Lecture #11: Multi-Tape Turing Machines, Nondeterministic Turing Machines, and the Church-Turing Thesis
Multi-Tape Turing Machines That Compute Functions

Let Σ_1 and Σ_2 be alphabets — finite and nonempty sets — and suppose that $\sqcup$ does not belong to either Σ_1 or Σ_2. Consider a (partial or total) function $f : \Sigma_1^\star \to \Sigma_2^\star$. If k is a positive integer then a ***k-tape Turing machine M, that computes the function $f : \Sigma_1^\star \to \Sigma_2^\star$*** can be defined. Formally,

$$M = (Q, \Sigma_1, \Sigma_2, \Gamma, \delta, q_0, q_{\text{halt}})$$

where Σ_1 and Σ_2 are as described above and where Q, Γ, δ, q_0 and q_{halt} satisfy the following conditions.

- M has a finite set Q of states just as a (1-tape) Turing machine that computes a function — so that Q includes a ***start state*** q_0 and a ***halt state*** q_{halt}. We will now require that $q_0 \neq q_{\text{halt}}$. As argued below, this does not significantly limit the computational power of these machines.

- M's ***tape alphabet*** is a finite set Γ such that

$$\Sigma_1 \cup \Sigma_2 \cup \{\sqcup\} \subseteq \Gamma,$$

  just as for a (1-tape) Turing machine that computes a function $f : \Sigma_1^\star \to \Sigma_2^\star$. As before, we will require that $Q \cap \Gamma = \emptyset$.

- As for k-tape Turing machines that recognize or decide languages, the ***transition function*** is a partial function

$$\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k.$$

  We will require that $\delta(q, \sigma_1, \sigma_2, \dots, \sigma_k)$ is defined for all $\sigma_1, \sigma_2, \dots, \sigma_k \in \Gamma$ whenever $q \in Q$ and $q \neq q_{\text{halt}}$, and we will require that $\delta(q_{\text{halt}}, \sigma_1, \sigma_2, \dots, \sigma_k)$ is ***undefined*** for all $\sigma_1, \sigma_2, \dots, \sigma_k \in \Gamma$.

- Configurations are as described for k-tape Turing machines that recognize languages, and these are represented as strings of symbols over the alphabet $Q \cup \Gamma \cup \{\sharp\}$, where $\sharp$ is a symbol such that $\sharp \notin Q \cup \Gamma$. Transition functions are applied to (non-halting) transitions in the same way as they are for k-tape Turing machines that recognize languages, as well.

- The ***initial configuration*** for an input string $\omega \in \Sigma_1^\star$ is the same as for multi-tape Turing machines that recognize languages: ω is written on the leftmost cells of the first tape, with an infinite number of copies of $\sqcup$ to its right. All other tapes are filled with $\sqcup$'s, and all tape heads are located at the leftmost cells of their tapes. The Turing machine is in its start state, so that the initial configuration for ω could be the string

  $$q_0\omega\sharp q_0\sharp q_0\sharp \dots \sharp q_0$$

  consisting of $q_0\omega$, followed by $k-1$ copies of $\sharp q_0$.

- If $f(\omega)$ is defined, for an input string $\omega \in \Sigma_1^\star$ then, if $\mu = f(\omega) \in \Sigma_2^\star$, then M's computation on ω should end in a halting state, where the machine is in state q_{halt} and

  - for $1 \le i \le k-1$, the i^{th} tape is filled with blanks, with the tape head resting at the leftmost cell of the tape, and
  - $\mu = f(\omega)$ is stored at the leftmost cells of the k^{th} tape, with an infinite number of $\sqcup$'s to the right, and with the tape head resting at the leftmost cell of the tape as well.

  Thus this final configuration would be represented by the string

  $$q_{\mathsf{halt}}\sharp q_{\mathsf{halt}}\sharp \dots \sharp q_{\mathsf{halt}}\mu$$

  beginning with $k-1$ copies of the string $q_{\mathsf{halt}}\sharp$, and ending with the string $q_{\mathsf{halt}}\mu$.

  Note that the output is written on the ***last*** tape — not the first.

- If $f(\omega)$ is not defined, for an input string $\omega \in \Sigma_1^\star$, then M loops on input ω.

Since these Turing machines are quite similar to k-tape Turing machines that recognize languages, an example will not be given here. With that noted, the following claims can be proved.

Claim #1: Let Σ_1 and Σ_2 be alphabets (such that $\sqcup \notin \Sigma_1$ and $\sqcup \notin \Sigma_2$), let $f : \Sigma_1^\star \to \Sigma_2^\star$, and let k be a positive integer. If there exists a (1-tape) Turing machine M_1 that computes the function f then there exists a k-tape Turing machine M_2 that computes the function f as well.

Sketch of Proof. Suppose, first that M_1's start state, q_0, is equal to its halt state, q_{halt}. Then M_1 is easily modified, so that this is not the case, by adding a new start state, $\widehat{q}_0$ — and extending M_1's transition function, δ, by setting $\delta(\widehat{q}_0, \sigma)$ to be $(q_0, \sigma, \mathrm{L})$ for every symbol σ in M's tape alphabet: This simply adds one initial move that goes to M_1's "original" start configuration, so that the function computed by the machine has not been changed. We may therefore assume that M_1's halt state is different from its start state.

If $k = 1$ then it suffices to set M_2 to be M_1. Suppose, therefore, that $k \geq 2$.

Since M_2's output should be on its k^{th} tape, instead of the first, it is *not* sufficient for M_2 to simply simulate M_1, ignoring all but its first tape. Instead one more symbol should be added to M_2's tape alphabet, (that is, M_2's tape alphabet should consist of M_1's tape alphabet, Γ_1, along with one new symbol, $\mathtt{X}$, such that $\mathtt{X} \notin \Gamma_1$). M_2 should then carry out the following process.

1. Write a copy of the input ω, on the k^{th} tape. Then erase the first tape (so that it is filled with $\sqcup$'s) and move the tape heads for the first and k^{th} tapes back to their initial positions — without changing tapes $2, 3, \ldots, k-1$ at all. (The new symbol, $\mathtt{X}$, can be used to mark the leftmost cell at the beginning of this step, in order to make this easy to do.)

2. Simulate M_1 — using the k^{th} tape instead of the first tape, and ignoring all the others. Halt if (and when) M_1 would.

The first stage of this process can certainly be carried out using a number of steps that is at most linear in the length of the input string. In the second step, only one step of M_2 is needed to simulate each step of M_1 — and, since the k^{th} tape will store any output that has been generated, it is easily proved that M_2 computes the same function as M_1 does, as required. $\square$

Claim #2: Let Σ_1 and Σ_2 be alphabets (such that $\sqcup \notin \Sigma_1$ and $\sqcup \notin \Sigma_2$), let $f : \Sigma_1^\star \to \Sigma_2^\star$, and let k be a positive integer. If there exists a k-tape Turing machine M_1 that computes f then there exists a (1-tape) Turing machine M_2 that computes f as well.

Sketch of Proof. If $k = 1$ then it suffices to set M_2 to be M_1. Suppose, therefore, that $k \geq 2$.

Consider the simulation of a k-tape Turing machine that was described in the lecture notes to prove Claim #2 (so that it includes symbols with $2k$ "tracks" as described in the notes).

- Suppose that M_2's tape alphabet is as described in that proof and that M_2's tape is used to represent the contents of M_1's tapes, and the location of M_1's tape heads, as described in the notes, as well.

- A simulation will have the same ***initialization*** phase and ***step-by-step simulation*** stage — except that M_2 will not halt at the same time as M_1 — can be used.

- A new ***cleanup*** stage is now required: When M_1 would halt, M_2 includes a representation of M_1's tapes, with the first $k-1$ tapes filled with $\sqcup$'s and with the desired output,

$f(\omega)$ on the k^{th} tape — and with the special symbol \$ at the leftmost cell. During the cleanup stage M_2's tape should be changed, so that it simply stores $f(\omega)$, with $\sqcup$'s to the right of this, and with the tape head at the leftmost cell of the tape, instead.

The details of this are similar to the details of the initialization phase — and completing these is left as an ***exercise***.

A consideration of the description of a "k-tape Turing machine that computes a function" should now be sufficient to confirm that M_2 is a (one-tape) Turing machine that computes the same function as M_1, as needed to prove the claim. $\square$

Having multi-tape Turing machines that recognize (or decide) languages and that compute functions simplify Turing machine design, because it makes it easier to imagine, and implement, algorithms that use other algorithms as ***subroutines***. If time allows this will be considered when the tutorial exercise for this topic is completed.