Lecture #12: Multi-Tape Turing Machines, Nondeterministic Turing Machines, and the Church-Turing Thesis

Lecture Presentation

Review of Preparatory Material

Designing a Multi-Tape Turing Machine for Additional of Binary Numbers

During the previous lecture (and in a supplemental document for it) ***Turing machines that compute functions*** were introduced. Now that ***multi-tape Turing machines*** (that recognize languages) have been introduced, ***multi-tape Turing machines that compute functions*** can be introduced too — and a supplemental document with information about them is available.

Let $\Sigma_{\text{binary}} = \{0, 1\}$, and let $f_{\text{b\_inc}} : \Sigma^\star_{\text{binary}} \to \Sigma^\star_{\text{binary}}$ such that the following properties are satisfied for every string $\omega \in \Sigma^\star_{\text{binary}}$:

- If ω is the unpadded binary representation of a non-negative integer n then $f_{\text{b\_inc}}(\omega)$ is the unpadded binary representation of $n + 1$.
- On the other hand, if ω is *not* the unpadded binary representation of any non-negative integer, then $f_{\text{b\_inc}}(\omega) = \lambda$, the empty string.

Let $f_{\text{b\_dec}} : \Sigma^\star_{\text{binary}} \to \Sigma^\star_{\text{binary}}$ such that the following properties are satisfied for every string $\omega \in \Sigma^\star_{\text{binary}}$:

- If ω is the unpadded binary representation of a *positive* integer n, then $f_{\text{b\_dec}}(\omega)$ is the unpadded binary representation of $n - 1$.
- On the other hand, if ω is *not* the unpadded binary representation of any positive integer, then $f_{\text{b\_dec}}(\omega) = \lambda$, the empty string.

In the supplemental document for the previous lecture about Turing machines that compute functions, a Turing machine that computes the above function $f_{\text{b\_inc}}$ was presented, along with a proof of its correctness. The document ended with an exercise; if the exercise was completed then a Turing machine that computes the function $f_{\text{b\_dec}}$ was obtained, and proved to be correct, as well.

Now let $\Sigma_{\text{pair}} = \{0, 1, \#\}$ and consider a function $f_{\text{add}} : \Sigma^\star_{\text{pair}} \to \Sigma^\star_{\text{binary}}$ such that the following properties are satisfied for every string $\omega \in \Sigma^\star_{\text{pair}}$:

- If $\omega = \mu \# \nu$, where $\mu, \nu \in \Sigma^\star_{\text{binary}}$ are the unpadded binary representations of a pair of non-negative integers n and m, respectively, then $f_{\text{add}}(\omega)$ is the unpadded binary representation of their sum, $n + m$.
- Otherwise $f_{\text{add}}(\omega) = \lambda$, the empty string.

Consider the algorithm shown on the following page.

On input $\omega \in \Sigma_{\text{pair}}$:

1. If ω has the form $\mu \# \nu$, where μ and ν are the unpadded binary representations of non-negative integers then write μ onto Tape #2 and write ν onto Tape #3, erasing Tape #1 and moving all tape heads to their leftmost positions. That is, go from the configuration

$$q_0 \, \omega \, \sharp \, q_0 \, \sharp \, q_0 \;\; = \;\; q_0 \, \mu \# \nu \, \sharp \, q_0 \, \sharp \, q_0$$

to a configuration

$$q_1 \, \sharp \, q_1 \, \mu \, \sharp \, q_1 \, \nu$$

and continue to the next step. On the other hand, if ω does not have this form then erase the first tape, moving the tape head back to its leftmost position, and halt — that is, go from the configuration $q_0 \, \omega \, \sharp \, q_0 \, \sharp \, q_0$ to the configuration

$$q_{\text{halt}} \, \sharp \, q_{\text{halt}} \, \sharp \, q_{\text{halt}}$$

Let n and m be the non-negative integers whose representations are on Tapes #2 and #3, respectively.

2. `while` $(n > 0)$ `{`
3. $n := n - 1$ (updating Tape #2 to make this change)
4. $m := m + 1$ (updating Tape #3 to make this change)

   `}`
5. Erase the copy of "0" that is now on Tape #2 — so that the tape head for this points to its leftmost cell — and go to state q_{halt}, in order to return the unpadded binary representation of m.

Figure 1: An Algorithm to Compute the Function f_{add}

Suppose, in particular, that this is being implemented using a 3-tape Turing machine, with input alphabet Σ_{pair}, whose tape alphabet also includes "dotted" copies of the symbols in Σ_{pair}, that is, symbols $\dot{0}$, $\dot{1}$, and $\dot{\#}$.

Proving the Correctness of This Algorithm

Implementing Step 1

For a non-empty string $\zeta = \alpha_1\alpha_2\alpha_3\ldots\alpha_k \in \Sigma^\star_{\text{binary}}$ (and for $\alpha_1, \alpha_2, \ldots, \alpha_k \in \Sigma_{\text{binary}}$), let ζ_M be the string in $\Gamma^\star$ obtained by marking the leftmost symbol — so that $\zeta_M = \dot{\alpha_1}\alpha_2\alpha_3\ldots\alpha_k$.

How Does This Concern the “Church-Turing Thesis”?