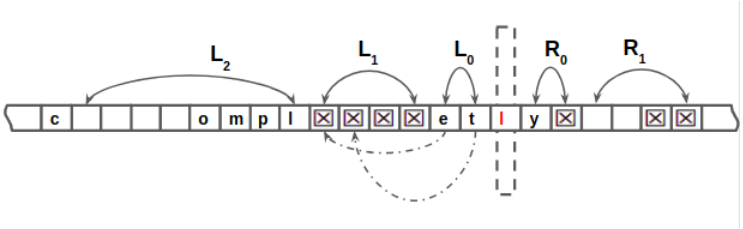


Figure 10: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

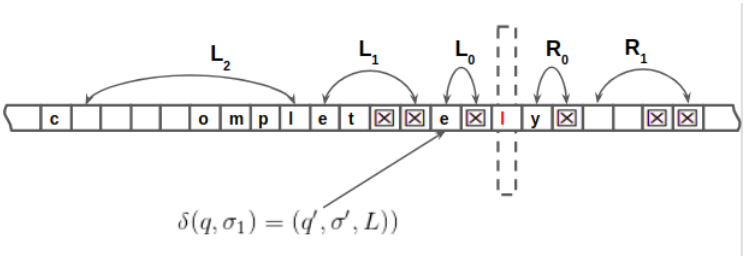


Figure 11: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.

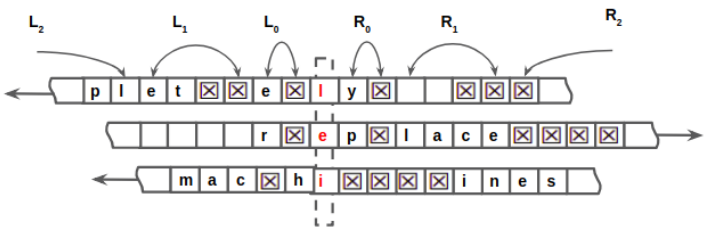


Figure 12: Simulating a machine M with three tapes using a machine $\widetilde{M}$ with a single tape.