MULTITAPE TURING MACHINES

Mihai-Valentin DUMITRU
mihai.dumitru2201@upb.ro

October 2023

We introduce a new model of computation – a Turing Machine with k heads, each positioned over one of k tapes, for some natural number $k \geq 2$.

At first glance, this model might seem "more powerful" than a single-tape machine: it can make use of information written on multiple tapes at once. Indeed, there are some problems which can be solved *more efficiently* on a multitape Turing Machine.

However, from the point of view of computability, we shall soon see that there is no difference in "computational power": the classes of problems that can be *solved*[†] by a k-tape[‡] machine are exactly those that can be *solved* by a single-tape machine.

Then why bother with multitape Turing Machines in the context of Computability Theory? The main reason is that they can make some proofs easier to follow. Our next relevant goal is to show the existence of a *"Universal Turing Machine"*, a single machine that can *solve* any function that is *solvable*. That construction will involve a Turing Machine with 3 tapes, for ease of visualization.

We now informally describe how a multitape machine can be visualized and how it works:

- There is still a single control unit of the machine, a single internal state.
- There are k different tapes (imagine them laid out vertically, one above the other)
- for each tape, there is a read/write head which, at any step of computation, is positioned over some cell of that tape.
- Depending on the combination of internal state and *the k-tuple of currently read symbols*, the machine makes some transition.
- When transitioning, the machine does three things:
  1. It goes to a new internal state; or remains in the same one.
  2. For each of its heads, it writes some new symbol on the cell under it; or leaves that cell unchanged.

     More explicitly, the machine writes a new symbol on each tape, not necessarily the same symbol on all tapes.
  3. It moves each head, independently, to point to the cell immediately on the left or right of the current cell; or leaves it where it is.

     More explicitly, the machine moves each head at most one position, not necessarily all heads in the same direction.

We will now quickly go through some formal definitions to bring us to the same point where we left the single-tape machine (e.g. configurations, acceptance, decision etc.)

If you understood the definitions for the single-tape machine, these should seem to you very straightforward, so few further explanations are provided and you can probably skip straight to section 5.

[†]Remember that by "*solve*" we mean one of: accept, decide, compute
[‡]Regardless of how high the value of k is.

1 Formal definition

Definition 1.1. A multitape Turing Machine is a 6-tuple: $(Q, \Sigma, \Gamma, B, q_1, \delta)$.

- The first five members: $Q, \Sigma, \Gamma, B, q_1$ – have the same significance as for a single-tape Turing Machine.
- $\boxed{\delta : Q \times \Gamma^k \to Q' \times (\Gamma \times \{\leftarrow, -, \rightarrow\})^k}$

2 Configurations

Definition 2.1. A *configuration* of a k-tape Turing Machine $M = (Q, \Sigma, \Gamma, B, q_1, \delta)$ is a $(k+1)$-tuple $((\gamma_L^1, \gamma_R^1), (\gamma_L^2, \gamma_R^2), ...(\gamma_L^k, \gamma_R^k), \mathbf{q})$ where $\mathbf{q}$ is the current state and each γ_L^i and γ_R^i encode the contents to the left and right of the head, for the i^{th} tape.

Definition 2.2. The *initial configuration* of a k-tape Turing machine $M = (Q, \Sigma, \Gamma, B, q_1, \delta)$ on input w is $((\varepsilon, w), (\varepsilon, \square), ..., (\varepsilon, \square), q_1)$.

We shall call it $C_0^M(w)$.

This tells us that, initially, the input is written on the first tape and all the others are empty.

Definition 2.3. A *halting configuration of a k-tape Turing machine $M = (Q, \Sigma, \Gamma, B, q_1, \delta)$ on input w* is $((\phi^1, \mu^1), (\phi^2, \mu^2), ..., (\phi^k, \mu^k), q_f)$, where ε is the empty string, $\phi^i, \mu^i \in \Gamma^*$, for $i = \overline{1, k}$ and $q_f \in \{Y, N, H\}$. Specifically, an *accepting configuration* is one where $q_f = Y$ and a *rejecting configuration* is one where $q_f = N$.

3 Going from one configuration to another

Definition 3.1. Let $M = (Q, \Sigma, \Gamma, B, q_1, \delta)$ be a k-tape Turing Machine.

We say that $C \vdash_M C'$ (read "from configuration C, machine M goes to configuration C' in one step") if, in a single transition, the machine goes from configuration C to configuration C'.

Remember that a configuration is a tuple consisting of k pairs of strings and a state. For each k pair, depending on the transition function δ, the following changes are applied to determine the next configuration:

1. $\delta(\mathbf{p}, \mathtt{a}) = (\mathbf{s}, \mathtt{b}, -)$:
   1.1. $(\phi, \mathtt{a}\mu) \vdash_M (\phi, \mathtt{b}\mu)$
2. $\delta(\mathbf{p}, \mathtt{a}) = (\mathbf{s}, \mathtt{b}, \rightarrow)$:
   2.1. $(\phi, \mathtt{ac}\mu) \vdash_M (\phi\mathtt{b}, c\mu)$
   2.2. $(\phi, \mathtt{a}) \vdash_M (\phi\mathtt{b}, \square)$
3. $\delta(\mathbf{p}, \mathtt{a}) = (\mathbf{s}, \mathtt{b}, \leftarrow)$:
   3.1. $(\phi c, \mathtt{a}\mu) \vdash_M (\phi, \mathtt{cb}\mu)$
   3.2. $(\varepsilon, \mathtt{a}\mu) \vdash_M (\varepsilon, \square\mathtt{b}\mu)$

Where:

- $\phi, \mu \in \Gamma'^*$ (possibly-empty strings of tape symbols)
- $\mathtt{a}, \mathtt{b}, \mathtt{c} \in \Gamma$
- $\mathbf{q}, \mathbf{s} \in Q$

We abuse notation here and use a specialized version of concatenation where concatenating a blank to the end of the empty string results in the empty string: $\varepsilon\square = \varepsilon$.

Combined with our specification of "well-behaved machines", this convention keeps our rules succinct (consider the specifications for the transitions that move the head right).

The next configuration will consist of all k pairs obtained from applying the rules, together with the new state $\mathbf{s}$.

4 How multitape Turing Machines run

In all definitions in this section:

- M is a k-tape Turing Machine $(Q, \Sigma, \Gamma, B, q_1, \delta)$
- $w, v \in \Sigma^*$
- $\phi^i \in \Gamma'^*, i = \overline{1, k}$
- $\mu^i \in \Gamma^*, i = \overline{1, k}$ (only the first symbol may be a B)

Definition 4.1. $M[w] \to TRUE \overset{\text{def}}{=} C_0^M(w) \vdash_M^* ((\phi^1, \mu^1), (\phi^2, \mu^2), ..., (\phi^k, \mu^k), Y)$

Definition 4.2. $M[w] \to FALSE \overset{\text{def}}{=} C_0^M(w) \vdash_M^* ((\phi^1, \mu^1), (\phi^2, \mu^2), ..., (\phi^k, \mu^k), N)$

Definition 4.3. $M[w] \to v \overset{\text{def}}{=} C_0^M(w) \vdash_M^* ((\phi^1, \mu^1), (\phi^2, \mu^2), ..., (\phi^k, \mu^k), H)$ and:

- $\phi^k\mu^k = v,$ if $v \neq \varepsilon$
- $\phi^k = \varepsilon, \mu^k = B,$ if $v = \varepsilon$

The output of a k-tape machine consists in the tape contents of the k-th tape.

The rest of the definitions in Section 4 of the previous lecture are identical for multitape Turing Machines. Same for definitions 5.1, 6.1 and 7.1.

5 Machine equivalence

Before we prove that for each multitape Turing machine, there exists an equivalent single-tape one, we need to specify what we mean by "equivalence" between two machines.

Definition 5.1. Let M_1 and M_2 be two Turing Machines with input alphabets Σ_1 and Σ_2 respectively; and a word expressible in both alphabets, $w \in \Sigma_1 \cap \Sigma_2$:

$$\begin{aligned} M_1[w] \equiv M_2[w] \overset{\text{def}}{=} & (M_1[w] \to X \wedge M_2[w] \to X) \vee \\ & (M_1[w] \to TRUE \wedge M_2[w] \to TRUE) \vee \\ & (M_1[w] \to FALSE \wedge M_2[w] \to FALSE) \vee \\ & (M_1[w] \to \perp \wedge M_2[w] \to \perp) \end{aligned}$$

where X is a word expressible in both tape alphabets $X \in (\Gamma_1' \cap \Gamma_2')^*$.

Read this as: *"on input w, M_1 and M_2 yield the same result"*.

The term "result" here is a bit of a stretch, because the machines could also not halt. This is why formal definitions are important, to avoid confusion caused by the vagueness of natural terms.

So two machines "yield the same result" on an input if one of these holds true:

- both machines *accept* the input, by eventually transitioning to state Y
- both machines *reject* the input, by eventually transitioning to state N
- both machines eventually transition to state H leaving the same sequence of non-blanks on the tape
- both machines continue computation forever, never halting

Definition 5.2. For two machines M_1 and M_2 with the same input alphabet Σ:

$$M_1 \equiv M_2 \overset{\text{def}}{=} \forall w \in \Sigma^*, M_1[w] \equiv M_2[w]$$

6 Equivalence with single-tape Turing Machines

We claimed that multitape machines and single-tape machines are computationally equivalent. We now formulate this as a theorem and prove it.[§]

Theorem 6.1. For any k-tape machine: $M_m = (Q_m, \Sigma, \Gamma_m, B_m, q_{1_m}, \delta_m)$, there exists a single-tape machine $M_s = (Q_s, \Sigma, \Gamma_s, B_s, q_{1_s}, \delta_s)$ such that $M_m \equiv M_s$.

Note that we're only talking about machines with the same input alphabet.

We devise a precise *simulation* of M_m's behavior: M_s will mimic the effect of each transition of M_m on the input. For each transition that M_m makes, M_s will need to make *several* (in fact, quite a large number of) transitions to simulate it.

We will keep the contents of all k tapes concatenated onto the single tape, separated by a new symbol $\diamond$ that we introduce ($\diamond \notin \Gamma_m, \diamond \notin \Sigma$).

We also need to keep track of where each of the k heads are placed onto their corresponding tape. To do this, for each symbol $g \in \Gamma_m$, we introduce a new symbol $\underline{g}$ that stands for "the head being positioned over symbol g".

M_s uses blanks to delimit the relevant portion of the tape; so does M_m for its k tapes. To avoid confusion, we distinguish between *"the blank symbol of M_m"* and *"the blank symbol of M_s"*.

[§]We will actually prove a stronger result: not only can we construct a single-tape machine that gives the same *result*, but one that can simulate every step of the multitape machine's computation. However, this is not relevant for our purposes, so we won't focus on it.

For example, if $\Gamma_m = \{0, 1, \square_m\}$, then Γ_s will contain at least: $0, 1, \square_m, \underline{0}, \underline{1}, \underline{\square_m}, \diamond, \square$.

For each state q of M_m we will need several states in M_s. First we will make a simple copy; the copy of q_{1_m} will be q_{1_s}, the initial state of M_s.

The very first thing M_s needs to do when started on input w is to recreate the initial configuration of $M_m[w]$.

For example, consider the input 10011 and $k = 4$. M_s will start with the following tape:

$$...\square\square\square 10011 \square\square\square...$$

Using $2k + 3$ auxiliary states, M_s will modify the tape to look like this:

$$...\square\square\square \diamond \underline{1}0011 \diamond \underline{\square_m} \diamond \underline{\square_m} \diamond \underline{\square_m} \diamond \square\square\square...$$

Then it will move its head over the first underlined symbol; in this example, the leftmost 1. From here, the simulation of M_m's transitions begins.

M_s will simulate each transition of M_m, one at a time; for ease of reference, we shall refer to the simulation of one transition of M_m as a *"phase"*. Each phase consists of the following steps:

1. At the start of a phase, M_s will be in some state q_copy, corresponding to a state q of M_m and the head of M_s will be positioned over the leftmost underlined symbol – i.e. where the first head of M_m is.

   M_s needs to go rightward through the whole tape, finding the k symbols marked with an underline (which indicate the position of one of the k heads of M_m).

   Each time it finds the position of one of M_m's heads, it needs to *memorize* the symbol under it, in order to build the current "symbol k-tuple" which determines the transition. Remember that memorization of a fixed amount of information can be achieved using states:

   - from state q_copy, when we encounter a relevant symbol $\underline{g_1}$ ($g_1 \in \Gamma_m$), we transition to one of $|\Gamma_m|$ possible states $q\_copy_{g_1}$

   - from each of the $|\Gamma_m|$ states $q\_copy_{g_1}$, when we encounter a relevant symbol $\underline{g_2}$, we transition to one of $|\Gamma_m|$ possible states $q\_copy_{g_1 g_2}$

   - etc.

   For other symbols, remain in the same state and move right.

   In summary, for each state q of M_m we need:

   - a copy of it, q_copy

   - $|\Gamma_m|$ new states $q\_copy_{g_1}$

   - $|\Gamma_m|^2$ new states $q\_copy_{g_1 g_2}$

   - $|\Gamma_m|^3$ new states $q\_copy_{g_1 g_2 g_3}$

   - ...

   - $|\Gamma_m|^k$ new states $q\_copy_{g_1 g_2 g_3 \ldots g_k}$

   A total of $|Q_m|(1 + |\Gamma_m| \frac{(|\Gamma_m|^k - 1)}{|\Gamma_m| - 1})$ new states.

2. Once we reach a state of the form $q\_copy_{g_1 g_2 g_3 \ldots g_k}$, we have to simulate the transition given by

   $$\delta_m(q, (g_1, g_2, ..., g_k)) = (s, (c_1, D_1), (c_2, D_2), ..., (c_k, D_k))$$

   (Where $c_1, c_2, ..., c_k \in \Gamma_m$ and $D_1, D_2, ..., D_k \in \{\leftarrow, -, \rightarrow\}$.)

We will do this by going back through the tape and performing the necessary changes for each head, by first transitioning into a new state $pc\_s^k_{g_1g_2g_3...g_k}$.

The k superscript signifies that we are now performing the changes for the k^{th} head; after this, we'll transition to $pc\_s^{k-1}_{g_1g_2g_3...g_k}$ and go left until we find the $(k-1)^{th}$ head and so on.

The s shows that we need such a collection of new states for each state s of M_m; after M_s will perform all symbol modifications, it will transition to the simple copy of s and start a new phase.

So for each of the $|Q||\Gamma_m|^k$ states $q\_copy_{g_1g_2g_3...g_k}$, we add $k|Q'|$ more states.

3. Each head can only move at most one position to the left or to the right, so our changes will be *localized* on the cell with the marked symbol and its two immediate neighbors.

   For example, consider the following tape section of M_s that corresponds to some tape i of M_m:

   $$... \diamond 001010\underline{0}11011101 \diamond ...$$

   The i^{th} head of M_m will change the symbol currently read (or leave it as it is) and move the head onto the cell to the left/right or keep it where it is. If the head moves, we need to change the current symbol to its non-underlined variant, and change the corresponding symbol (either the one to the left, or the one to the right, depending on where M_m's i^{th} head will go), to its underlined variant.

   But nothing else on this i^{th} tape can change during this one transition; only three symbols, at most, will differ, but the rest will be the same:

   $$... \diamond 0010???1011101 \diamond ...$$

   We now describe the transitions for states $pc\_s^j_{g_1g_2g_3...g_k}$ with $2 \le j \le k$.

   Let $\delta_m(q,(g_1,g_2,...,g_k)) = (s,(c_1,D_1),(c_2,D_2),...,(c_k,D_k))$.

   From state $pc\_s^j_{g_1g_2g_3...g_k}$, on symbol $\underline{g_j}$ we write:

   - $\underline{c_j}$, if $D_j = -$
   - c_j, otherwise

   If $D_j = \rightarrow$ we move right into a state $auxR\_s^j_{g_1g_2g_3...g_k}$ for which

   $$\delta(auxR\_s^j_{g_1g_2g_3...g_k}, h) = (pc\_s^{j-1}_{g_1g_2g_3...g_k}, \underline{h}, \leftarrow), \forall h \in \Gamma_m$$

   Simply put, we move one cell to the right and underline whatever is there to mark that that's where the current (j^{th}) head moved; we then move on to find the next head.

   Similarly for $D_j = \leftarrow$, we move left and use a state of the form $auxL$.

   If $D_j = -$, we move left and change state to $pc\_s^{j-1}_{g_1g_2g_3...g_k}$.

   So for each of these $|Q||\Gamma_m|^k(k-1)|Q'|$ states we add 2 more states.

4. For $j = 1$, auxiliary states of the form $auxL$ and $auxR$ will transition to state s_{copy}.

   An additional $2|Q||\Gamma_m|^k|Q'|$ states.

   This concludes the current phase.

It follows, by inductive reasoning, that M_s correctly simulates the behavior of M_m.

M_s then has to simply delete all symbols from its tape, except those strictly between the last two $\diamond$s. This can be achieved with just a few extra steps. Now, M_s has the same output as M_m.

6.1 Shifting the tape contents

A complication arises when one of the simulated k heads is either on the first or the last *non*-$\square_m$ symbol, i.e. it's immediately to the left/right of a $\diamond$. For example, if the head is on the first symbol and then it moves left, it will end up on our special delimitator $\diamond$. However, from M_m's point of view, the head should find here a blank symbol. So we need to add one on demand; and we can do so by shifting the rest of our tape's contents, one cell to the right.

$$10001\underline{1}01 \diamond 1\underline{0}11 \diamond \underline{0}11000001 \diamond 1101\underline{0}11\diamond$$

$$10001\underline{1}01 \diamond 1\underline{0}11 \diamond \square_m\underline{0}11000001 \diamond 1101\underline{0}11\diamond$$

$$10001\underline{1}01 \diamond 1\underline{0}11 \diamond \underline{\square_m}011000001 \diamond 1101\underline{0}11\diamond$$

To shift tape symbols, we first need to mark the place where we want to create a "hole". For this we can employ an additional $2(|\Gamma_m| - 1) + 1$ symbols (the factor of 2 because we have to address the underlined versions too; the "-1", because we'll never have to shift if there's already a blank symbol there; the added 1 because of the $\diamond$). You can imagine these symbols as a "primed" version of the original; e.g. $1'$, $0'$, $\diamond'$, etc.

We then need one state to find the end of the tape contents, one state for each symbol (including "primed" versions) to copy it and another state to go left to the next symbol to copy.

All the while, we need to remember what we were doing when the need to move tape contents to the right appeared. We need to add these extra states for each of the $auxR\_s^j_{g_1g_2g_3\dots g_k}$ and $auxL\_s^j_{g_1g_2g_3\dots g_k}$, $j = \overline{1,k}$ states.

Note 6.1. Our single-tape simulating machine seems very large compared to the original multitape machine. We had to add a lot of tape symbols and a lot of states.

The number of steps involved to simulate just one step of the multitape machine also seems intimidating.

For now, we are not concerned at all with "speed", or "efficiency"; we just want to show that we can do this simulation in whatever way possible.

When we move on to Complexity Theory, we will be interested in such measurements; surprisingly, we'll see a reasonable framework of judging performance, by which this simulation is not so bad.